SYNOPSYS®

DesignWare® DW_axi_gm

Databook

DW_axi_gm – Product Code

2.04a
March 2020

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

Synopsys, Inc.
www.synopsys.com

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.01b onward.

Date	Release	Description
2.04a	March 2020	■ Revision version change for 2020.03a release ■ Added Appendix 8, “Basic Core Module (BCM) Library” ■ Updated synthesis results in “Performance” on page 77 ■ “Signal Descriptions” on page 47, “Parameter Descriptions” on page 39, and “Internal Parameter Descriptions” on page 71 auto-extracted from the RTL
2.03a	February 2018	■ Revision version change for 2018.02a release ■ Updated synthesis results in “Performance” on page 77 ■ Removed Chapter 2 Building and Verifying a Component or Subsystem from the databook and added the contents in the newly created user guide. ■ “Signal Descriptions” on page 47, “Parameter Descriptions” on page 39, and “Internal Parameter Descriptions” on page 71 auto-extracted from the RTL with change bars
2.02a	March 2016	■ Revision version change for 2016.03a release ■ Added “Running SpyGlass® Lint and SpyGlass® CDC” section ■ Added “Running Spyglass on Generated Code with coreAssembler” section ■ “Signal Descriptions” on page 47 and “Parameter Descriptions” on page 39 auto-extracted from the RTL ■ Added “Internal Parameter Descriptions” on page 71 ■ Modified area and power numbers in “Performance” on page 77
2.01a	October 2014	■ Version change for 2014.10a release. ■ Added “Integration Considerations” chapter.
2.00a	June 2013	Added information about new and modified signals and parameters to support AMBA 4 AXI specifications.
1.06a	May 2013	Added sideband signals and parameters. Added timing diagrams showing sideband signals. Corrected the parameter names for AXI and GIF address width, data width, and ID width. Updated the template.

Date	Release	Description
1.05a	Oct 2012	Added the product code on the cover and in Table 1-1.
1.05a	Jun 2012	Version change for 2012.06b release.
1.04b	Oct 2011	Modified information on unlimited outstanding transactions; modified graphics for gclk being derived from gclken; corrected gclken input delay.
1.04a	Jun 2011	Updated system diagram in Figure 1-1; enhanced “Related Documents” section in Preface.
1.04a	Jan 2011	Corrected sresp waveform for Figure 3-3; added “Outstanding Transaction Limits” section; changed delays for most GIF Request Channel signals.
1.03a	Sep 2010	Added information for locked sequence support; corrected names of include files and vcs command used for simulation.
1.02a	Dec 2009	Corrected READ_BLOCK parameter name to GM_BLOCK_READ, WRITE_BLOCK parameter name to GM_BLOCK_WRITE; updated GM_REQ_BUFFER parameter dependencies; updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.02a	May 2009	Removed references to QuickStarts, as they are no longer supported.
1.02a	Oct 2008	Version change for 2008.10a release.
1.01c	Jun 2008	Version change for 2008.06a release.
1.01b	Jul 2007	Updated parameter names that changed from GMX* to GM*.
1.01b	Jun 2007	Version change for 2007.06a release.

Preface

This databook provides information that you need to interface the DW_axi_gm component to the Synopsys DesignWare Synthesizable Components for AMBA environment. This component conforms to the AMBA 3 AXI and AMBA 4 AXI specifications defined in the *AMBA AXI and ACE Protocol Specification* from ARM.

The information in this databook includes a functional description, and signal and parameter descriptions, as well as information on how you can configure, create RTL for, simulate, and synthesize the component using coreConsultant. Also provided are an overview of the component testbench, a description of the tests that are run to verify the coreKit, and synthesis information for the coreKit.

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_axi_gm.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_axi_gm.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_axi_gm signals.
- Chapter 5, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals and Parameters chapters.
- Chapter 6, "Verification" provides information on verifying the configured DW_axi_gm.
- Chapter 7, "Integration Considerations" includes information you need to integrate the configured DW_axi_gm into your design.
- Appendix A, "Glossary" provides a glossary of general terms.

Related Documentation

- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI/AMBA 4 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA 3 AXI and AMBA 4 AXI, see the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI*.

Web Resources

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

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

- Prepare the following debug information, if applicable:
  - For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the <*core tool startup directory*>/debug.tar.gz file.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD).
    - Identify the hierarchy path to the DesignWare instance.
    - Identify the timestamp of any signals or locations in the waveforms that are not understood.
- *For the fastest response*, enter a case through SolvNetPlus:

  a. https://solvnetplus.synopsys.com

  Note SolvNetPlus does not support Internet Explorer. Use a supported browser such as Microsoft Edge, Google Chrome, Mozilla Firefox, or Apple Safari.

  b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).

  c. Complete the mandatory fields that are marked with an asterisk and click **Save**.

  Ensure to include the following:
  - **Product L1:** DesignWare Library IP
  - **Product L2:** <name of L2>

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

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

  https://solvnetplus.synopsys.com/s/article/SolvNetPlus-Usage-Help-Resources

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
  - Include the Product L1 and Product L2 names, and Version number in your e-mail so it can be routed correctly.
  - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
  - Attach any debug files you created.
- Or, telephone your local support center:
  - North America:
    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:
    https://www.synopsys.com/support/global-support-centers.html

Product Code

Table 1-1 lists all the components associated with the product code for DesignWare AMBA Fabric.

Table 1-1 DesignWare AMBA Fabric – Product Code: 3768-0

Component Name	Description
DW_ahb	High performance, low latency interconnect fabric for AMBA 2 AHB
DW_ahb_eh2h	High performance, high bandwidth AMBA 2 AHB to AHB bridge
DW_ahb_h2h	Area efficient, low bandwidth AMBA 2 AHB to AHB Bridge
DW_ahb_icm	Configurable multi-layer interconnection matrix
DW_ahb_ictl	Configurable vectored interrupt controllers for AHB bus systems
DW_apb	High performance, low latency interconnect fabric & bridge for AMBA 2 APB for direct connect to AMBA 2 AHB fabric
DW_apb_ictl	Configurable vectored interrupt controllers for APB bus systems
DW_axi	High performance, low latency interconnect fabric for AMBA 3 AXI or AMBA 4 AXI
DW_axi_a2x	Configurable bridge between AMBA 3 AXI/AMBA 4 AXI components and AHB components or between AMBA 3 AXI/AMBA 4 AXI components and AMBA 3 AXI/AMBA 4 AXI components.
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_hmx	Configurable high performance interface from an AHB master to an AMBA 3 AXI or AMBA 4 AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI or AMBA 4 AXI subsystems

Table 1-1 DesignWare AMBA Fabric – Product Code: 3768-0

Component Name	Description
DW_axi_x2h	Bridge from AMBA 3 AXI or AMBA 4 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or busses

1 Product Overview

The DW_axi_gm is a configurable module between a generic interface (GIF) and the AMBA AXI bus. The DW_axi_gm component has both an AMBA AXI-compliant master interface and a simplified GIF. It is part of the family of DesignWare Synthesizable Components for AMBA.

The DW_axi_gm AXI master interface connects to an AXI bus — for example, the DW_axi Interconnect for AMBA AXI interfaces. The DW_axi_gm GIF connects to a third-party controller that initiates transactions to the DW_axi_gm.

1.1 DesignWare Synthesizable Components for AMBA System Overview

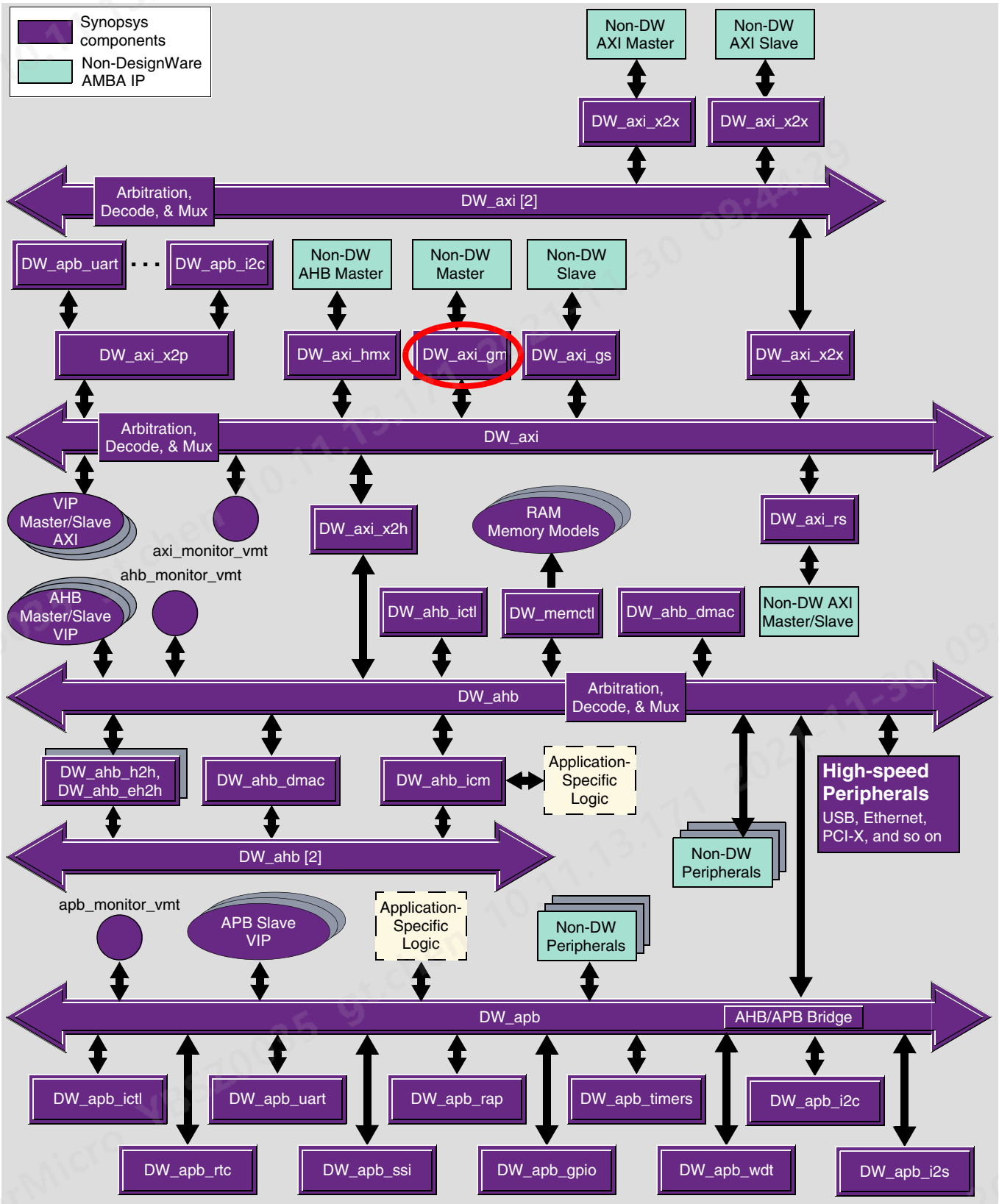
The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA 3 AXI/AMBA 4 AXI (Advanced eXtensible Interface) components.

Note You must have a DWC-AMBA-Fabric-Source license to enable the AXI4 interface.

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Attention Links resolve only if you are viewing this databook from your $DESIGNWARE_HOME tree, and to only those components that are installed in the tree.

Figure 1-1 Example of DW_axi_gm in a Complete System



You can connect, configure, synthesize, and verify the DW_axi_gm within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

If you want to configure, synthesize, and verify a single component such as the DW_axi_gm component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

The DesignWare DW_axi_gm Generic Interface (GIF)-to-AMBA AXI Module provides a method to transfer GIF-generated transactions to an AMBA AXI bus. It provides the subsystem designer with an easy way to reuse existing custom or third-party controllers in new designs that use the high-performance AMBA AXI bus. The DW_axi_gm uses a simplified interface, reducing the complexity of design required to port a custom or third-party controller to the AXI bus.

Functionally, the DW_axi_gm GIF receives transactions from an external, custom/third-party controller. Internal to the DW_axi_gm, these GIF transactions are translated into AXI transactions. The DW_axi_gm has an AXI master interface that initiates the translated GIF transactions on an AXI bus.

The GIF consists of:

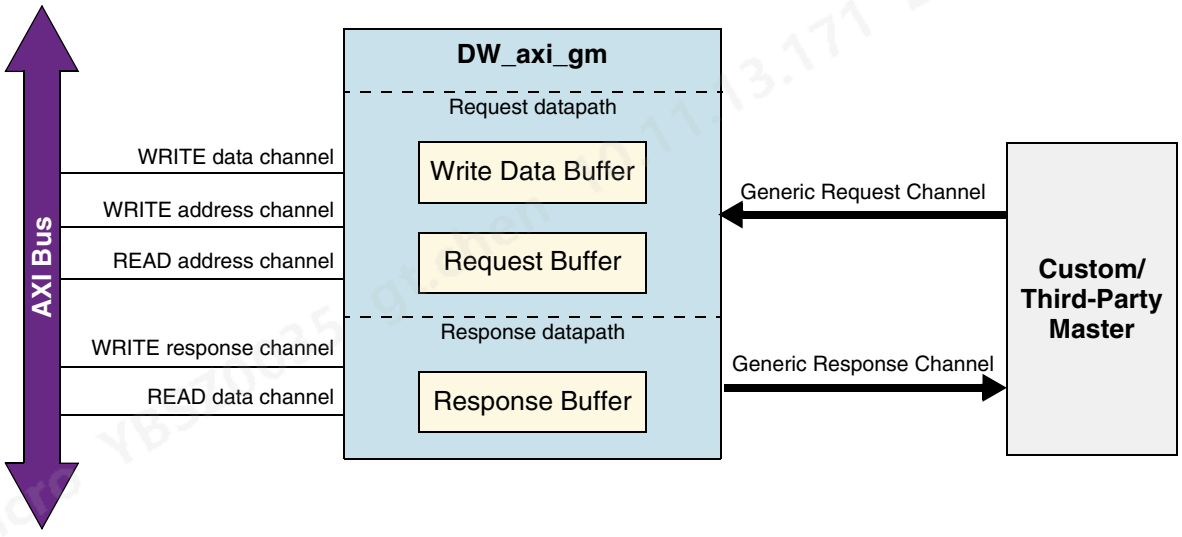
- A combined read/write Generic Request Channel that receives new transactions from the custom or third-party controller
- A combined read/write Generic Response Channel that provides read data and write responses

The DW_axi_gm is able to pipeline multiple GIF transactions. Internal to the DW_axi_gm are user-configurable buffers, which allow channel transaction queuing if the destination channel ever stalls transactions.

1.2.1 DW_axi_gm Block Diagram

Figure 1-2 shows a block diagram of a DW_axi_gm peripheral.

Figure 1-2 DW_axi_gm Block Diagram



As illustrated in Figure 1-2, the DW_axi_gm is partitioned into three buffers that communicate with the AXI bus through the five AXI channels and the GIF through the two generic channels.

- Write Data buffer – Buffers write data from the GIF.
- Request buffer – Shared for read and write requests from the GIF.
- Response buffer – Shared for read responses, read data, and write responses from the AXI bus.

1.3 Features

The DW_axi_gm supports the following features:

- Complies with the following specifications:
  - AMBA 3 AXI
  - AMBA 4 AXI
- Full support of AXI low-power interface
  - Transaction activity status to an external low-power controller (LPC) for enabling or disabling the clock.
  - Configurable maximum number of clock cycles the system must remain idle before entering low-power mode.
- Two-way flow control
- Synchronous point-to-point communication on generic interface (GIF)
- Independent request and response GIF channels
- Unlimited outstanding transactions (when GM_LOWPWR_HS_IF = 0)
- Sideband/user signals on all channels
- Synchronous clock support; including slower GIF clock
- Configurable micro-architecture
- Transaction block control

For details on these features, see "Functional Description" on page 17.

The DW_axi_gm has the following known limitations:

- No support for AXI exclusive accesses
- No support for write data interleaving
- No control over completion of transactions in locked sequences; GIF master is required to follow AXI master protocol rules for locked sequences.
- When the AXI4 interface is selected, region signaling is not supported.
- Sideband/user signal width verified up to 64 bits only

1.4 Standards Compliance

The DW_axi_gm conforms to the AMBA 3 AXI and AMBA 4 AXI specifications defined in the *AMBA AXI and ACE Protocol Specification* from ARM.

1.5 Verification Environment Overview

The DW_axi_gm includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The "Verification" on page 73 section describes the DW_axi_gm testbench environment.

1.6 Where To Go From Here

At this point, you may want to get started working with the DW_axi_gm component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components — coreConsultant and coreAssembler. For information on the different coreTools, see *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_axi_gm component, see *"Overview of the coreConsultant Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 2 User Guide*

For more information about implementing your DW_axi_gm component within a DesignWare subsystem using coreAssembler, see *"Overview of the coreAssembler Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

This chapter describes the DW_axi_gm, including the Generic Interface (GIF) and overall functionality.

2.1 Overview

The Generic Interface Module for an AMBA AXI master — DW_axi_gm —simplifies the connection of custom or third-party master controllers to the AMBA AXI bus using a Generic Interface (GIF). The GIF consists of two independent channels for requests and responses. Each channel is a point-to-point connection from a single source to a single sink. The channels offer two-way flow control similar to the valid/ready handshake on the AXI bus.

As the DW_axi_gm accepts new transactions from the GIF, it transfers them to the AXI master read or write channel and can be configured to buffer GIF transactions if the AXI is unable to immediately accept a transaction. Similarly, read data and write responses from the AXI interface can be configured to be buffered if the GIF is unable to immediately accept a response.

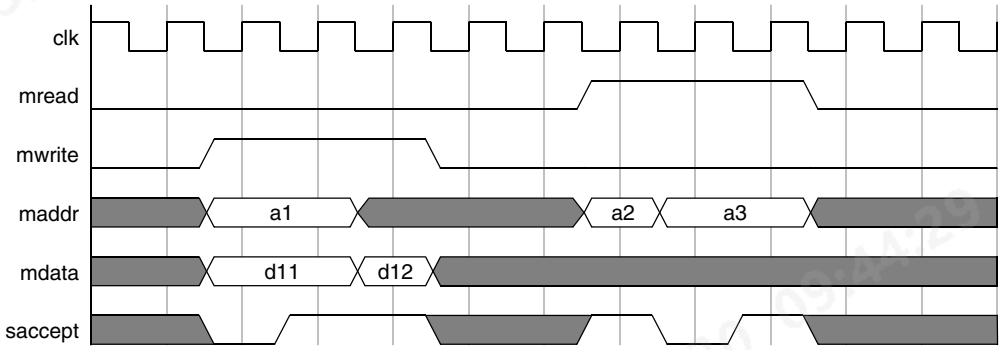
By default, DW_axi_gm makes no restriction on AXI transaction completion or data interleaving. If required for the connected custom or third-party master controller, you can configure the GIF to require all transactions (reads and writes) to execute in order.

All channels on the DW_axi_gm support sideband/user signal propagation from the AXI to the GIF and vice versa. When the AXI4 interface is enabled, the DW_axi_gm supports QoS (Quality of Service) signal propagation from the GIF to the AXI4 interface as well as user signal propagation from the GIF to the AXI4 interface and the AXI4 interface to the GIF.

2.2 GIF Request Channel

Figure 2-1 illustrates a timing diagram for a two-beat write and two consecutive reads to the GIF Request Channel.

Figure 2-1 Write and Read Request



A write request starts when the mwrite signal asserts high; similarly, a read request starts when the mread signal asserts high. The mwrite and mread signals should never be driven high at the same time. If this happens while waiting for a new mread/mwrite request, both signals are treated as if both are low, and no access starts. If mread is ever driven high during a mwrite burst sequence, it is ignored and the burst cycle is executed as if mread were low. Once a valid access starts, a high on the saccept signal indicates that the slave accepts the current request cycle. If a low occurs on the saccept signal, the master must extend the current request cycle.

Single transactions and read bursts have only one request cycle. Write bursts have multiple cycles, referred to as beats. Each beat must be accepted separately by the saccept signal, which must be sampled high *N* times for a burst that has *N* beats. The address in a write burst must be asserted only in the first beat; addresses in other beats are optional and are ignored by the DW_axi_gm slave in order to simplify the address generation logic. All other write burst control signals must remain stable through the entire burst sequence.

Once a valid write burst access starts, all write data for the write burst must be delivered to DW_axi_gm before the next GIF request transaction can start. GIF Transaction Examples, Figure 2-7 on page 25 shows an example of how write data can be "stalled" if the generic master cannot provide all write data beats continuously. The GIF request interface requires that if the mwrite signal is turned off before the completion of all data beats — stalling the write transaction — the remaining write data beats must complete before the next read (or new write) transaction. If the mread signal is asserted during a stalled write burst, the read transaction is ignored and the GIF request cycle is considered idle. If the mwrite signal is asserted for a new write transaction during a stalled write burst, data corruption occurs, since the new write transaction data beats are considered the next data beats of the unfinished write burst.

As seen in Figure 2-1, the saccept signal can be low or high on the first beat. If the slave cannot latch the address phase, then it must drive saccept low to ensure that the same address is still driven by the master. If the slave can latch at least one address phase, it can drive saccept high on the first cycle, and if necessary, drive saccept low on subsequent cycles.

Figure 2-1 does not show all the control signals. Additional control signals are mlock, msize, mlen, and mburst. Except for mlock, all other control signals are identical to their counterparts on the AXI read and write request channels. The DW_axi_gm does not support exclusive accesses, only locked accesses, and therefore only a one-bit signal is needed to indicate a locked access.

The DW_axi_gm also supports sideband/user signals, with a maximum width of 256 bits on each channel. The masideband signal on the GIF request channel passes information to the AXI write address or AXI read address sideband/user signal and the mwsideband signal passes information to the AXI write data sideband/user signal.

Attention

If you use both the DW_axi_gm and DW_axi_gs components, you should understand that the DW_axi_gs differs in the definition of the request channel:

- GIF reads are handled differently by the DW_axi_gs and DW_axi_gm. The DW_axi_gs initiates multi-beat GIF read bursts to the external GIF slave, generating addresses for each beat of the burst. In contrast, the DW_axi_gm supports only a single cycle GIF read burst request from the external GIF master; therefore only a single address is used. This simplifies address generation logic in peripherals connected to the DW_axi_gs and DW_axi_gm GIF.
- GIF writes are handled slightly differently by the DW_axi_gs and DW_axi_gm. The DW_axi_gs initiates multi-beat GIF write bursts to the external GIF slave, generating addresses for each beat of the burst. Similarly, the DW_axi_gm supports a multi-beat GIF write burst request from the external GIF master, but it needs only the address for the first beat of the burst; subsequent beat addresses are not necessary and are ignored. This simplifies address generation logic in peripherals connected to the DW_axi_gs and DW_axi_gm GIF.
- The mlock signal does not exist on the DW_axi_gs.
- The mlast signal does not exist on DW_axi_gm, which keeps track of the write beats internally.

2.2.1 Outstanding Transaction Limits

By default, there is no limit to the number of read and write transfers that are outstanding in the DW_axi_gm.

When the low-power interface is added — that is, GM_LOWPWR_HS_IF = 1 — the maximum number of outstanding read and write transactions is limited by the coreConsultant/coreAssembler parameters GM_MAX_PENDTRANS_READ and GM_MAX_PENDTRANS_WRITE. This is due to the need to track outstanding transactions in order to control the cactive output in the low-power interface.

- If a read request is received (mread) while the maximum number of read transactions are outstanding, saccept is de-asserted until a read transaction completes.
- If a write request is received (mwrite) while the maximum number of write transactions are outstanding, saccept is de-asserted until a write transaction completes.

If the read transaction limit has been reached while the write transaction limit has not been reached, write transactions are accepted. If the write transaction limit has been reached while the read transaction limit has not been reached, read requests are accepted.

Because of the shared request channel on the GIF — and in order to avoid a combinatorial path from mread/mwrite to saccept — saccept goes low by default if either the read or write transaction limit is reached. saccept signal then asserts one cycle after sampling a request for the transaction direction (read or write), which has not yet reached its transaction limit. This also occurs for subsequent new transfers while the read or write (not both) transaction limit has been reached; that is, saccept de-asserts between each transfer in order to wait until the direction of the next transfer can be sampled before deciding to assert saccept.

According to the GIF protocol, once a write burst is started, it must complete before any new write or read bursts are sent. For this reason, saccept does not depend on the number of outstanding read or write

transfers once a write burst has started but not yet completed. Once a write burst has been started, saccept remains high if there is available buffer space.

Figure 2-2 illustrates how the outstanding transaction limits work.

Figure 2-2 Outstanding Transaction Control

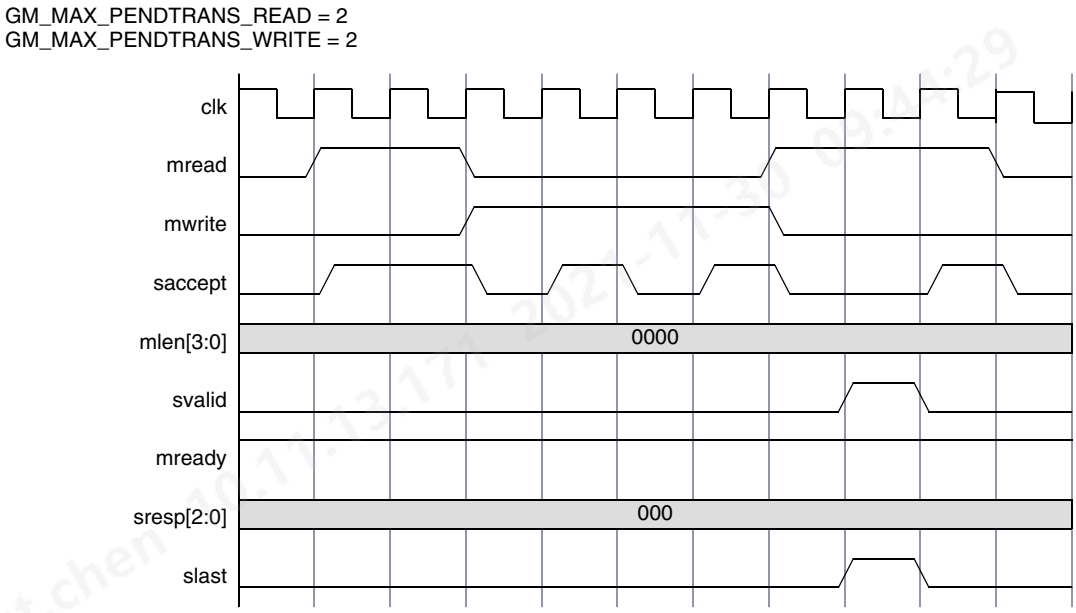


Figure 2-2 shows the following:

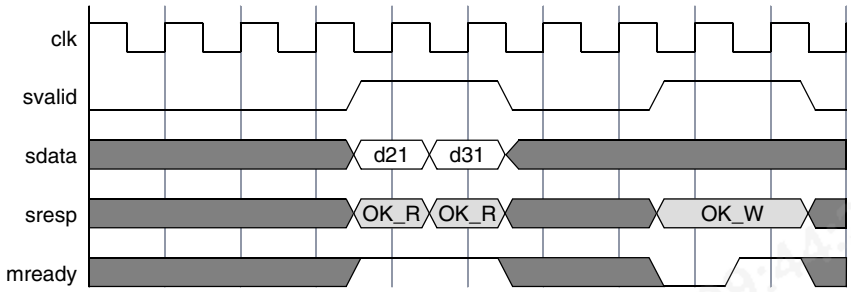
1. Two read requests, after which the outstanding read transaction limit has been reached.
2. A write transaction; note that saccept de-asserts immediately after the two reads.
3. saccept asserts again after one cycle; the DW_axi_gm has decoded that a write request is present, but the write transaction limit has not been reached.
4. After the write has been accepted, saccept again goes low for one cycle after waiting until the next cycle to see if a read or write request arrives.
5. The next cycle brings another write request; after another one-cycle delay (recalling the need to avoid a combinatorial path from mread/mwrite to saccept), saccept asserts again.
6. After these two writes, there is another read request, which must wait until one of the previous read transactions completes before saccept can be asserted.

Note It is recommended that GM_MAX_PENDTRANS_READ and GM_MAX_PENDTRANS_WRITE be set to values that correspond to or exceed the outstanding transaction capabilities of the attached GIF master. This avoids any loss of throughput due to DW_axi_gm needing to enforce the outstanding read and write transaction limits by pulling saccept low.

2.3 GIF Response Channel

Figure 2-3 illustrates a timing diagram for a read and write to the GIF response channel.

Figure 2-3 Read and Write Response



The svalid signal indicates a valid response, which can be comprised of data and the response information for a read, or of just the response information for a write. If the master is not ready to accept the response, it can drive mready low, at which point the slave extends the current cycle.

A read response contains the actual data to be read, along with the response information; both are asserted in the same cycle. In contrast, a write response contains only the response information. Additionally, there is always one response for every beat in a burst of a read transaction, whereas there is only one response for a complete burst in a write transaction.

Since read and write responses can come out-of-order with respect to the request sequence, the response signal has a separate encoding for read and write responses. This encoding is different from the encoding used on the AXI bus. If in-order read and write responses are required, relative to the original request order, see "Transaction Block Control" on page 29.

If an attached GIF master uses different transaction IDs, it is also possible for reads to complete out-of-order compared to other reads. The same is true for writes responses. DW_axi_gm does not enforce these ordering rules and directly passes results from the AXI channels to the GIF response channel.

The srsideband signal on the GIF response channel gets the information from the AXI read data or the AXI write response sideband/user signal.

Attention

If you use both the DW_axi_gm and DW_axi_gs components, you should know there is a difference in the encoding of the response signal. For the DW_axi_gm, the sresp output signal distinguishes between SLVERR and DECERR and is three bits wide, whereas the DW_axi_gs understands only SLVERR and has a 2-bit wide sresp input signal. If you do not need this distinction from the DW_axi_gm sresp output signal, bit 1 can be ignored on the master, which results in the same 2-bit encoding for master and slave. For the sresp output signal encoding details, see "sresp[2:0]" on page 55.

2.4 GIF Transaction Examples

Figure 2-4 shows the behavior of sideband/user signals for a single beat read transaction. Note the following:

1. On the GIF, the masideband signal is asserted only after the mread signal is asserted.
2. The masideband signal is transferred to the arsideband/aruser signal on the AXI interface and is qualified by the arvalid signal.
3. The rsideband/ruser signal on the AXI interface is transferred to the srsideband signal on the GIF and is qualified by the svalid signal.

Figure 2-4 Read Transaction of a Single Beat

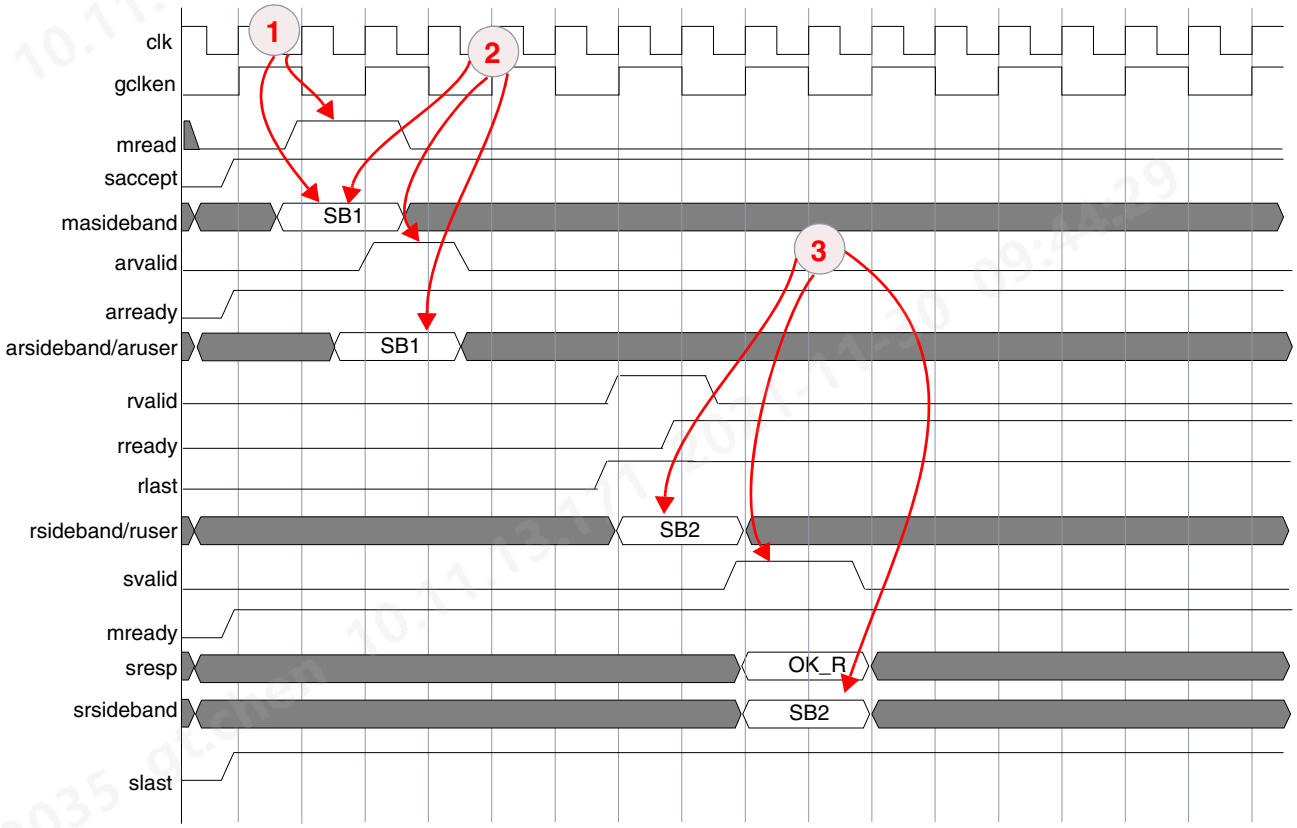


Figure 2-5 shows the behavior of sideband/user signals for a single beat write transaction. Note the following:

1. On the GIF, the masideband and mwsideband signals are asserted only after the mwrite signal.
2. The masideband signal is transferred to the awsideband/awuser signal on the AXI interface and is qualified by the awvalid signal.
3. The mwsideband signal is transferred to the wsideband/wuser signal on the AXI interface and is qualified by the wvalid signal.
4. The bsideband/buser signal on the AXI interface is transferred to the srsideband signal on the GIF and is qualified by the svalid signal.

Figure 2-5 Write Transaction of a Single Beat

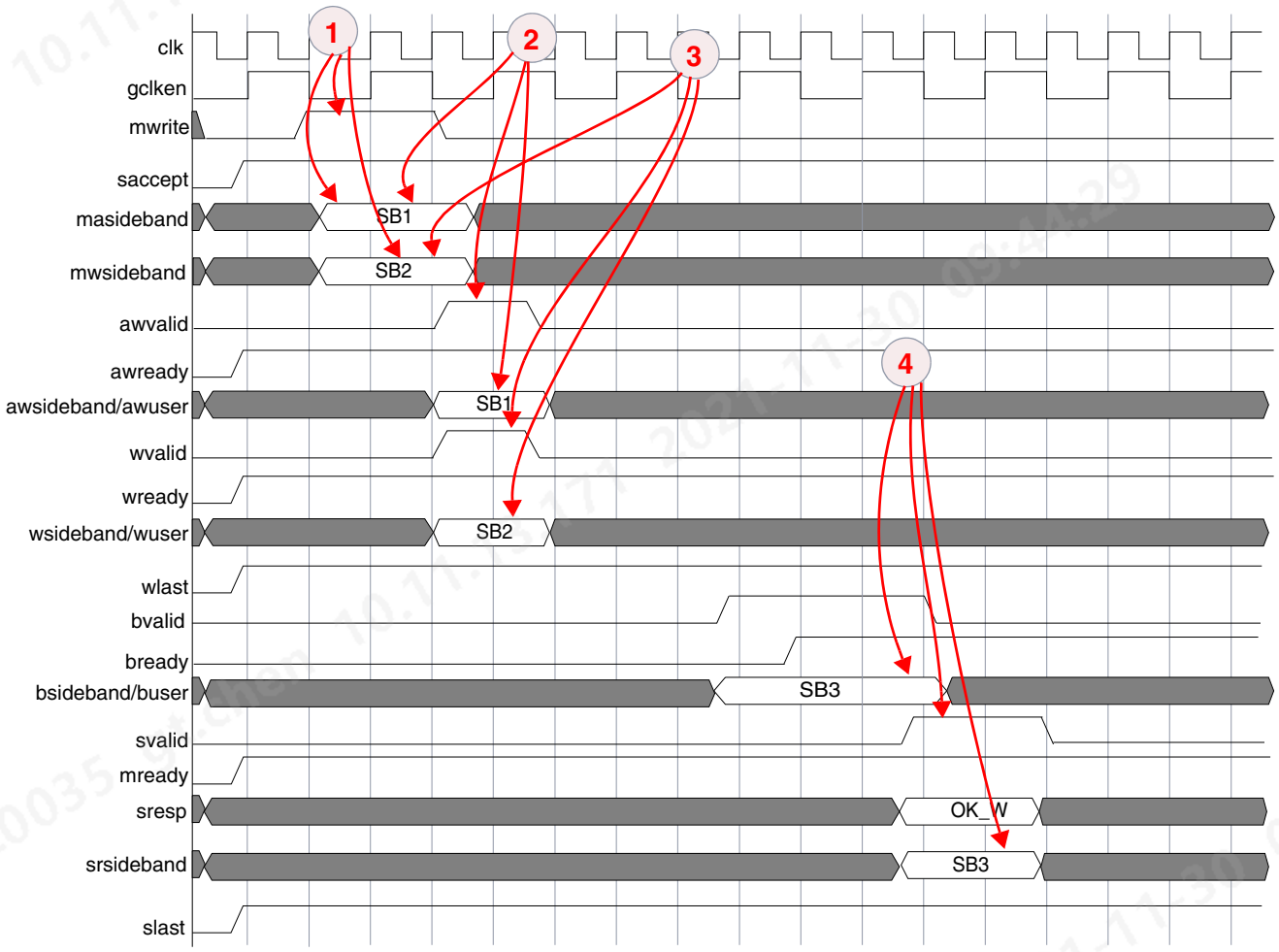
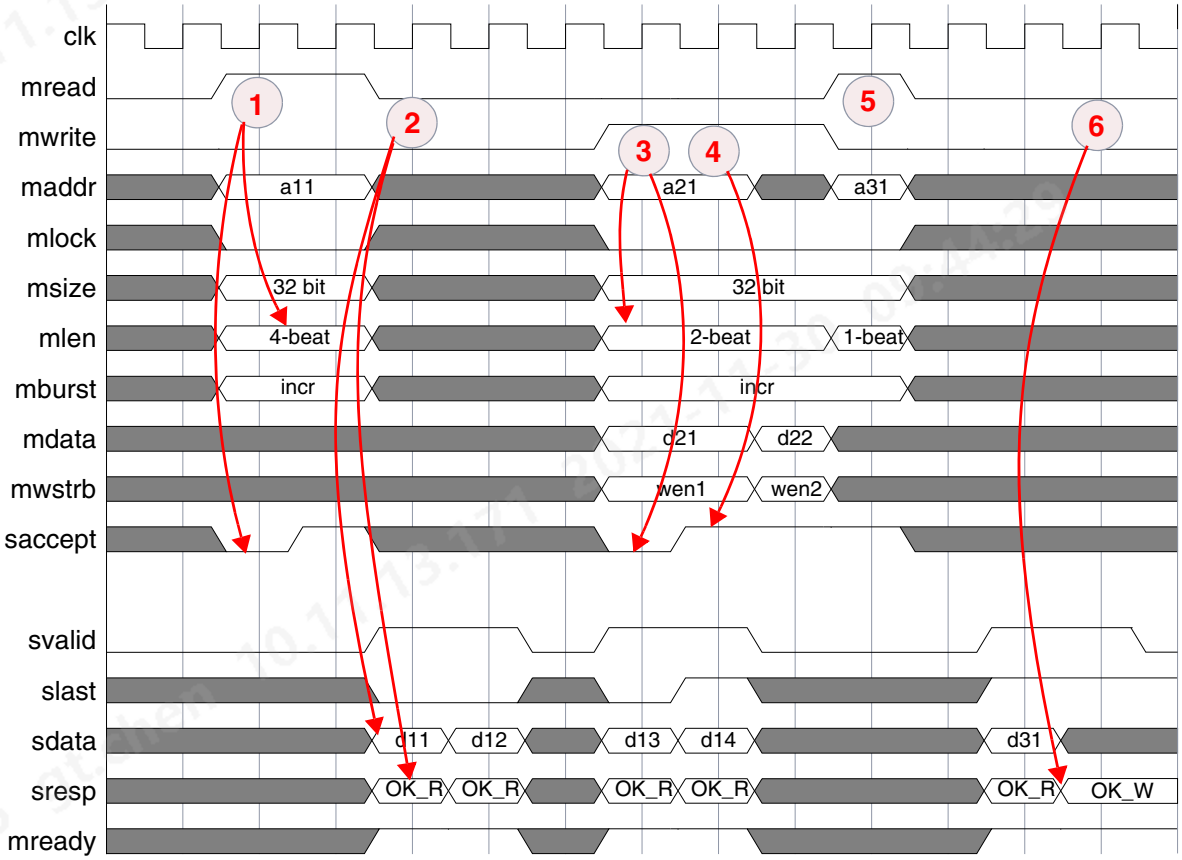


Figure 2-6 shows several read and write transactions for the generic interface. As discussed, in a read burst, saccept acknowledges acceptance of the entire burst when sampled high for a single cycle. For write burst requests, saccept acknowledges acceptance of only one beat at a time. Therefore, read bursts on the DW_axi_gm have single-cycle requests and multiple-cycle responses. In contrast, write bursts have multiple-cycle requests and single-cycle responses.

Figure 2-6 illustrates a GIF transaction sequence consisting of a 4-beat read, 2-beat write, and a 1-beat read.

Figure 2-6 DW_axi_gm Read and Write Bursts



Note the following behaviors in Figure 2-6:

1. Start a 4-beat read burst; DW_axi_gm not ready and drives saccept low for one cycle
2. The first read data/response comes back
3. Start a 2-beat write burst; DW_axi_gm not ready and drives saccept low for one cycle
4. The first write request accepted
5. Single read immediately follows write burst
6. Write response comes back after read data/response, which is okay by default; configurable transaction blocking can be used to enforce in-order transactions and responses

Figure 2-7 illustrates a one-beat write GIF request. Note the following:

1. mwrite asserted to DW_axi_gm; AXI write addr/data
2. awvalid and wvalid asserted from DW_axi_gm; AXI write response
3. bvalid and bready assert — bready only on the last AXI clock cycle before GIF rising edge
4. svalid asserted from DW_axi_gm

Figure 2-7 1-Beat Write GIF Request

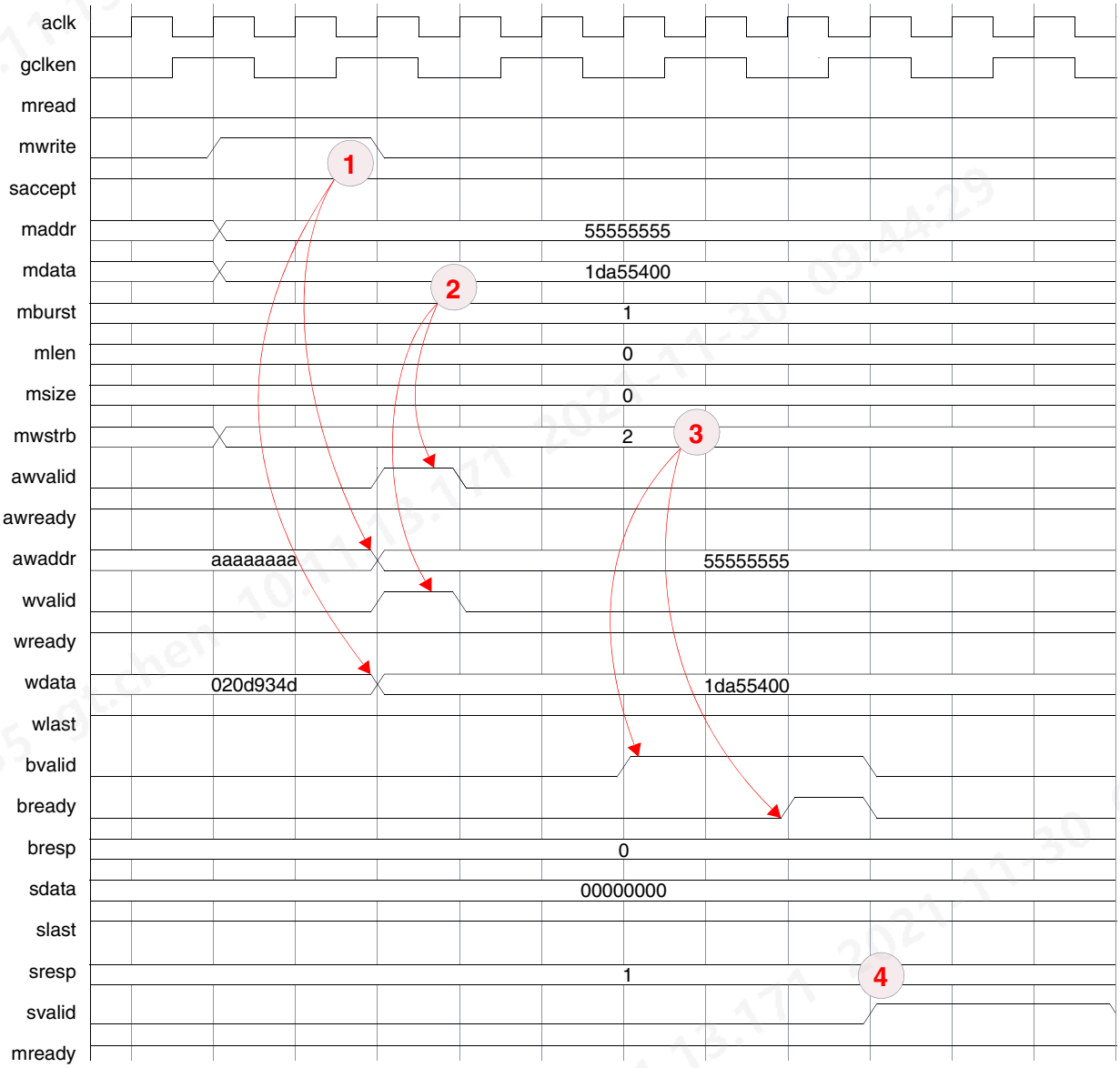


Figure 2-8 illustrates a one-beat read GIF request. Note the following:

1. mread asserted to DW_axi_gm; AXI read addr/data
2. arvalid and rvalid asserted from GM; AXI read response
3. svalid asserted from DW_axi_gm

Figure 2-8 1-Beat Read GIF Request

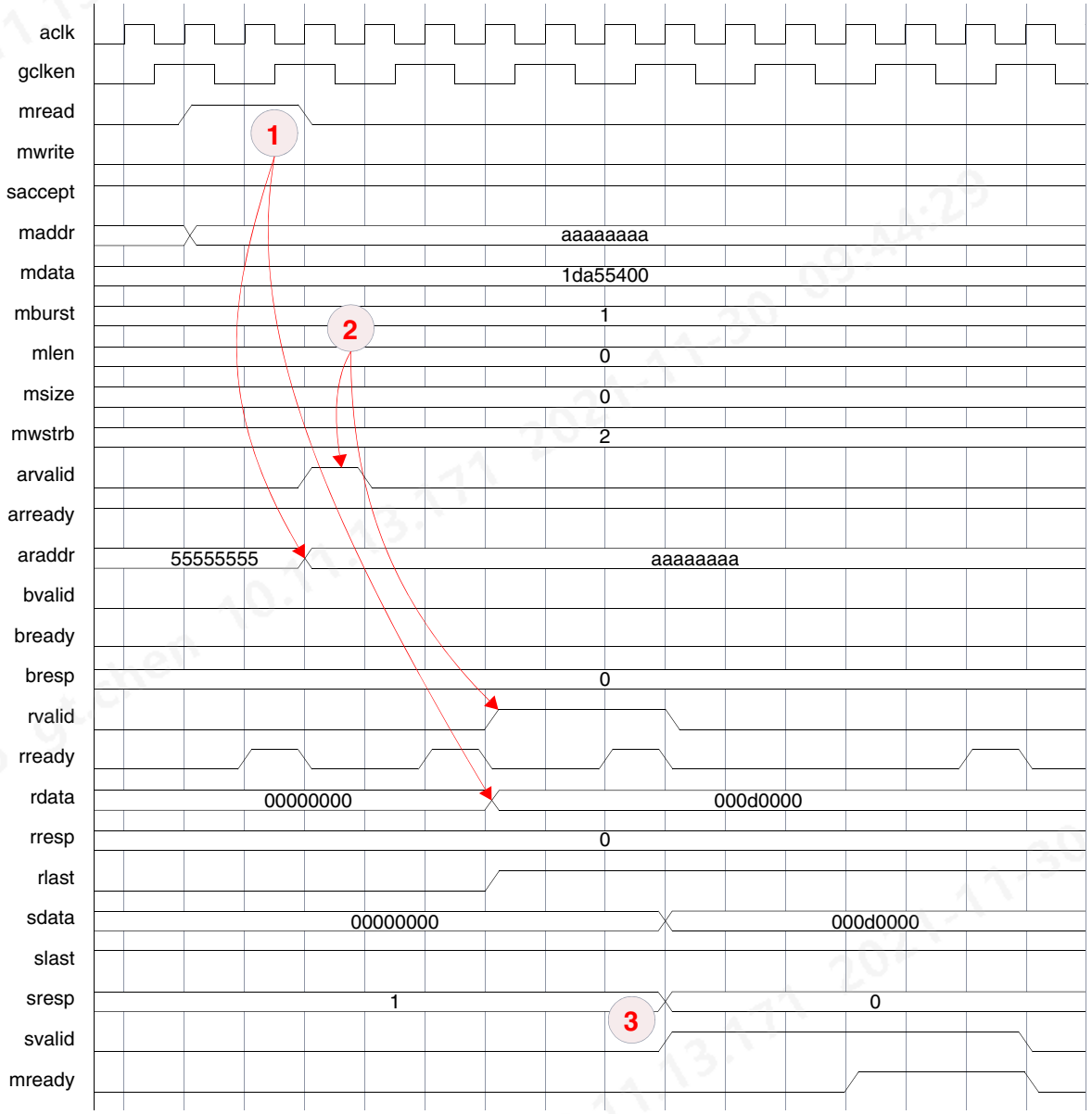
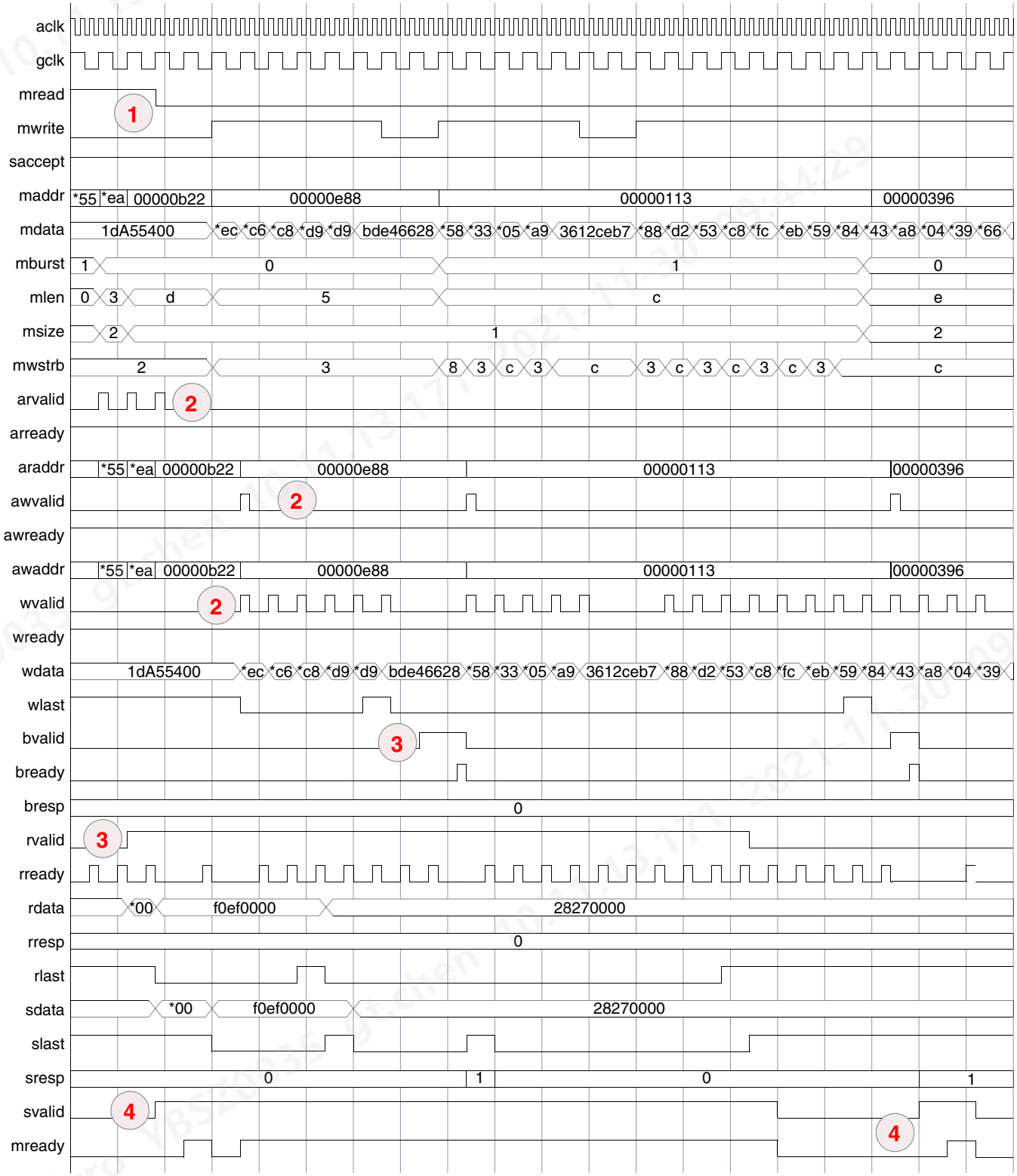


Figure 2-9 illustrates a multiple-beat read and multiple-beat write GIF request that consists of: three reads (1-beat, 4-beat, and 14-beat) and two writes (6-beat and 13-beat). You can see in Figure 2-9 that reads and writes can be completed out-of-order. In this example, the first write is initiated on the GIF request channel after the last read, but the write response is executed on the GIF response channel before the last read data is completed. The numbers on the diagram refer to these general events:

1. mread/mwrite assertions on the GIF request channel
2. arvalid/awvalid/wvalid assertions on AXI request channels
3. bvalid/rvalid assertions on AXI response channels
4. svalid assertions on the GIF response channel; 6-beat write is returned before the 14-beat read (out of order is allowed), then final 13-beat write is the last svalid/mready pair in the lower right corner

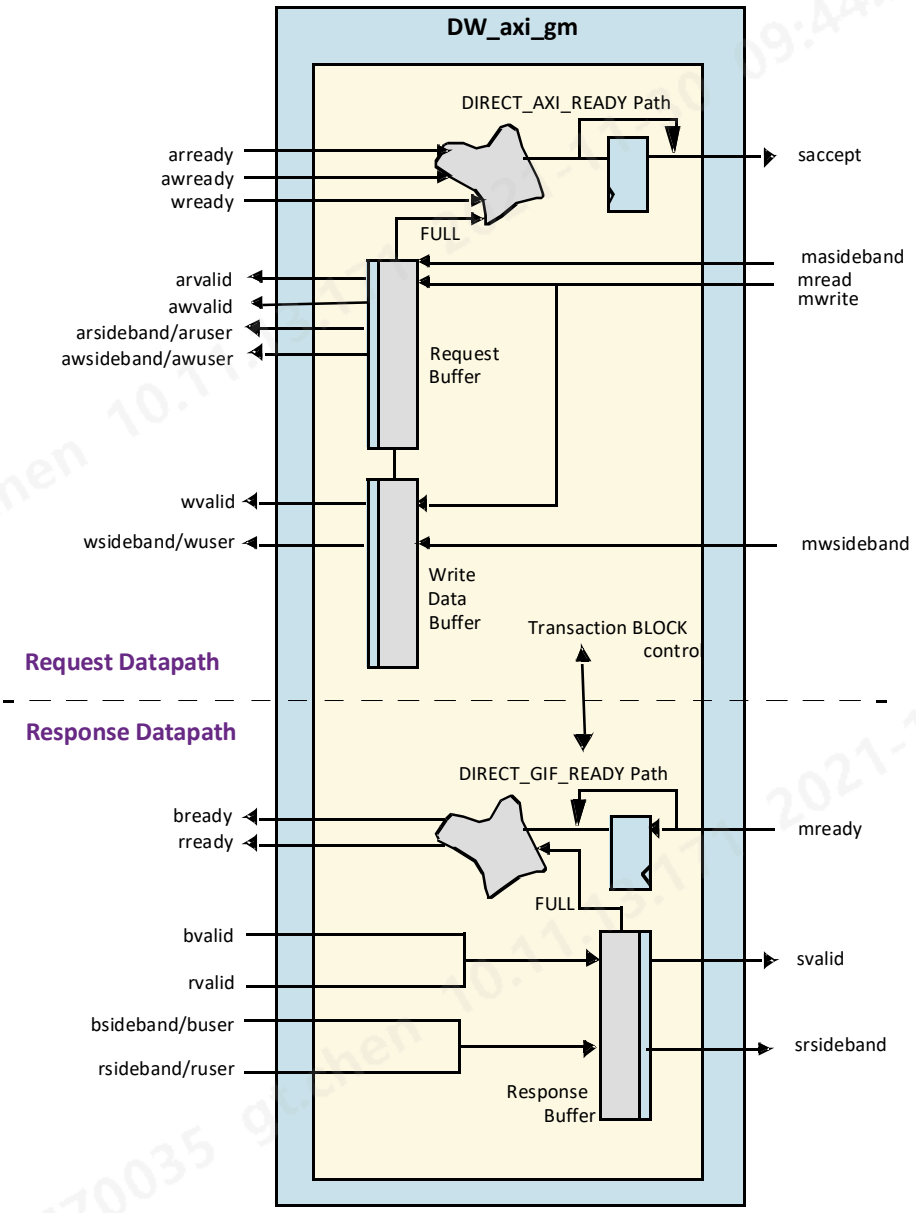
Figure 2-9 Multiple-Beat Read and Write Series



2.5 Microarchitecture

Figure 2-10 illustrates the microarchitecture of the DW_axi_gm. The diagram shows only handshake signals in order to simplify the relationships. DIRECT_AXI_READY, DIRECT_GIF_READY, Transaction BLOCK control, and Buffer depths are all configurable. See "Parameter Descriptions" on page 39 for configuration parameter details, and the remainder of this chapter for functionality details.

Figure 2-10 DW_axi_gm Microarchitecture



The Request buffer stores address and control signals for read and write requests that come in from the generic interface. The Write Data buffer stores write data and associated write strobes. The Response buffer stores read data and all responses from read and write transactions.

The mread- and mwrite-to-saccept handshake does not have a direct path from the mread/mwrite signals to the saccept signal in order to facilitate timing closure. The same holds true for the valid/ready handshake on each AXI write response and read data channels.

2.6 Direct-Ready Feedthroughs

In order to facilitate timing closure, you can configure a direct-ready feed-through in order to have either registered or unregistered ready signals from the GIF side to AXI and vice versa. If GM_DIRECT_AXI_READY is set to true, the AXI ready signals arready, awready and wready are combinationally connected to GIF saccept. If GM_DIRECT_GIF_READY is set to true, the GIF mready signal is combinationally connected to bready and rready.

Note There are performance degradation issues when ready signals are registered and the buffer depth is set to one. In this case data cannot stream through the one-deep buffers, and every access requires two cycles. This holds true for every single beat of a burst. To avoid this penalty the buffer depth has to be set to a minimum of two. To maintain streaming of one access every cycle, the receiving interface must maintain the same access rate, as well. Request and response data path do not influence each other and can be configured independently.

2.7 Transaction Block Control

This is another configurable feature of the DW_axi_gm. The user has the choice to block any new GIF transactions when either a read and/or write is outstanding. A GIF transaction is considered to be blocked, when a new GIF transaction cannot commence until the preceding AXI response of the transaction has completed; this ensures that the proceeding GIF response of the transaction occurs in order to the GIF master. This is in contrast to posted transactions, where GIF transactions can commence back-to-back without the need to wait for a response to come back first.

The default configuration for the DW_axi_gm does not block transactions and does not pose any limitation on the number of outstanding transactions.

Note The number of outstanding transactions is limited if the low-power interface is selected; that is, if GM_LOWPWR_HS_IF = 1.

Transaction blocking can be enabled separately for read and write transactions by setting the GM_BLOCK_READ or GM_BLOCK_WRITE configuration to true.

GM_BLOCK_READ blocks any new GIF transactions (read or write) when a GIF read is outstanding. GM_BLOCK_WRITE blocks any new GIF transactions (read or write) when a GIF write is outstanding. Whether transaction blocking is necessary or not depends on the capabilities of the custom or third-party master controller that is connected to the DW_axi_gm GIF. If both GM_BLOCK_READ and GM_BLOCK_WRITE are true, the depth of the GIF request buffer (GM_REQ_BUFFER) is limited to 1.

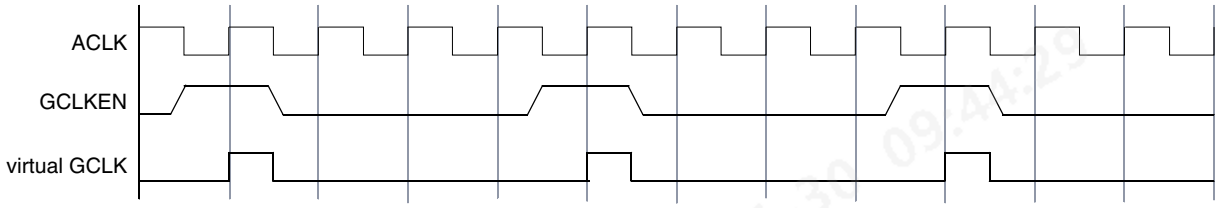
2.8 Synchronous Generic Clock

The AXI and GIF can operate using the same clock, or the GIF can run at a slower, synchronous frequency. To run at a slower frequency, the AXI clock frequency must be an integer multiple of the GIF clock

frequency. To run on a slower clock, you must generate an enable signal that enables the AXI clock only on every *N*th clock; this signal must be connected to the gclken port of the DW_axi_gm.

Figure 2-11 illustrates a generic clock that is four times slower that the AXI clock.

Figure 2-11 GCLK = ACLK / 4



2.9 Low-Power Interface

You can configure the DW_axi_gm to include an AXI low-power handshaking interface. This interface allows the following:

- DW_axi_gm informs the system low-power controller (LPC) when it has no outstanding transactions
- LPC requests the DW_axi_gm to enter into a low-power state

You can include this low-power interface in your design by setting the GM_LOWPWR_HS_IF parameter to 1.

The low power handshaking interface includes the following signals:

- csysreq input – de-asserted by the LPC to initiate a low-power state; asserted by LPC to initiate an exit from a low-power state
- csysack output – de-asserted by DW_axi_gm to acknowledge a request to enter a low-power state; asserted by DW_axi_gm to acknowledge a request to exit a low-power state
- cactive output – de-asserted by DW_axi_gm when the clock can be removed after the next positive edge; csysack must also be de-asserted

Note When GM_LOWPWR_HS_IF = 1, there are limits to the maximum number of outstanding read/write transactions set by GM_MAX_PENDTRANS_READ and GM_MAX_PENDTRANS_WRITE, respectively. This limit is applied at the GM master port by stalling the channel prior to the channel FIFO. Therefore, you should set a limit that is greater than the depth of the relevant address channel FIFO.

The sequence of events for entering a low-power state is as follows:

1. The LPC requests the DW_axi_gm to enter a low-power state by de-asserting the csysreq signal.
2. Since the DW_axi_gm does not have a power-up or power-down sequence, it always acknowledges a csysreq signal on the next cycle by asserting or de-asserting the csysack signal.

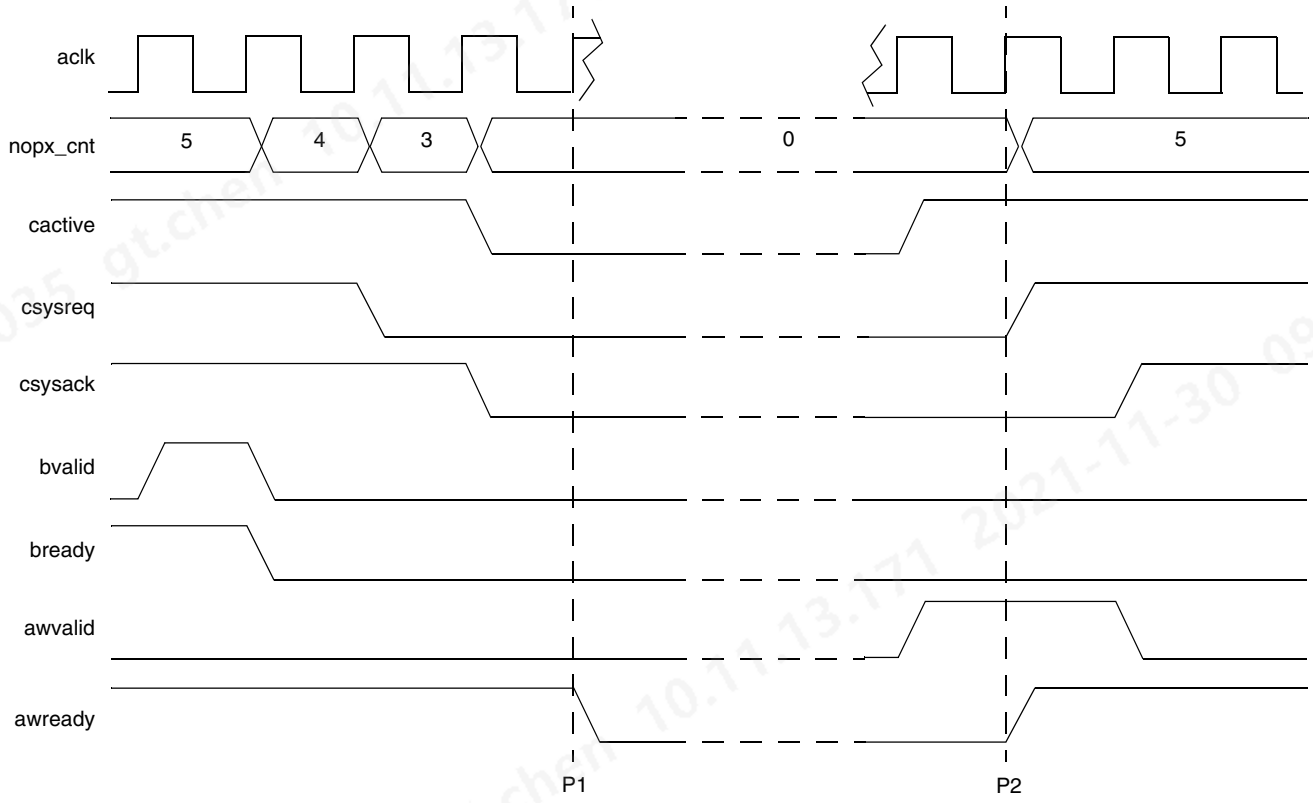
   When requested by the LPC, the low-power state depends on the value of the cactive signal at the time that the csysack signal is sampled by the LPC after the csysreq signal has been de-asserted.

3. The cactive signal de-asserts when the DW_axi_gm has no outstanding transactions. If you set the GM_LOWPWR_NOPX_CNT parameter to "X" in coreConsultant, the cactive signal de-asserts after *X* cycles, during which there are no outstanding transactions, have elapsed.
4. DW_axi_gm enters a low-power state when *all* of the cactive, csysreq, and csysack signals are low (0) when sampled at the positive edge of aclk; this is illustrated at P1 in Figure 2-12.

Note Entry to a low-power state happens at any cycle in which the previously mentioned conditions are satisfied.

While in a low-power state, the awready, wready, and arready signals are held low, which prevents any new transaction from starting and thus allows the clock to be safely disabled; this is illustrated between points P1 and P2 of Figure 2-12.

Figure 2-12 Low-Power State Entry and Exit (GM_LOWPWR_NOPX_CNT = 5)



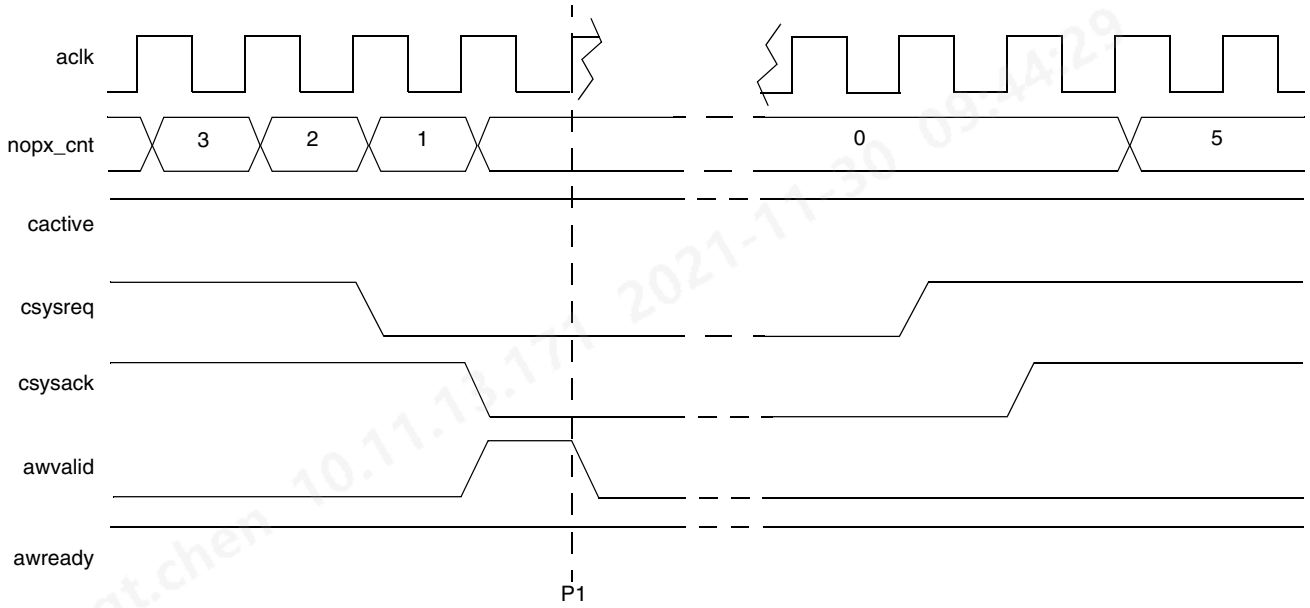
The DW_axi_gm exits a low-power state when the cactive signal goes high and is sampled at the positive edge of aclk; illustrated at P2 of Figure 2-12.

An exit from low-power can be initiated by:

- LPC asserting the csysreq signal, which causes the DW_axi_gm to assert the cactive signal, illustrated in Figure 2-14.
- A master asserting the awvalid or arvalid signals of the DW_axi_gm, which causes the DW_axi_gm to assert the cactive signal, illustrated in Figure 2-12.

Figure 2-13 shows the DW_axi_gm rejecting a request to enter a low-power state. At P1, the no-pending-transaction (nopx_cnt) counter has reached 0, but on this same cycle the awvalid signal asserts, which keeps the cactive signal asserted. The LPC samples at P1 that the DW_axi_gm has rejected the entry to low-power mode, and therefore must not remove the clock.

Figure 2-13 DW_axi_gm Low-Power Interface Rejects Low-Power Entry



2.9.1 cactive Signal De-assertion

The cactive signal de-asserts the GM_LOWPWR_NOPX_CNT parameter *aclk* cycles after the last pending transaction completes. If the csysreq signal de-asserts while the component is counting down to 0 from *GM_LOWPWR_NOPX_CNT*, the cactive signal will de-assert on the next cycle.

After the positive edge of the aclk signal where the cactive signal and the csysack signal are sampled low, the low-power controller (LPC) may remove the clocks from the DW_axi_gm (P1 in Figure 2-12). At this point the bready and awready signals also is driven low.

2.9.2 cactive Signal Assertion

The cactive signal asserts combinatorially when the awvalid or arvalid signals are asserted, at which point it remains asserted until all outstanding transactions have completed and *GM_LOWPWR_NOPX_CNT* cycles have elapsed.

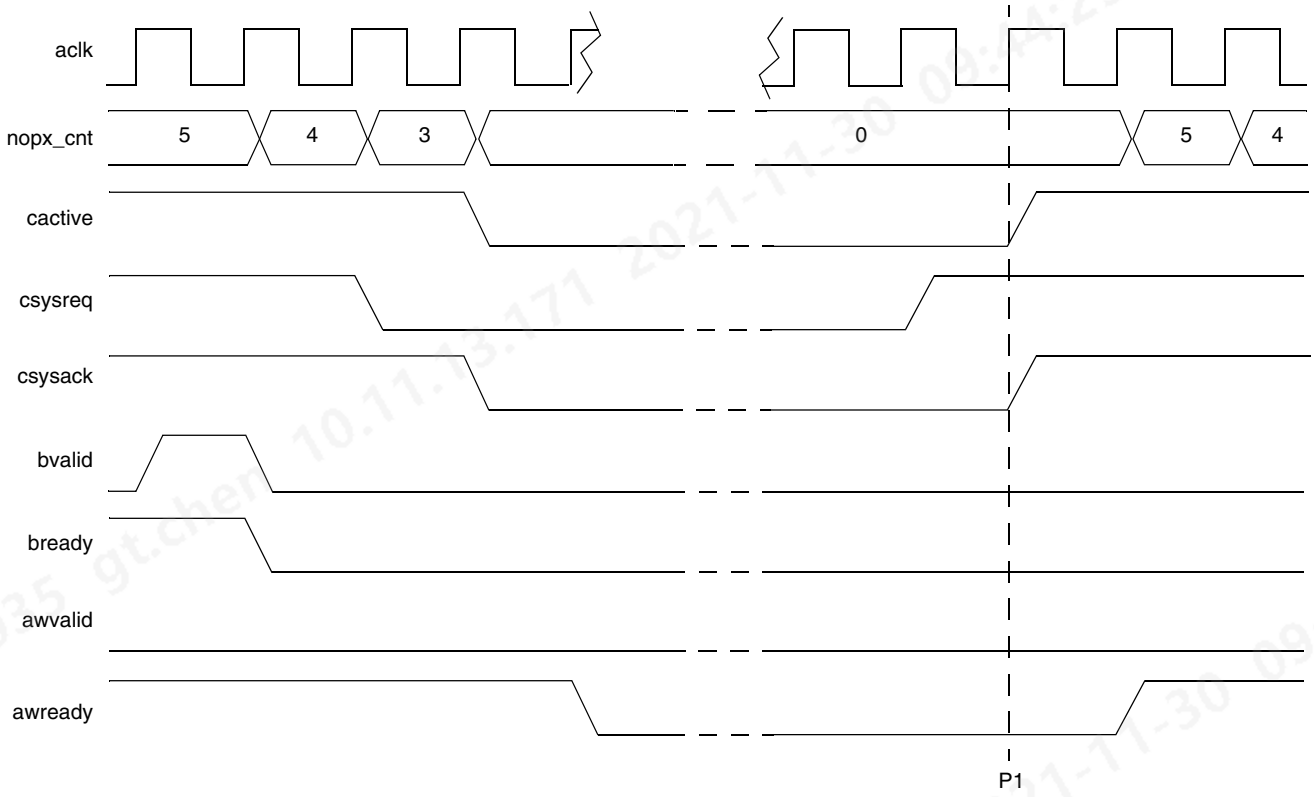
Note Early write data does not cause the cactive signal to assert or to remain asserted.

If the cactive signal is low and the csysreq signal is asserted, the DW_axi_gm asserts the cactive signal at the same time as the csysack signal in order to complete the low-power exit handshake. After the clock edge where the cactive and csysack signals are sampled high on exit from low-power mode, the DW_axi_gm keeps the cactive signal asserted for *GM_LOWPWR_NOPX_CNT* cycles, after which it will either:

- De-assert until there are active transactions again.
- De-assert until another low-power exit handshake is completed.

This can be seen at P1 in Figure 2-14 where the cactive signal is asserted 1 clock cycle after the csysreq signal asserts, and the no-pending-transaction counter (nopx_cnt) starts to decrement again from the next cycle.

Figure 2-14 LPC Initiates Low-Power Exit



Note

When coming out of reset, if the awvalid and arvalid signals equal 0:

- cactive signal equals 0, if GM_LOWPWR_NOPX_CNT parameter equals 0
- cactive signal equals 1, if GM_LOWPWR_NOPX_CNT parameter is greater than 0 and will de-assert when *GM_LOWPWR_NOPX_CNT* clock cycles have elapsed

2.10 AMBA 4 AXI Optional Signaling Interface

When the AXI4 interface is enabled, the following optional signals are available on the DW_axi_gm:

- QoS signals (awqos and arqos), for priority signaling. However, these signals are not processed by the DW_axi_gm; the DW_axi_gm just transfers the information in these signals from the GIF to the AXI4 interface.
- User signals (awuser, aruser, wuser, ruser, and buser) for user data signaling.

Note You must have a DWC-AMBA-Fabric-Source license to enable the AXI4 interface.

Figure 2-15 shows the optional signals for an AXI4 write transaction. Note the following:

1. The masideband signal on the GIF request channel is qualified by the mwrite signal and transferred to the awuser signal on the AXI4 write address channel, which is qualified by the awvalid signal.
2. The mwsideband signal on the GIF request channel is qualified by the mwrite signal and transferred to the wuser signal on the AXI4 write data channel, which is qualified by the wvalid signal.
3. The buser signal on the AXI4 write response channel is qualified by the bvalid signal and transferred to the srsideband signal on the GIF response channel, which is qualified by the svalid signal.
4. The mqos signal on the GIF address channel is qualified by the mwrite signal and transferred to the awqos signal on the AXI4 write address channel, which is qualified by the awvalid signal.

Figure 2-15 Optional Signals in an AXI4 Write Transaction

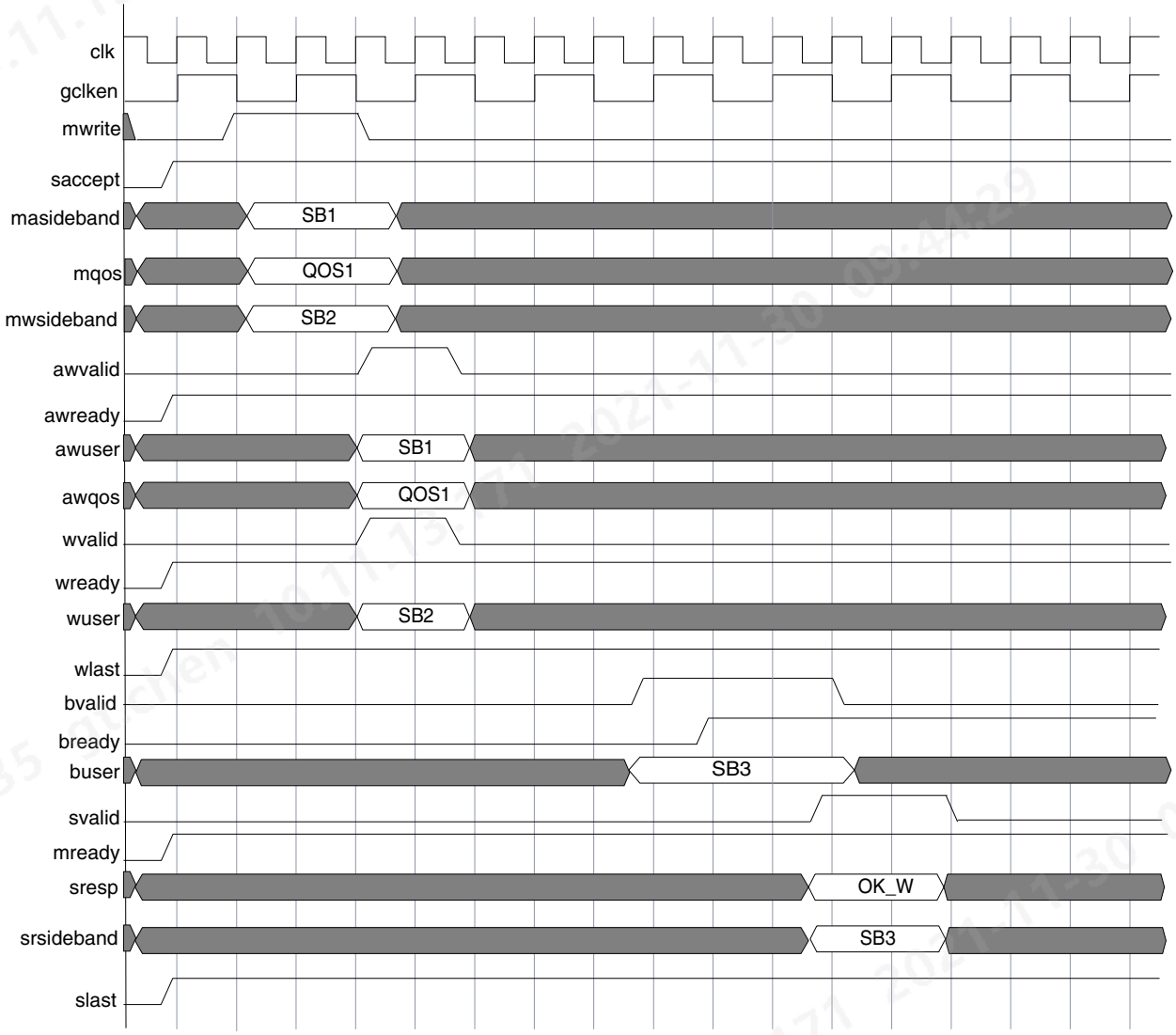
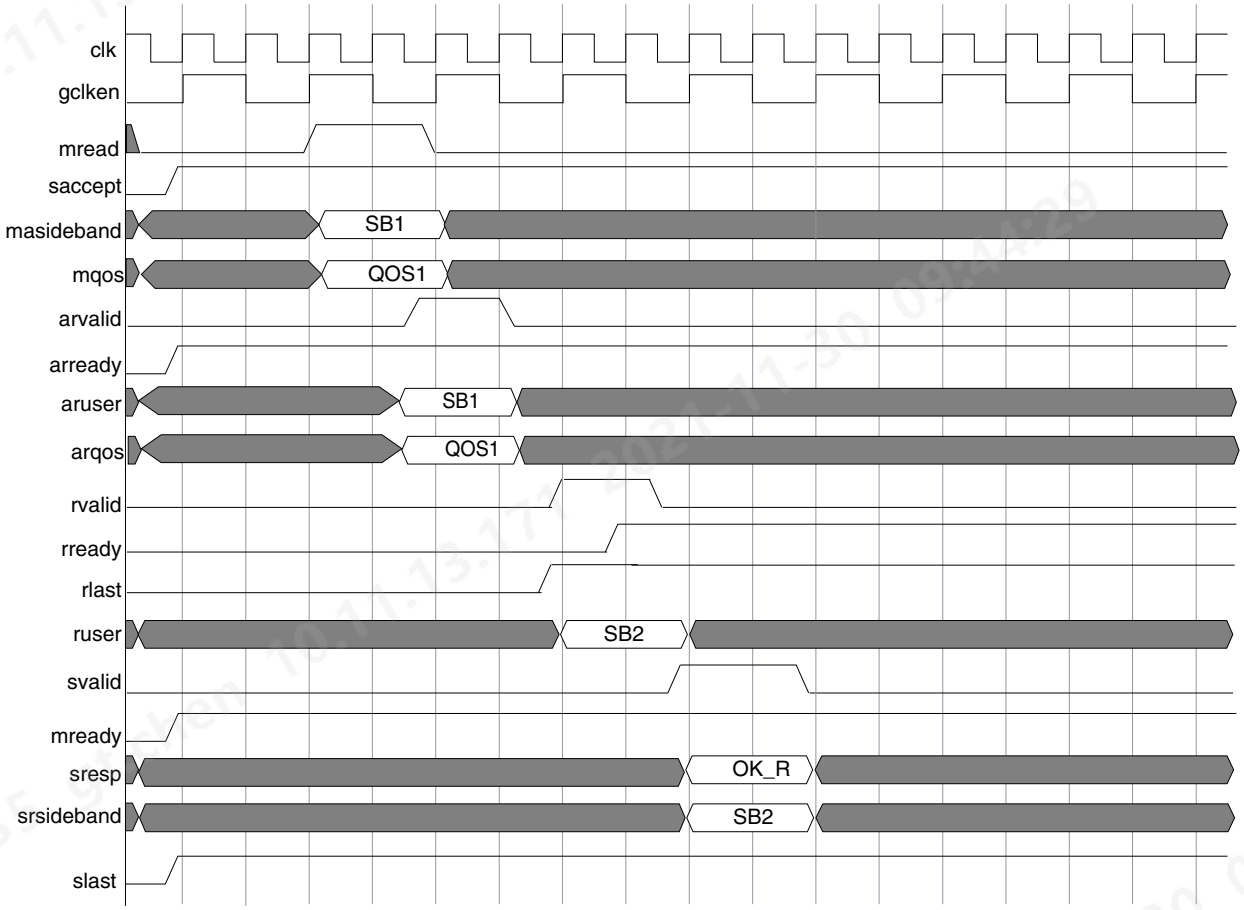


Figure 2-16 shows the optional signals for an AXI4 read transaction. Note the following:

1. The masideband signal on the GIF request channel is qualified by the mread signal and transferred to the aruser signal on the AXI4 read address channel, which is qualified by the arvalid signal.
2. The ruser signal on the AXI4 read data channel is qualified by the rvalid signal and transferred to the srsideband signal on the GIF response channel, which is qualified by the svalid signal.
3. The mqos signal on the GIF address channel is qualified by the mread signal and transferred to the arqos signal on the AXI4 read address channel, which is qualified by the arvalid signal.

Figure 2-16 Optional Signals in an AXI4 Read Transaction



2.11 Performance

The DW_axi_gm matches the performance of the AXI bus with the smallest possible footprint when used in the default minimum configuration. Performance can be measured with respect to throughput, latency, and clock speed.

Using the default configuration for the Direct-Ready parameters, latency occurs over one clock cycle such that an access started on the GIF side is asserted on the AXI side with the delay of one clock cycle. However, a clock cycle is measured on the GIF. If the GIF clock is slowed down by a factor of *N*, compared to the AXI clock, the access on the AXI interface is delayed by *N* AXI clock cycles.

Even with a slower GIF clock, throughput has 100% efficiency; that is, data streams through the DW_axi_gm without wait cycles when the receiving side is ready to accept the data. This is true for request and response data path. Only one configuration reduces throughput to 50%; for more information, see "Direct-Ready Feedthroughs" on page 29.

2.12 Locked Sequence Support

The DW_axi_gm provides support for locked transaction sequences. However, it does not implement any control or monitoring over the status of the individual transactions in the locked sequence. Therefore, the GIF master must ensure the following:

- All previous, unlocked transactions to the target slave have completed before issuing the first, locking transaction in the locked sequence
- All transactions in the locked sequence have completed before requesting the final unlocking transaction in the locked sequence
- Unlocking transaction has completed before issuing new transactions.

This feature is available only when the AXI 3 interface is selected (GM_AXI_INTERFACE_TYPE = 0).

3 Parameter Descriptions

This chapter details all the configuration parameters. **You can use the coreConsultant GUI configuration reports to determine the actual configured state of the controller.** Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

The parameter descriptions in this chapter include the **Enabled:** attribute which indicates the values required to be set on other parameters before you can change the value of this parameter.

These tables define all of the configuration options for this component.

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
Top Level Parameters	
Select Interface AXI3/AXI4.	Select AXI Interface Type as AXI3 or AXI4 **Values:** ■ AXI3 (0) ■ AXI4 (1) **Default Value:** AXI3 **Enabled:** DWC-AMBA-Fabric-Source license required **Parameter Name:** GM_AXI_INTERFACE_TYPE
GIF & AXI Address Width	Address width on AXI and GIF interfaces. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** GM_AW
GIF & AXI Data Width	Data width on AXI and GIF interfaces. No distinction is made between read and write channels. **Values:** 8, 16, 32, 64, 128, 256, 512 **Default Value:** 32 **Enabled:** Always **Parameter Name:** GM_DW
GIF & AXI ID Width	Width of transaction ID field of the AXI system (awid, arid, wid, rid, bid) and GIF master (mid, sid) interfaces. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 **Default Value:** 4 **Enabled:** Always **Parameter Name:** GM_ID
GIF & AXI Burst Length Width	Width of the burst length field of the AXI and GIF interfaces. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** Always **Parameter Name:** GM_BW

Table 3-1 Top Level Parameters (Continued)

Label	Description
Include QoS Signals?	If set to True, the QoS is enabled in DW_axi_gm. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** GM_AXI_INTERFACE_TYPE != 0 **Parameter Name:** GM_AXI_HAS_QOS

3.2 Performance Parameters

Table 3-2 Performance Parameters

Label	Description
Request Path Buffer Configuration	
Combinational GIF Ready	If true, the mready input is combinationally connected to the rready/bready outputs. If false, the mready input is registered, inserting one cycle of latency. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** GM_DIRECT_GIF_READY
Request Buffer Depth	Depth of GIF request buffer. Higher values allow GIF requests to be buffered, rather than stalled, if the AXI address channel stalls DW_axi_gm transactions. Higher values also increase gate count. If both GM_BLOCK_READ and GM_BLOCK_WRITE are true, GM_REQ_BUFFER must be set to 1. If GM_DIRECT_GIF_READY is false, then it is recommended to set to 2 or higher in order to avoid performance degradation. **Values:** 1, 2, 3, 4 **Default Value:** 1 **Enabled:** Always **Parameter Name:** GM_REQ_BUFFER
Write Data Buffer Depth	Depth of GIF write data buffer. Higher values allow GIF write data to be buffered, rather than stalled, if the AXI write data channel stalls DW_axi_gm transactions. Higher values also increase gate count. If GM_DIRECT_GIF_READY is false, it is recommended to set GM_WDATA_BUFFER to 2 or higher in order to avoid write performance degradation. **Values:** 1, 2, 3, 4 **Default Value:** 1 **Enabled:** Always **Parameter Name:** GM_WDATA_BUFFER

Table 3-2 Performance Parameters (Continued)

Label	Description
Response Path Buffer Configuration	
Combinational AXI Ready	If true, AXI awready, wready, and rready inputs are combinationally connected to the GIF saccept output. If false, these inputs are registered, inserting one cycle of latency. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** GM_DIRECT_AXI_READY
Response Buffer Depth	Depth of combined AXI read and write response buffer. Higher values allow AXI responses to be buffered, rather than stalled, if the GIF response channel stalls DW_axi_gm transactions. Higher values also increase gate count. If GM_DIRECT_AXI_READY is false, recommended to be set to 2 or higher to avoid read and write response performance degradation. **Values:** 1, 2, 3, 4 **Default Value:** 1 **Enabled:** Always **Parameter Name:** GM_RESP_BUFFER
Protocol	
Block all Reads	If true, current GIF read request must complete (all read data received from AXI read data channel) before the next GIF request is accepted by DW_axi_gm. If false, GIF requests are allowed to be queued in the request buffer and transferred to the AXI read address channel before outstanding read requests complete. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** GM_BLOCK_READ
Block all Writes	If true, current GIF write request must complete (write reponse received from AXI write response channel) before the next GIF request is accepted by DW_axi_gm. If false, GIF requests are allowed to be queued in the request buffer and transferred to the AXI write address channel before outstanding write requests complete. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** GM_BLOCK_WRITE

3.3 Low Power Interface Configuration Parameters

Table 3-3 Low Power Interface Configuration Parameters

Label	Description
Low Power Interface Configuration	
Low Power Interface Enable	If true, the low-power handshaking interface (csysreq, csysack, and cactive signals) and associated control logic is implemented. If false, no support for low-power handshaking interface is provided. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** GM_LOWPWR_HS_IF
Number of inactive clock cycles before component requests power down	Number of AXI clock cycles to wait before cactive signal de-asserts, when there are no pending transactions. Note that if csysreq de-asserts while waiting this number of cycles, cactive de-asserts immediately. If a new transaction is initiated during the wait period, the counting will be halted, cactive will not de-assert, and the counting will be reinitiated when there are no pending transactions. This parameter is available only if GM_LOWPWR_HS_IF is true. **Values:** 0, ..., 4294967295 **Default Value:** 0 **Enabled:** GM_LOWPWR_HS_IF==1 **Parameter Name:** GM_LOWPWR_NOPX_CNT
Maximum number of outstanding read transactions	Maximum number of AXI read transactions that may be outstanding at any time Available only if GM_LOWPWR_HS_IF is true **Values:** 1, ..., 32 **Default Value:** 4 **Enabled:** GM_LOWPWR_HS_IF==1 **Parameter Name:** GM_MAX_PENDTRANS_READ
Maximum number of outstanding write transactions	Maximum number of AXI write transactions that may be outstanding at any time Available only if GM_LOWPWR_HS_IF is true **Values:** 1, ..., 32 **Default Value:** 4 **Enabled:** GM_LOWPWR_HS_IF==1 **Parameter Name:** GM_MAX_PENDTRANS_WRITE

3.4 Sideband/User Signals Parameters

Table 3-4 Sideband/User Signals Parameters

Label	Description
Sideband/User Signals	
Include Sideband/User Bus for Address Channels?	If set to True, then all AXI and GIF address channels have an associated sideband/user bus in AXI3/AXI4 mode, respectively. The read/write address channel sideband/user bus is routed in the same way as the other read/write address channel control signals. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DWC-AMBA-Fabric-Source license required **Parameter Name:** GM_HAS_ASB
Included Sideband Bus for Write Data Channels?	If set to True, then all AXI and GIF write data channels have an associated sideband/user bus in AXI3/AXI4 mode, respectively. The write data channel sideband/user bus is routed in the same way as the other write data channel control signals. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DWC-AMBA-Fabric-Source license required **Parameter Name:** GM_HAS_WSB
Include Sideband Bus for Response Channels?	If set to True, then all AXI and GIF read data and response channels have an associated sideband/user bus in AXI3/AXI4 mode, respectively. The read data channel and write response sideband/user bus is routed in the same way as the other read data channel and write response control signals. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DWC-AMBA-Fabric-Source license required **Parameter Name:** GM_HAS_RSB

Table 3-4 Sideband/User Signals Parameters (Continued)

Label	Description
Width	
Width of the Address Channel Sideband bus	When the GM_HAS_ASB parameter is set to True, you can set the address channel sideband/user bus width. **Values:** 1, ..., GM_MAX_SBW **Default Value:** 1 **Enabled:** GM_HAS_ASB == 1 **Parameter Name:** GM_A_SBW
Width of the Write Data Channel Sideband bus	When the GM_HAS_WSB parameter is set to True, you can set the write address channel sideband/user bus width. **Values:** 1, ..., GM_MAX_SBW **Default Value:** 1 **Enabled:** GM_HAS_WSB == 1 **Parameter Name:** GM_W_SBW
Width of the Response Channel Sideband bus	When the GM_HAS_RSB parameter is set to True, you can set the response channel sideband/user bus width. **Values:** 1, ..., GM_MAX_SBW **Default Value:** 1 **Enabled:** GM_HAS_RSB == 1 **Parameter Name:** GM_R_SBW

4 Signal Descriptions

This chapter details all possible I/O signals in the controller. For configurable IP titles, your actual configuration might not contain all of these signals.

Inputs are on the left of the signal diagrams; outputs are on the right.

Attention: For configurable IP titles, do not use this document to determine the exact I/O footprint of the controller. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the I/O signals for your actual configuration at workspace/report/IO.html or workspace/report/IO.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the I/O signals that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the widths might change depending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

In addition to describing the function of each signal, the signal descriptions in this chapter include the following information:

Active State: Indicates whether the signal is active high or active low. When a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the inactive state (opposite of the active state).

Registered: Indicates whether or not the signal is registered directly inside the IP boundary without intervening logic (excluding simple buffers). A value of No does not imply that the signal is not synchronous, only that there is some combinatorial logic between the signal's origin or destination register and the boundary of the controller. A value of N/A indicates that this information is not provided for this IP title.

Synchronous to: Indicates which clocks in the IP sample this input (drive for an output) when considering all possible configurations. A particular configuration might not have all of the clocks listed. This clock might not be the same as the clock that your application logic should use to clock (sample/drive) this pin. For more details, consult the clock section in the databook.

Exists: Names of configuration parameters that populate this signal in your configuration.

Validated by: Assertion or de-assertion of signals that validates the signal being described.

Attributes used with Synchronous To

- Clock name - The name of the clock that samples an input or drive and output.
- None - This attribute may be used for clock inputs, hard-coded outputs, feed-through (direct or combinatorial), dangling inputs, unused inputs and asynchronous outputs.
- Asynchronous - This attribute is used for asynchronous inputs and asynchronous resets.

The I/O signals are grouped as follows:

4.1 Clock and Resets Signals

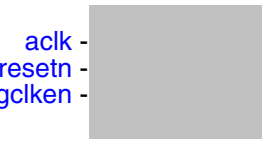


Table 4-1 Clock and Resets Signals

Port Name	I/O	Description
aclk	I	AXI clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aresetn	I	AXI reset. Asynchronous assertion and Synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of aclk. DW_axi_gm does not contain any logic to perform this synchronization; therefore, the logic must be provided externally. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
gclken	I	Clock enable signal. Allows operation on clock slower than aclk. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.2 GIF Request Channel Signals

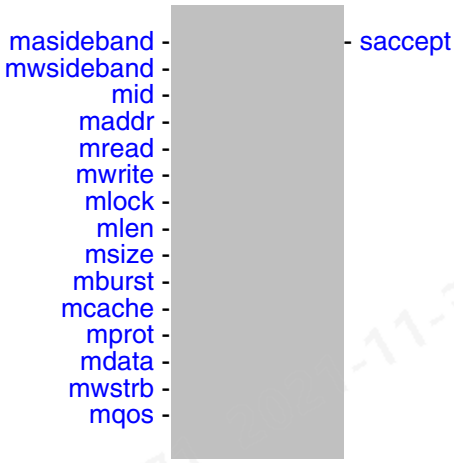


Table 4-2 GIF Request Channel Signals

Port Name	I/O	Description
saccept	O	GIF command accept. Slave accepts current mread or mwrite request by driving saccept high. On saccept high, address and write data are updated by master. Read bursts are single-cycle requests, whereas all other bursts are multi-cycle, requiring saccept to be sampled high for every beat. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
masideband[(GM_A_SBW_INT-1):0]	I	GIF Sideband signal for address. This signal is transferred to the AXI awsideband/awuser or arsideband/aruser signals based on the write/read operation. **Exists:** (GM_HAS_ASB == 1) **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 GIF Request Channel Signals (Continued)

Port Name	I/O	Description
mwsideband[(GM_W_SBW_INT-1):0]	I	GIF Sideband signal for write data. This signal is transferred to the AXI wsideband/wuser signal. **Exists:** (GM_HAS_WSB == 1) **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mid[(GM_ID-1):0]	I	GIF ID. This signal is transferred to the AXI channel signals awid, arid, and wid. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
maddr[(GM_AW-1):0]	I	GIF address. This signal is transferred to the AXI channel signals awaddr and araddr. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mread	I	GIF read indicator. Indicates valid read request. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
mwrite	I	GIF write indicator. Indicates valid write request. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
mlock	I	GIF locked access. This signal is used to determine the AXI channel signals, namely awlock and arlock. **Exists:** (GM_AXI_INTERFACE_TYPE == 0) **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-2 GIF Request Channel Signals (Continued)

Port Name	I/O	Description
mlen[(GM_BW-1):0]	I	GIF burst length. This signal is transferred to the AXI channel signals awlen and arlen. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
msize[2:0]	I	Size of transfer. This signal is transferred from the AXI channel signals awsize and arsize. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mburst[1:0]	I	GIF burst type. This signal is transferred to the AXI channel signals awburst and arburst. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mcache[3:0]	I	GIF cache type. This signal indicates the bufferable, cacheable/modifiable, write-through, write-back, and allocatable attributes of the transaction. The GIF cache type matches with the AXI cache type. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mprot[2:0]	I	GIF protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access. GIF protection type matches AXI protection type. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 GIF Request Channel Signals (Continued)

Port Name	I/O	Description
mdata[(GM_DW-1):0]	I	GIF write data. This signal is transferred to the AXI write data channel signal wdata. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mwstrb[((GM_DW/8)-1):0]	I	Write byte lane write strobes. This signal indicates byte lanes that must be updated in memory. One write strobe for each 8 bits of write data bus. MWSTRB[n] is associated with the following byte of MDATA: MDATA[8n+7: 8n] This signal is transferred to the AXI write data channel signal wstrb. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
mqos[3:0]	I	Optional. Quality of Service identifier, conveying priority information associated with each transaction on the Address channel of GIF Interface. **Exists:** (GM_AXI_HAS_QOS == 1) **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 GIF Response Channel Signals

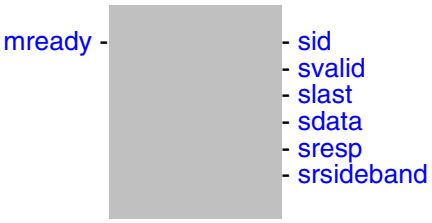


Table 4-3 GIF Response Channel Signals

Port Name	I/O	Description
sid[(GM_ID-1):0]	O	GIF response ID. The identification tag of the read and write responses. This signal is transferred from the AXI channel signals rid and bid. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
svalid	O	GIF valid Response indicator. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
slast	O	Slave last response indicator. This signal is transferred from the AXI read data channel signal rlast **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
sdata[(GM_DW-1):0]	O	GIF read data. This signal is transferred from the AXI read data channel signal rdata. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-3 GIF Response Channel Signals (Continued)

Port Name	I/O	Description
sresp[2:0]	O	GIF response. The encoding is as follows: ■ h'0 OK_R; Read transaction finished successfully ■ h'1 OK_W; Write transaction finished successfully ■ h'2 Reserved ■ h'3 Reserved ■ h'4 SLVERR_R; Addressed slave generated error for read access ■ h'5 SLVERR_W; Addressed slave generated an error for write access ■ h'6 DECERR_R; Decode error from interconnect on read access ■ h'7 DECERR_W; Decode error from interconnect on write access **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
srsideband[(GM_R_SBW_INT-1):0]	O	GIF Sideband signal for response. This signal is transferred from the AXI bsideband/buser or rsideband/ruser signal. **Exists:** (GM_HAS_RSB == 1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mready	I	GIF response accept. When master is not ready to accept response, it drives mready low. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 AXI Write Address Channel Signals

Table 4-4 AXI Write Address Channel Signals

Port Name	I/O	Description
awid[(GM_ID-1):0]	O	AXI write address identification. Gives identification tag for write address signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awvalid	O	AXI write address valid. Indicates valid write address and control information are available. Address and control information remain stable until awready signal is high. ■ 0: Address and control information not available. ■ 1: Address and control information available. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
awaddr[(GM_AW-1):0]	O	AXI write address. Specifies address of first transfer in write burst transaction. Associated control signals used to determine addresses of remaining transfers in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 AXI Write Address Channel Signals (Continued)

Port Name	I/O	Description
awlen[(GM_BW-1):0]	O	AXI Burst length. Specifies exact number of transfers in burst; determines number of data transfers associated with address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsize[2:0]	O	AXI write burst size. Indicates size of each transfer in burst. Byte lane strobes indicate exactly which byte lanes to update **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awburst[1:0]	O	AXI Write Burst. When combined with size information, shows how an address for each transfer within burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock[(GM_W_LTW-1):0]	O	AXI write lock. Provides additional information about characteristics of transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awcache[3:0]	O	AXI write cache. Indicates bufferable, cacheable, write-through, and write-back attributes of transaction. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 AXI Write Address Channel Signals (Continued)

Port Name	I/O	Description
awprot[2:0]	O	AXI write protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsideband[(GM_A_SBW_INT-1):0]	O	AXI Sideband signal for write address. **Exists:** (GM_HAS_ASB == 1&&GM_AXI_INTERFACE_TYPE==0) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awuser[(GM_A_SBW_INT-1):0]	O	AXI User signal for write address. **Exists:** (GM_HAS_ASB == 1&&GM_AXI_INTERFACE_TYPE==1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awqos[3:0]	O	Optional. Quality of Service identifier, conveying priority information associated with each transaction on Write Address channel of AXI Interface. **Exists:** (GM_AXI_HAS_QOS == 1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awready	I	AXI write address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 AXI Write Data Channel Signals

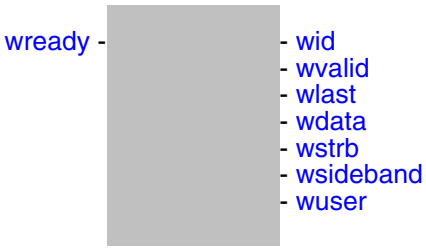


Table 4-5 AXI Write Data Channel Signals

Port Name	I/O	Description
wid[(GM_ID-1):0]	O	AMBA AXI3 write identification tag of write data transfer. **Exists:** (GM_AXI_INTERFACE_TYPE == 0) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wvalid	O	AXI write valid. This signal indicates that valid write data and strobes are available. ■ 0: Write data and strobes not available ■ 1: Write data and strobes available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wlast	O	AXI write last. Indicates the last transfer in a write burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wdata[(GM_DW-1):0]	O	AXI write data. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 AXI Write Data Channel Signals (Continued)

Port Name	I/O	Description
wstrb[((GM_DW/8)-1):0]	O	AXI write strobes. Indicates byte lanes that must be updated in memory. There is one write strobe for each eight bits of the write data bus. WSTRB[n] is associated with the following byte of WDATA: WDATA[8n+7: 8n] **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wsideband[(GM_W_SBW_INT-1):0]	O	AXI Sideband signal for write data channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named wsideband. ■ When the AXI4 interface is enabled, this signal is named wuser. This signal is transferred from the GIF masideband for write transactions. **Exists:** (GM_HAS_WSB == 1&&GM_AXI_INTERFACE_TYPE==0) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wuser[(GM_W_SBW_INT-1):0]	O	AXI User signal for write data channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named wsideband. ■ When the AXI4 interface is enabled, this signal is named wuser. This signal is transferred from the GIF masideband for write transactions. **Exists:** (GM_HAS_WSB == 1&&GM_AXI_INTERFACE_TYPE==1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wready	I	AXI write ready. Indicates that the slave can accept the write data. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.6 AXI Write Response Channel Signals

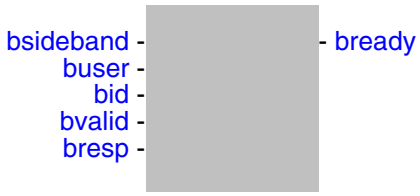


Table 4-6 AXI Write Response Channel Signals

Port Name	I/O	Description
bready	O	AXI response ready. Indicates master can accept response information. ■ 0: Master not ready ■ 1: Master ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
bsideband[(GM_R_SBW_INT-1):0]	I	Sideband signal for the write response channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named bsideband. ■ When the AXI4 interface is enabled, this signal is named buser. This signal is transferred to the GIF srsideband for write transactions. **Exists:** (GM_HAS_RSB == 1&&GM_AXI_INTERFACE_TYPE==0) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 AXI Write Response Channel Signals (Continued)

Port Name	I/O	Description
buser[(GM_R_SBW_INT-1):0]	I	AXI User signal for the write response channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named bsideband. ■ When the AXI4 interface is enabled, this signal is named buser. This signal is transferred to the GIF srsideband for write transactions. **Exists:** (GM_HAS_RSB == 1&&GM_AXI_INTERFACE_TYPE==1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bid[(GM_ID-1):0]	I	AXI write response identification tag. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bvalid	I	AXI write response valid. Indicates that valid write response is available. ■ 0: Write response not available ■ 1: Write response available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
bresp[1:0]	I	AXI write response. Indicates status of write transaction. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.7 AXI Read Address Channel Signals

Table 4-7 AXI Read Address Channel Signals

Port Name	I/O	Description
arid[(GM_ID-1):0]	O	AXI read address identification. Tag for read address signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arvalid	O	AXI read address valid. When high, indicates read address and control information is valid and remains stable until arready is high. ■ 0: Address and control information not valid ■ 1: Address and control information valid **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
araddr[(GM_AW-1):0]	O	AXI read address. Gives initial address of read burst transaction. Only start address of burst is provided, and control signals issued alongside address show how address is calculated for remaining transfers in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-7 AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arlen[(GM_BW-1):0]	O	AXI read burst length. The burst length provides the exact number of transfers in a burst and determines the number of data transfers associated with address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize[2:0]	O	AXI read burst size. Indicates size of each transfer in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arburst[1:0]	O	AXI read Burst. Combined with the size information, this shows how the address for each transfer within a burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock[(GM_W_LTW-1):0]	O	AXI read lock. Provides additional information about the characteristics of the transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache[3:0]	O	AXI read cache. Indicates bufferable, cacheable, read-through, and read-back attributes of transaction. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-7 AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arprot[2:0]	O	AXI read protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsideband[(GM_A_SBW_INT-1):0]	O	Sideband/user signal for the read address channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named arsideband. ■ When the AXI4 interface is enabled, this signal is named aruser This signal is transferred from the GIF masideband for read transactions. **Exists:** (GM_HAS_ASB == 1&&GM_AXI_INTERFACE_TYPE==0) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aruser[(GM_A_SBW_INT-1):0]	O	Sideband/user signal for the read address channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named arsideband. ■ When the AXI4 interface is enabled, this signal is named aruser. This signal is transferred from the GIF masideband for read transactions. **Exists:** (GM_HAS_ASB == 1&&GM_AXI_INTERFACE_TYPE==1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arqos[3:0]	O	Optional. Quality of Service identifier. Indicates priority of information associated with each transaction on Read Address channel of the AXI Interface. **Exists:** (GM_AXI_HAS_QOS == 1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-7 AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arready	I	AXI read address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.8 AXI Read Data Channel Signals

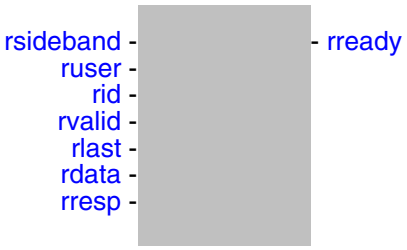


Table 4-8 AXI Read Data Channel Signals

Port Name	I/O	Description
rready	O	AXI read ready. Indicates master can accept read data and response information. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rsideband[(GM_R_SBW_INT-1):0]	I	Sideband signal for the read data channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named rsideband. ■ When the AXI4 interface is enabled, this signal is named ruser. This signal is transferred to the GIF srside-band for read transactions. **Exists:** (GM_HAS_RSB == 1&&GM_AXI_INTERFACE_TYPE==0) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-8 AXI Read Data Channel Signals (Continued)

Port Name	I/O	Description
ruser[(GM_R_SBW_INT-1):0]	I	User signal for the read data channel. The signal name changes between the AXI3 and the AXI4 configurations. ■ When the AXI3 interface is enabled, this signal is named rsideband. ■ When the AXI4 interface is enabled, this signal is named ruser. This signal is transferred to the GIF srside-band for read transactions. **Exists:** (GM_HAS_RSB == 1&&GM_AXI_INTERFACE_TYPE==1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rid[(GM_ID-1):0]	I	AXI Read identification tag. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rvalid	I	AXI read valid. Indicates that required read data is available and read transfer can complete. ■ 0: Read data not available ■ 1: Read data available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rlast	I	AXI read last. Indicates last transfer in read burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rdata[(GM_DW-1):0]	I	AXI read data **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-8 AXI Read Data Channel Signals (Continued)

Port Name	I/O	Description
rresp[1:0]	I	AXI read response. Indicates status of read transaction. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.9 AXI Low Power Interface Signals

Table 4-9 AXI Low Power Interface Signals

Port Name	I/O	Description
csysack	O	Low-power request acknowledgement. ■ De-asserted by DW_axi_gm to acknowledge request to enter low-power state. ■ Asserted by DW_axi_gm to acknowledge request to exit low-power state **Exists:** (GM_LOWPWR_HS_IF == 1) **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
cactive	O	Clock active request. De-asserted by DW_axi_gm to inform the system low-power controller (LPC) that the clock can be removed. ■ 1: Peripheral clock required. ■ 0: Peripheral clock not required. **Note**: Clock must be removed on positive edge after cactive and csysack signals are sampled low (0). **Exists:** (GM_LOWPWR_HS_IF == 1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
csysreq	I	System low-power request from system clock controller. ■ De-asserted by system low power controller (LPC) to initiate entry into a low power state. ■ Asserted to initiate exit from a low power state. **Exists:** (GM_LOWPWR_HS_IF == 1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

5 Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

Some expressions might refer to TCL functions or procedures (sometimes identified as **function_of**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Table 5-1 Internal Parameters

Parameter Name	Equals To
GM_A_SBW_INT	=(GM_HAS_ASB == 1 ? GM_A_SBW : 0)
GM_MAX_SBW	256
GM_R_SBW_INT	=(GM_HAS_RSB == 1 ? GM_R_SBW : 0)
GM_W_LTW	=(GM_AXI_INTERFACE_TYPE == 0 ? 2 : 1)
GM_W_SBW_INT	=(GM_HAS_WSB == 1 ? GM_W_SBW : 0)

6 Verification

This chapter provides an overview of the testbench available for DW_axi_gm verification. Once you have configured the DW_axi_gm in coreConsultant and have set up the verification environment, you can automatically run simulations.

Attention	These tests are provided only to verify the user's specific configuration and to provide reference waveforms so that the user can see how sample transactions affect input/output signal values. The tests cannot be modified, and the testbench cannot be reused as a starting point for a custom testbench.

6.1 Overview of Vera Tests

The DW_axi_gm verification testbench can perform the following tests:

- test_random
- test_lock

For each test executed, the run script writes all simulation output files to the respective *workspace*/sim/test_*name* directory. The runtest.log file in *workspace*/sim includes all simulation results. All of these files are visible from the Reports tab of the "Setup and Run Simulations" activity in the GUI, or they can be viewed directly from the text files in the *workspace*/sim directory.

6.1.1 test_random

This test verifies all system features, except locked access transactions, using randomly generated transactions. This test generates 10,000 GIF transactions and verifies all responses. Although the test uses random transactions, the random seed is statically determined from the current testbench and DUT setup. If neither testbench nor DUT are changed, the test runs exactly the same in consecutive simulations.

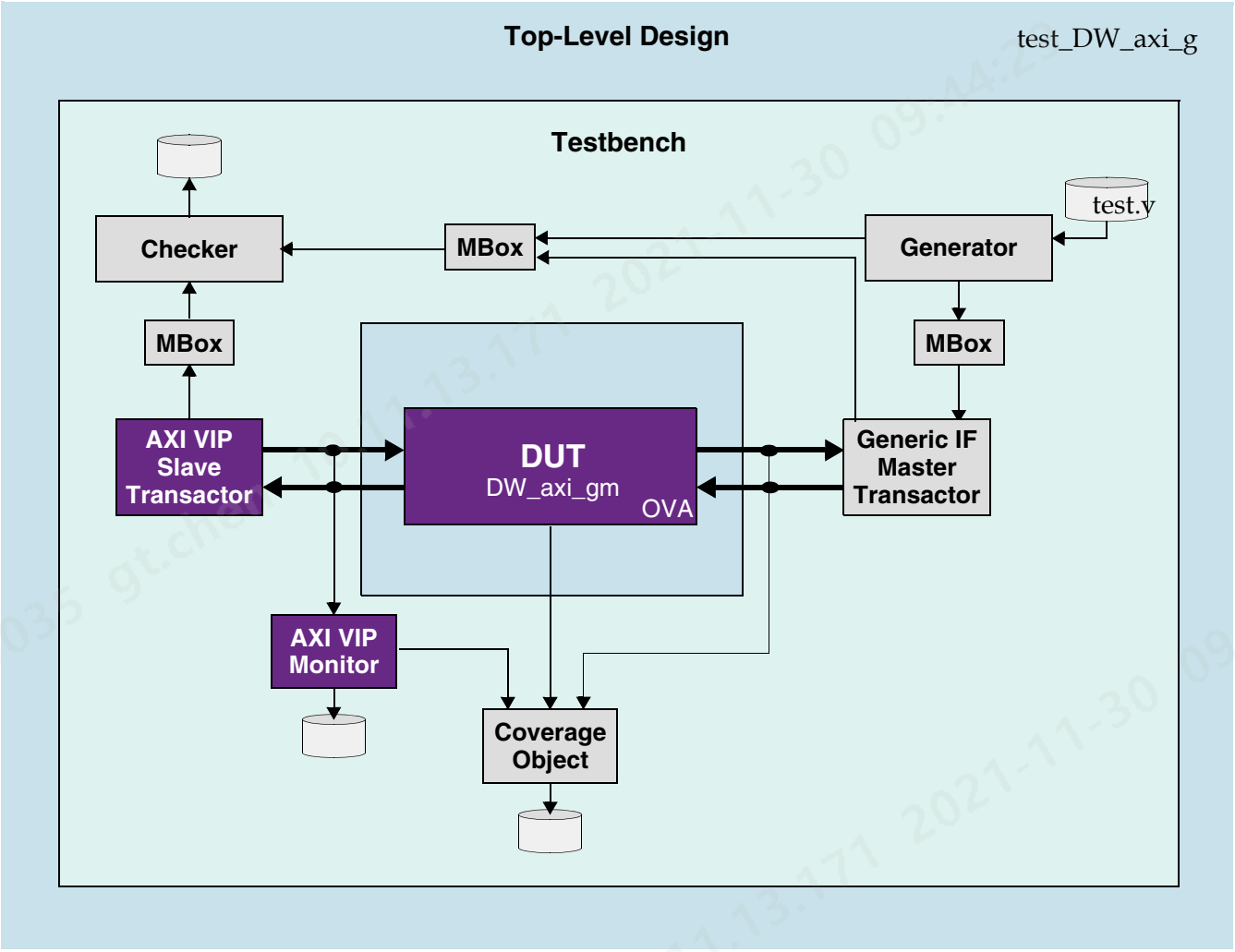
6.1.2 test_lock

This test verifies locked transactions using randomly generated instructions and is applicable only when the AXI3 mode is enabled. This test generates 10,000 mixed GIF locked and normal access transactions and verifies all responses. Although the test uses random transactions, the random seed is statically determined from the current testbench and DUT setup. If neither testbench nor DUT are changed, the test runs exactly the same in consecutive simulations.

6.2 Overview of DW_axi_gm Testbench

The following diagram illustrates the DW_axi_gm testbench, which applies random constraint testing and uses assertions in the DUT.

Figure 6-1 DW_axi_gm Testbench



The following define the testbench components:

- Generator – Produces stimulus to the design. Creates constrained random combinations of burst and single instructions, as well as specified instructions for directed tests.
- Transactors – Comprised of two transactors: Generic Interface (GIF) transactor and AXI VIP slave.

  The GIF transactor is a directed transactor that converts stimulus from the generator into binary vectors and drives it into the DUT. The GIF transactor also implements temporal aspects, such as rate of transactions.

The AXI VIP slave is an automatic transactor that reacts to AXI transactions. The slave is programmable and keeps record of the incoming executions in a local buffer, which can be accessed by the checker.

- Checker – Reads the AXI VIP buffer and compares the results with the expected queue; also known as the scoreboard.
- Mail Box – Used to pass information on the fly.
- Coverage Object – Records all coverage points that are hit by the testcases. It can extract coverage information directly from the AXI or from the OVA statements in the source code. The log file allows determination of the coverage grade of the DUT.

Most transaction testing is done with the DUT acting as a black box, while some tests determine specific implementation details. These tests use assertions, which can also be used to collect functional coverage information.

Results are checked using a predictive approach; that is, the reference is derived from the generated stimulus. No reference models for the DUT are used.

Each transaction generates two complete logs. The GIF master transactor logs the entire transaction, including response, and sends it to the checker once the transaction completes. The AXI VIP slave has watchpoints enabled, which are triggered once a transaction completes. The watchpoints also log the complete transaction, including response, and send it to the checker. The checker can then compare the originating and resulting portions of each transaction.

For example, the request phase for a burst originates from the GIF master, and the resulting request is logged on the AXI interface. For read data the situation is reverse. The read data originates from the AXI slave, and the resulting data is logged on the GIF.

Note If DW_axi_gm is configured for AXI4 mode, the testbench is automatically capable of supporting AXI4 traffic using AMBA VMT VIPs.

7 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment.

7.1 Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_axi_gm.

7.1.1 Power Consumption, Frequency, Area, and DFT Coverage

Table 7-1 provides information about the synthesis results (power consumption, frequency, area) and DFT coverage of the DW_axi_gm using the industry standard 16nm technology library.

Table 7-1 Synthesis Results for DW_axi_gm

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Default Configuration	400MHz	1863	1 nW	0.090 mW	100	99.83	99.9

Table 7-1 Synthesis Results for DW_axi_gm

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Typical Configuration - 1 GM_AXI_HAS_QOS = 1 GM_AXI_INTERFACE_TYPE = 0 GM_A_SBW = 64 GM_BLOCK_READ = 0 GM_BLOCK_WRITE = 1 GM_BW = 8 GM_DIRECT_AXI_READY = 1 GM_DIRECT_GIF_READY = 1 GM_DW = 512 GM_HAS_ASB = 1 GM_HAS_RSB = 1 GM_HAS_WSB = 1 GM_LOWPWR_HS_IF = 1 GM_LOWPWR_LEGACY_IF = 1 GM_REQ_BUFFER = 4 GM_RESP_BUFFER = 4 GM_R_SBW = 64 GM_WDATA_BUFFER = 4 GM_W_SBW = 64	400 MHz	60077	5 nW	2.670 mW	100	99.99	100
Typical Configuration - 2 GS_AW = 64 GS_AXI_EX_ACCESS = 16 GS_AXI_INTERFACE_TYPE= 1 GS_BID_BUFFER = 4 GS_BW = 8 GS_DIRECT_AXI_READY = 1 GS_DIRECT_GIF_READY = 1 GS_DW = 8 GS_GIF_LITE = 0 GS_ID = 13 GS_LOWPWR_HS_IF = 0 GS_LOWPWR_LEGACY_IF = 0 GS_REQ_BUFFER = 4 GS_RESP_BUFFER = 4 GS_RID_BUFFER = 4 GS_WDATA_BUFFER = 4	400 MHz	51470	4 nW	2.330 mW	100	99.99	100

8 Basic Core Module (BCM) Library

The Basic Core Module (BCM) Library is a library of commonly used blocks for the Synopsys DesignWare IP development. These BCMs are configurable on an instance-by-instance basis and, for the majority of BCM designs, there is an equivalent (or nearly equivalent) DesignWare Building Block (DWBB) component.

This chapter provides more information about the BCMs used in DW_axi_gm.

8.1 BCM Library Components

Table 8-1 describes the list of BCM library components used in DW_axi_gm.

Table 8-1 List of BCM Library Components used in the Design

BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_gm_bcm06	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctl_s1_df
DW_axi_gm_bcm57	Synchronous Write-Port, Async. Read-Port RAM (Flip-Flop-Based)	DW_ram_r_w_s_dff
DW_axi_gm_bcm65	Synchronous (Single Clock) FIFO with Static Flags	DW_fifo_s1_sf

A Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
active AXI command	A command that has been generated by an AXI master and accepted by an AXI slave, but where the corresponding command has not completed. A read command completes when the last data beat of the read burst completes; that is, when the AXI slave asserts RLAST and RVALID and when the AXI master asserts RREADY. A write command completes when the AXI slave returns the write response and is accepted by the AXI master; that is, when the AXI slave asserts BVALID and the AXI master asserts BREADY
active DW_axi_gm command	A command that has been generated by an AXI master on the primary bus and accepted by an AXI slave on the secondary bus but where the corresponding command has not completed. In terms of the DW_axi_gm, a read command completes when the last data beat of the read burst completes on the primary bus; that is, when the DW_axi_gm Master Port asserts RLAST and RVALID and when the external AXI master asserts RREADY. In terms of the DW_axi_gm, a write command completes when the DW_axi_gm Master Port returns the write response and it is accepted by the AXI master on the primary bus; that is, when the DW_axi_gm Master Port asserts BVALID and the external AXI master asserts BREADY.
activity	A set of functions in coreConsultant that step you through configuration, verification, and synthesis of a selected core.
A-device model	A usb_device_vmt model configured as SRP capable host only device and dual role OTG A-device.
AHB	Advanced High-performance Bus — high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces (ARM Limited specification).
altered transaction	Transaction that the DW_axi_gm has either downsized, upsized, endian-mapped, or fanned out to multiple write interleaving channels. Similarly, a non-altered transaction is a transaction that none of these operations are performed on.

AMBA	Advanced Microcontroller Bus Architecture — a trademarked name by ARM Limited that defines an on-chip communication standard for high speed microcontrollers.
APB	Advanced Peripheral Bus — optimized for minimal power consumption and reduced interface complexity to support peripheral functions (ARM Limited specification).
APB bridge	DW_apb submodule that converts protocol between the AHB bus and APB bus.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
arbiter	AMBA bus submodule that arbitrates bus activity between masters and slaves.
AXI	Advanced eXtensible Interface Bus — a trademarked name by ARM Limited that defines an on-chip protocol for high-performance, high-frequency system designs.
B-device model	A usb_device_vmt model configured as non-OTG device (USB 2.0 standard function), SRP capable peripheral only device and dual role OTG B-device.
backward control path	For the read/write command channels and the write data channel, the backward control path is in the direction from ARREADY/AWREADY/WREADY from an external slave to ARREADY/AWREADY/WREADY to an external master. For the read data and write response channels, the backward control path is in the direction from RREADY/BREADY from an external master to RREADY/BREADY to an external slave
BFM	Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.
beat	A single data transfer, usually in a burst transaction. For example, on the AHB, an INCR4 transaction has four beats and INCR8 has eight beats.
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.
blocked command stream	A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.
blocked transaction	Transaction is considered blocked when it is not initiated before the preceding transaction has completed.
blocking command	A command that prevents a testbench from advancing to the next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.
burst	Number of data transfers in a transaction.
bus bridge	Logic that handles the interface and transactions between two bus standards, such as AHB and APB. See APB bridge.

command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command queue	First-in-first-out queue from which a model command execution engine retrieves commands; see also active command queue. VMT models support multiple command queues.
command stream	The communication channel between the testbench and the model.
component	A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.
configuration	The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.
configuration intent	Range of values allowed for each parameter associated with a reusable core.
core	Any configurable block of synthesizable IP that can be instantiated as a single entity (VHDL) or module (Verilog) in a design. Core is the preferred term for a big piece of IIP. Anything that requires coreConsultant for configuration, as well as anything in the DesignWare Cores library, is a core.
core developer	Person or company who creates or packages a reusable core. All the cores in the DesignWare Library are developed by Synopsys.
core integrator	Person who uses coreConsultant or coreAssembler to incorporate reusable cores into a system-level design.
coreAssembler	Synopsys product that enables automatic connection of a group of cores into a subsystem. Generates RTL and gate-level views of the entire subsystem.
coreConsultant	A Synopsys product that lets you configure a core and generate the design views and synthesis views you need to integrate the core into your design. Can also synthesize the core and run the unit-level testbench supplied with the core.
coreKit	An unconfigured core and associated files, including the core itself, a specified synthesis methodology, interfaces definitions, and optional items such as verification environment files and core-specific documentation.
cycle command	A command that executes and causes HDL simulation time to advance.
buffered write transaction	Occurs when the write response is provided by an intermediate instance and not by the actual payload sink. The actual write response is typically discarded unless it indicates an erroneous completion.
byte reordering	Changing the location of bytes in the data word across the bridge (little-endian to big-endian).
data interleaving	Refers to a data channel (read or write) source issuing data for different transactions without waiting for the data beats of previous transactions to complete.

decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.
design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
Design View	A simulation model for a core generated by coreConsultant.
design_dir	The VIP equivalent to workspace.
DesignWare Synthesizable Components for AMBA	The Synopsys name for the collection of AMBA-compliant coreKits and verification models delivered with DesignWare and used with coreConsultant or coreAssembler to quickly build DesignWare Synthesizable Component designs.
DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWareSynthesizable Components for AMBA.
downsizing	Functional operation of the DW_axi_gm bridge in a configuration where the primary bus data width or burst length width is larger than those of the secondary bus. Occurs where a transaction entering the DW_axi_gm from the primary bus may have its LEN and/or SIZE attributes changed or broken into smaller transactions.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
forward control path	For the read/write command channels and the write data channel, the forward control path is in the direction from AWVALID/ARVALID/WVALID from an external master to AWVALID/ARVALID/WVALID to an external slave. For the read data and write response channels, the forward control path is in the direction from RVALID/BVALID from an external slave to RVALID/BVALID to an external master.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
gasket	A model that serves as a connection layer between the VMT models and the DP/DM interface.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.

HNP	Host Negotiation Protocol.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
interleaving depth	Refers to the number of different data parts of transactions that can be active at any one time on that channel (read or write data). Interleaved transactions must have different IDs.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
locked sequence	A sequence of locked read or write commands from an external AXI master locking the read and write command channels of the addressed slave. The locking sequence includes both the initial locking command and the final unlocking command that is used to terminate the locked sequence.
locking command	Initial locking command of the locked sequence.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
master port	Port of the DW_axi_gm that connects to an external master on the primary bus.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.
non-blocking command	A testbench command that advances to the next testbench statement without waiting for the command to complete.
non-transaction altering configuration	Configuration of the DW_axi_gm bridge where altered transactions are not possible.
non-OTG device	The Device model configured as USB 2.0 standard function and does not contain OTG features.
OTG	On-The-Go.
payload, AXI	For the read/write command channels, the read/write address and control signals. For the read/write data channels, the read/write data and control. For the write response channel, the burst response and control. The payload is always in the direction of the forward control path and is specific to the AXI channel.

payload source	Source of the AXI channel payload. The payload source for the read/write command channels and the write data channel is an AXI master. The payload source for the read data and write response channel is an AXI slave.
payload sink	Sink of the AXI channel payload. The payload sink for the read/write command channels and the write data channel is an AXI slave. The payload sink for the read data and write response channel is an AXI master.
peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.
posted transaction	Transaction is considered posted when it is initiated before the preceding transaction has completed.
primary and secondary	Refers to the buses with respect to a particular instantiation of the DW_axi_gm. Primary and secondary is in reference to the direction of the initiation of transactions through the DW_axi_gm. Primary refers to the transaction issuing bus, and secondary is the transaction responding bus.
read transaction	Composed of the following two independent phases: 1. Read command phase 2. Read data phase; signalling of the last beat of read data terminates read transaction
resized transaction	Transaction that the DW_axi_gm has either upsized or downsized.
RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
secondary and primary	Refers to the buses with respect to a particular instantiation of the DW_axi_gm. Primary and secondary is in reference to the direction of the initiation of transactions through the DW_axi_gm. Primary refers to the transaction issuing bus, and secondary is the transaction responding bus.
slave	Device or model that is controlled by and responds to a master.
slave port	Port of the DW_axi_gm that connects to an external slave on the secondary bus.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
sparse data	Data bus data which is individually byte-enabled. The AXI bus allows sparse data transfers using the WSTRB signal, each byte lane individually enabled. The AHB bus does not allow sparse data transfers.
SRP	Session Request Protocol.

static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
subsystem	In relation to coreAssembler, highest level of RTL that is automatically generated.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
transaction	A read or write operation on the AXI or AHB that involves one or more data transfers. For example, on the AHB, an INCR8 write is considered as a single transaction; a SINGLE read or an INCR burst of unspecified length is also considered a single transaction. Any AXI read or write is considered to be a single transaction, regardless of its length.
transaction ordering	Order in which transactions are responded to. Responses to a series of transactions could be returned in an order different to the order in which they were issued; this is referred to as an out-of-order transaction. Re-ordered transactions must have different IDs.
transfer	A single sequence initiated by VALID and ended by READY. A write command phase, read command phase, write data phase, read data phase, write response phase are each, by themselves, a transfer.
upsizing	Functional operation of the DW_axi_gm bridge in a configuration where the primary bus data width is smaller than the secondary bus data width. Occurs here a transaction entering the DW_axi_gm from the primary bus may have its LEN and/or SIZE attributes changed to transform it into a transaction of the largest SIZE allowable on the secondary bus.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
wait state	A bus cycle where the device initiating the transfer must wait for a response.
waited transfer	AXI channel is waited if the payload sink de-asserts its *READY signal when the payload source asserts its *VALID signal. Waits states are inserted into the AXI channel transfer.
workspace	A network location that contains a personal copy of a component or subsystem. After you configure the component or subsystem (using coreConsultant or coreAssembler), the workspace contains the configured component/subsystem and generated views needed for integration of the component/subsystem at the top level.

wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
write transaction	Composed of the following three independent phases: 1. Write command phase 2. Write data phase 3. Write response phase; terminates the write transaction
zero-cycle command	A command that executes without HDL simulation time advancing.

Index

Z