Summary of Functional Testing and Integration Work Done for the DARPA CHIPS Program by New Wave Design and Verification

Document Number: 900-03-150
Revision: 1.0
Release Date: 9 Jan 2019

Prepared by:

New Wave Design & Verification LLC

4950 W 78th Street

Minneapolis, MN 55435

Record of Revisions

Version	Date	Description
1.0	01/09/2019	Initial Release

Table of Contents

List of Figures

1. Introduction

The goal of the CHIPS program is to establish and demonstrate a modular design and fabrication flow for electronic systems while addressing the rising cost, lead-time, and complexity of IC design. The central idea is that a system can be subdivided into functional circuit blocks or chiplets that are reusable IP blocks, where an IP block refers to a pre-designed functional circuit block realized in physical form. Complete electronic systems can then be created through integration of chiplets on an interposer, rather than through the design and fabrication of circuits in a monolithic flow. This modular design flow is anticipated to encompass rapid assembly and reconfiguration of various IP blocks through standard layouts and interfaces seamlessly linked to other chiplets. This common framework is expected to expand access to a large catalog of commercial off-the-shelf (COTS)/Government off-the-shelf (GOTS) IP blocks, allow reuse of existing IP blocks, and speed heterogeneous integration of blocks in other technologies and nodes. The overall CHIPS program is envisioned as the establishment, demonstration, and iteration of modular design flows enabled by interface standardization.

2. References

A. AIB Architecture Overview, V0.99
B. TLRB AIB PHY Design Preliminary Specification, Rev 0.7
C. Chiplet Protocol Interface (CPI), Rev 0.80

3. Functional Simulation

3.1 The Purpose of Functional Simulation

Functional simulation entails the construction of a test bench to stimulate a digital device-under-test (DUT) and check that the resulting outputs match the expected outputs. In ASIC or ASIP development, functional simulation is particularly important to ensure that the DUT, when fabricated, operates as expected. DUT fabrication is expensive and time-consuming. Functional simulation is a key factor in helping to ensure that the received fabricated DUT operates properly.

3.2 The Test Bench DUT

The test bench DUT (see Figure 1, below) consists of a CPI/AIB master connected to a CPI/AIB slave via Intel AIB micro-bumps. The interface to the outside world is an AXI4 master bus and an AXI4 slave bus. The master/slave pair behave as an AXI4 passthrough to the outside world. The CPI/AIB master/slave pair in the test bench DUT represent the communication link between *two different ASIC's*. On the master and slave ASIC's, a wide variety of circuits could be attached to the respective master or slave AXI4 busses. This test environment only checks the communication link, not prospective circuitry that might be attached to the master or slave AXI4 buses. Registers in the master and slave AIB's can be accessed via their respective APB buses.

A JTAG interface is also present for both the master and slave AIB's to control boundary scan operations within the AIB bumps.

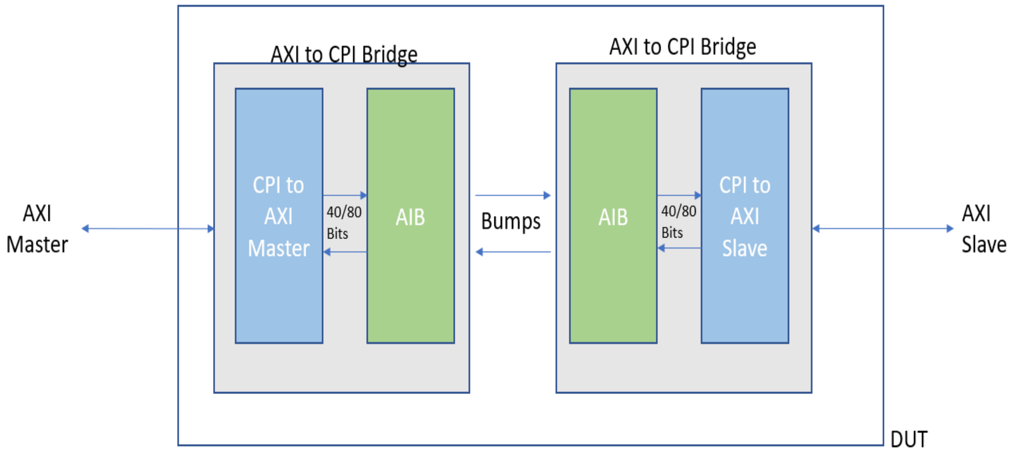

Figure 1: Test bench DUT

3.3 Test Approach

Once configured in Verilog RTL, a module for the DUT was created in the Xilinx Vivado block design environment. Vivado verification IP (VIP) for the AXI4 master and slave interfaces were then attached in the block design environment to the appropriate DUT channels. APB master verification blocks were written in SystemVerilog and corresponding modules were created in the block design environment. The APB modules were attached to the master and slave APB channels in the block design. See Figure 2, below.

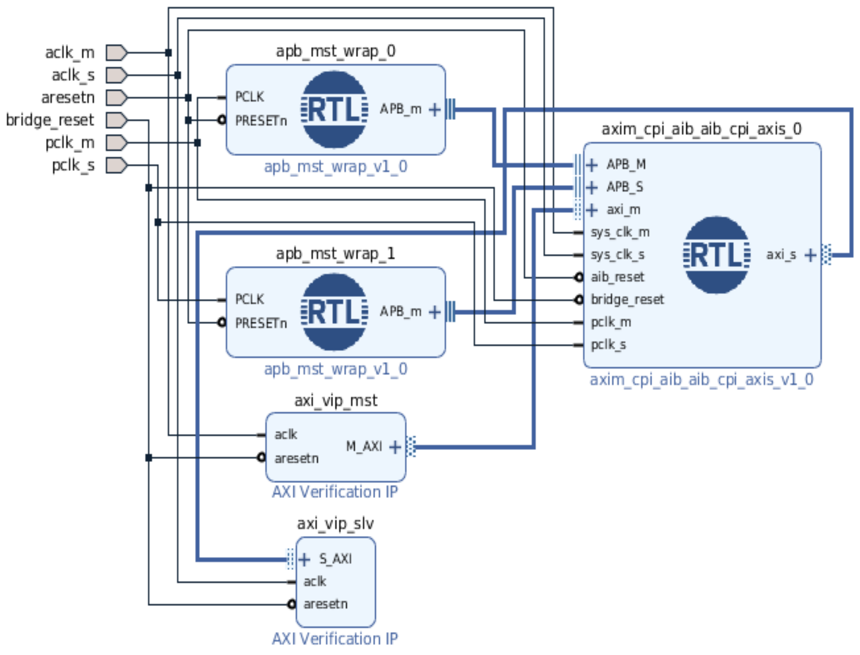

Figure 2: Test Bench with VIP and DUT

3.3.1 Use of Verification IP

Vivado AXI verification IP was used for the testing of the AXI master and slave buses. The use of verification IP to test standard bus interfaces has several advantages:

1) The AXI protocol is complex and testing this protocol would require great effort if done from scratch. Use of UVM AXI VIP's allows the leveraging of previous work and the extension of this work, if necessary.
2) The Vivado AXI VIP provides constrained-randomized stimulus which tends to find issues that would only be found at higher levels of integration.
3) The VIP also provides protocol rule checking via assertions that are derived from the AXI protocol functional definition.

3.3.2 AXI Bus Activity

1) Single AXI writes and reads are initially run with variable lengths between one and 8 bytes. The AXI data width is set to 128 bits.
2) Burst mode writes and reads are run with legal lengths of 1, 2, 3, 4, 8 and 16 128-bit beats.
3) Writes and reads are allowed to time-overlap to test the ability of the DUT AXI passthrough to handle simultaneous read and write activity.

3.3.3 Test Flexibility

1) As long as the interfaces to the test bench DUT remain the same (AXI, APB, JTAG), minor RTL changes may be accommodated without having to make major changes to the test environment.
2) A large list of parameters (0) can be adjusted to test different design configurations.

3.3.4 Multiple CAD Tool Environment

Two CAD tool suites were used in creating the New Wave simulation environment for DARPA CHIPS, Xilinx Vivado and Mentor Questasim. Xilinx Vivado allows for the quick assembly of the simulation environment using its block design tool. Vivado also provides the AXI Master and Slave VIP as starting points for the test stimulus. The block design tool allows the AXI VIP to be quickly connected to a module created in Vivado for the simulation DUT.

The circuit constructed from the Vivado block design tool (DUT + VIP) is passed to the Mentor Questasim simulator for simulation and analysis. Questasim was chosen for the actual simulation for the following reasons:

1) Questasim provides a better debugging environment than Vivado for the analysis of logic issues and UVM test bench issues.
2) Questasim allows the randomization seed for constrained-random testing to be altered. Vivado does not.
3) Questasim provides a rich code coverage environment for the computation, merging, filtering and displaying of code coverage metrics.

3.3.5 Simulation Corners

The simulations of the CHIPS test DUT were run over multiple corners. The variables for the corners were: DDR/SDR mode, CPI/AIB data width = 40 or 80 bits and randomization seed.

3.3.6 Conversion to a Scripted Methodology

The simulation environment was initially constructed using the Vivado block design tool within the Vivado GUI. This environment was then converted to a script-based environment using TCL, Bash, and Python scripts. The initial TCL script was generated from the Vivado GUI using the "write project TCL" command. The scripts allowed for the simulation DUT and the ASIC target AXI slave DUT to be repeatably built. The "git" configuration management tool was used to place all simulation source files, including build scripts to be version-controlled.

3.4 Test Bench Architecture

The test bench DUT is a back-to-back master/slave CPI/AIB pair shown in Figure 1, above.

3.4.1 Master/Slave Bump Mapping

The interface between the master and slave AIB's is a mapping between the bumps described in Tables 8 and 9 of the TLRB AIB PHY DESIGN PRELIMINARY SPECIFICATION. The mappings in these tables were converted to Verilog in the simulation DUT top-level source file.

3.4.2 Clocking Approach

The clocking approach used in the simulation DUT follows that described in Figure 8.3, option 1 of the AIB Architecture Overview. The external clock passes into the master via the CPI clk and AIB tx_clk inputs and transfers to the slave via the bumps. The recreated slave clock is passed out the slave's AIB rx_clk output. This recreated clock wraps back into the slave clk CPI input and the slave AIB tx_clk input.

3.4.3 Reset Approach

In normal operation, the CPI portion of the bridge should not be released from reset until the AIB portion of the bridge has been reset and configured via the APB bus. The preferred way to accomplish this is to first reset the APB bus (presetn), and perform the proper configuration writes to the APB, culminating in a write to the AIB reset register to release ADAPTER_RSTN. The ADAPTER_RSTN value from this register is fed out of the AIB to the CPI via the CPI's rx_rd_rstb and tx_wr_rstb reset inputs, guaranteeing that the CPI does not begin to run until the AIB reset sequence is complete.

3.5 Latency Measurement

The latency of AXI writes and reads is of interest to the user of the CHIPS CPI/AIB interface because it indicates the response time that can be supported by the system. To measure write latency, the times of the master writes of the last data beats in frames along with the write data values are captured. The times of the slave writes of last data beats in frames along with the write data values are also captured. The write latency from master to slave is the time difference between the master and the slave writes of the *same data value* (assumed to be well-randomized). The read latency is computed in a similar manner on randomized data beat passing from the slave to the master. A spreadsheet is used to pair up writes and reads of the same data values and compute the time differences. The maximum, minimum and average write and read latency times is computed by the spreadsheet. The write and read stimulus are comprised of about 200 overlapped writes and reads to give a realistic and randomized profile of typical system performance.

3.6 Coverage Approach

Various coverage metrics can be captured by the Questasim simulator and displayed in a readable HTML format for hyperlinked viewing using a common internet browser. Coverage metrics indicate how well the test stimulus covers the intent of the design as well as the physical coding of the design.

3.6.1 Coverage Merging

The Questasim tool allows the coverage data to be merged from multiple runs (over the simulation corners described above). Coverage data can also be merged over multiple instances of the same module. Both forms of merging were used in the coverage analysis.

3.6.2 Coverage Metrics

Coverage metrics are divided into various bins. The bins of interest are described below:

i. Assertion coverage—Indicates how well the design intent is covered, since the AXI VIP contain assertions that check that the AXI specifications.

ii. Statement coverage—Indicates how thoroughly the lines of RTL code have been traversed.

iii. Branch coverage—Indicates how thoroughly branches in the RTL code have been traversed.

iv. FSM state coverage—Indicates whether all states in an FSM have been reached.

v. FSM state transition—Indicates whether all state transitions in an FSM have been traversed.

vi. Cover groups, cover directives—User-specified coverage metrics to refine coverage collection and reporting in specific areas.

3.6.3 Coverage Exclusion

The Questasim tool allows various coverage metrics to be excluded from the output. Common exclusions that were made are described below:

i. Assertion exclusions: The primary assertion exclusions involved unused AXI functionality e.g. AXI4Lite behavior.

ii. FSM transition exclusions: The primary FSM transition exclusions involved transitions from arbitrary FSM states to the reset state. These transitions are essentially covered when the design clears X states after reset, but the tool does not automatically reflect this.

iii. Statement exclusions: The primary statement exclusions involved code that does not use 128-bit AXI. 32, 64 and 256-bit AXI cases were coded but will not be built in the ASIC.

iv. Branch exclusions: The primary branch exclusions involved code that does not use 128-bit AXI. 32, 64 and 256-bit AXI cases were coded but will not be built in the ASIC.

v. Verification code exclusions: The primary verification target is the RTL DUT, not the verification code. Therefore, the bulk of the verification code was excluded from the coverage assessment. Assertions, cover groups and cover directives found in the verification code *were* included in the coverage assessment.

4. Integration

New Wave performed the RTL coding to integrate master and slave CPI/AIB combined blocks. Micron provided the RTL code for the CPI blocks and Intrinsix provided the RTL code for the AIB blocks.

4.1 CPI/AIB Master Integrated Block

The CPI/AIB master integrated block (master_cpi_aib.sv), combines a CPI master bridge (Micron) with a master-configured AIB bridge (Intrinsix). The master_cpi_aib supports both 40- and 80-bit interfacing between the CPI and AIB. The master_cpi_aib also supports DDR and SDR modes of operation. A primary function of the master_cpi_aib block is to provide the necessary multiplexing/demultiplexing of the tx_data and rx_data trunks between the CPI and AIB master bridges to support the four SDR/DDR; 40/80-bit modes of operation.

4.1.1 CPI/AIB Master Block Port List

The following ports comprise the I/O's for the master_cpi_aib block:

Port Name	**Bit Range**	**Direction**	**Description**
clk			Clock for the CPI bridge **(Begin CPI ports)**
axi_m_araddr	[AR_AWIDTH-1:0]	input	AXI master read address channel address
axi_m_arid	[AR_IDWIDTH-1:0]	input	AXI master read address channel Id
axi_m_arlen	[7:0]	input	AXI master read address channel burst length
axi_m_arsize	[2:0]	input	AXI master read address channel burst size
axi_m_arburst	[1:0]	input	AXI master read address channel burst type
axi_m_aruser	[AR_USRWIDTH-1:0]	input	AXI master read address channel user
axi_m_arlock		input	AXI master read address channel lock
axi_m_arqos	[3:0]	input	AXI master read address channel qos
axi_m_arregion	[3:0]	input	AXI master read address channel region
axi_m_arprot	[2:0]	input	AXI master read address channel prot

Port Name	Bit Range	Direction	Description
axi_m_arvalid		input	AXI master read address channel valid
axi_m_arready		output	AXI master read address channel ready
axi_m_awaddr	[AW_AWIDTH-1:0]	input	AXI master write address channel address
axi_m_awid	[AW_IDWIDTH-1:0]	input	AXI master write address channel Id
axi_m_awlen	[7:0]	input	AXI master write address channel burst length
axi_m_awsize	[2:0]	input	AXI master write address channel burst size
axi_m_awburst	[1:0]	input	AXI master write address channel burst type
axi_m_awuser	[AW_USRWIDTH-1:0]	input	AXI master write address channel user
axi_m_awlock		input	AXI master write address channel lock
axi_m_awqos	[3:0]	input	AXI master write address channel qos
axi_m_awregion	[3:0]	input	AXI master write address channel region
axi_m_awprot	[2:0]	input	AXI master write address channel prot
axi_m_awvalid		input	AXI master write address channel valid
axi_m_awready		output	AXI master write address channel ready
axi_m_wdata	[W_DWIDTH-1:0]	input	AXI master write data channel data
axi_m_wstrb	[(W_DWIDTH/8)-1:0]	input	AXI master write data channel strobe
axi_m_wlast		input	AXI master write data channel last
axi_m_wuser	[W_USRWIDTH-1:0]	input	AXI master write data channel user
axi_m_wvalid		input	AXI master write data channel valid
axi_m_wready		output	AXI master write data channel ready
cpi_source_id	[7:0]	input	CPI Source component Id for this bridge
axi_m_rid	[R_IDWIDTH-1:0]	output	AXI master read response channel Id
axi_m_rdata	[R_DWIDTH-1:0]	output	AXI master read response channel data
axi_m_rlast		output	AXI master read response channel last
axi_m_rresp	[1:0]	output	AXI master read response channel response
axi_m_ruser	[R_USRWIDTH-1:0]	output	AXI master read response channel user
axi_m_rvalid		output	AXI master read response channel valid
axi_m_rready		input	AXI master read response channel ready
axi_m_bid	[B_IDWIDTH-1:0]	output	AXI master write response channel Id
axi_m_bresp	[1:0]	output	AXI master write response channel response
axi_m_buser	[B_USRWIDTH-1:0]	output	AXI master write response channel user
axi_m_bvalid		output	AXI master write response channel valid
axi_m_bready		input	AXI master write response channel ready
cr_init_delay	[63:0]	input	Number of clock cycles to delay credit init
ubump	[NBMP-1:0]	inout	IO channel micro-bumps **(Begin AIB ports)**
ubump_aux	[1:0]	inout	AUX channel micro-bumps
tx_clk		input	AIB clock input
rx_clk		output	AIB clock output generated from micro-bumps
conf_done		input	Configuration complete from all AIB bridges in system
config_done		output	Configuration complete for this AIB bridge
device_detect_ovrd		input	DFT: Overrides device detect function
por_ovrd		input	DFT: POR override for water test (0=override)

Port Name	Bit Range	Direction	Description
pclk		input	APB clock
presetn		input	APB reset
paddr	[11:0]	input	APB address
pwrite		input	APB direction
psel		input	APB select
penable		input	APB enable
pwdata	[31:0]	input	APB write data
prdata	[31:0]	output	APB read data
pready		output	APB bus ready
tdi		input	Test data in (JTAG)
tck		input	Test clock
tms		input	Test mode select
tdo		output	Test data out
trstn_or_por_rstn		input	JTAG resetn or por_rstn
rx_error		output	Receive parity error
rx_error_data	[DWIDTH-1:0]	output	Receive parity error data
tx_rlen_error		output	Transmit read length error
tx_rlen_error_sz	[2:0]	output	Transmit read length error AXI size
tx_rlen_error_len	[7:0]	output	Transmit read length error AXI length
tx_wlen_error		output	Transmit write length error
tx_wlen_error_sz	[2:0]	output	Transmit write length error AXI size
tx_wlen_error_len	[7:0]	output	Transmit write length error AXI length
dig_test_sel		input	Enable digital test bus
dig_test_bus	[7:0]	output	Output DFT: Debug signals from AIB IO

4.2 CPI/AIB Slave Integrated Block

The CPI/AIB slave integrated block (slave_cpi_aib.sv), combines a CPI slave bridge (Micron) with a slave-configured AIB bridge (Intrinsix). The slave_cpi_aib supports both 40 and 80-bit interfacing between the CPI and AIB. The slave_cpi_aib also supports DDR and SDR modes of operation. A primary function of the slave_cpi_aib block is to provide the necessary multiplexing/demultiplexing of the tx_data and rx_data trunks between the CPI and AIB slave bridges to support the four SDR/DDR; 40/80-bit modes of operation.

4.2.1 CPI/AIB Slave Block Port List

The following ports comprise the I/O's for the slave_cpi_aib block:

Port Name	Bit Range	Direction	Description
clk			Clock for the CPI bridge **(Begin CPI ports)**
axi_s_araddr	[AR_AWIDTH-1:0]	output	AXI slave read address channel address
axi_s_arid	[AR_IDWIDTH-1:0]	output	AXI slave read address channel Id
axi_s_arlen	[7:0]	output	AXI slave read address channel burst length
axi_s_arsize	[2:0]	output	AXI slave read address channel burst size

Port Name	Bit Range	Direction	Description
axi_s_arburst	[1:0]	output	AXI slave read address channel burst type
axi_s_aruser	[AR_USRWIDTH-1:0]	output	AXI slave read address channel user
axi_s_arlock		output	AXI slave read address channel lock
axi_s_arqos	[3:0]	output	AXI slave read address channel qos
axi_s_arregion	[3:0]	output	AXI slave read address channel region
axi_s_arprot	[2:0]	output	AXI slave read address channel prot
axi_s_arvalid		output	AXI slave read address channel valid
axi_s_arready		input	AXI slave read address channel ready
axi_s_awaddr	[AW_AWIDTH-1:0]	output	AXI slave write address channel address
axi_s_awid	[AW_IDWIDTH-1:0]	output	AXI slave write address channel Id
axi_s_awlen	[7:0]	output	AXI slave write address channel burst length
axi_s_awsize	[2:0]	output	AXI slave write address channel burst size
axi_s_awburst	[1:0]	output	AXI slave write address channel burst type
axi_s_awuser	[AW_USRWIDTH-1:0]	output	AXI slave write address channel user
axi_s_awlock		output	AXI slave write address channel lock
axi_s_awqos	[3:0]	output	AXI slave write address channel qos
axi_s_awregion	[3:0]	output	AXI slave write address channel region
axi_s_awprot	[2:0]	output	AXI slave write address channel prot
axi_s_awvalid		output	AXI slave write address channel valid
axi_s_awready		input	AXI slave write address channel ready
axi_s_wdata	[W_DWIDTH-1:0]	output	AXI slave write data channel data
axi_s_wstrb	[(W_DWIDTH/8)-1:0]	output	AXI slave write data channel strobe
axi_s_wlast		output	AXI slave write data channel last
axi_s_wuser	[W_USRWIDTH-1:0]	output	AXI slave write data channel user
axi_s_wvalid		output	AXI slave write data channel valid
axi_s_wready		input	AXI slave write data channel ready
axi_s_rid	[R_IDWIDTH-1:0]	output	AXI slave read response channel Id
axi_s_rdata	[R_DWIDTH-1:0]	output	AXI slave read response channel data
axi_s_rlast		output	AXI slave read response channel last
axi_s_rresp	[1:0]	output	AXI slave read response channel response
axi_s_ruser	[R_USRWIDTH-1:0]	output	AXI slave read response channel user
axi_s_rvalid		output	AXI slave read response channel valid
axi_s_rready		input	AXI slave read response channel ready
axi_s_bid	[B_IDWIDTH-1:0]	output	AXI slave write response channel Id
axi_s_bresp	[1:0]	output	AXI slave write response channel response
axi_s_buser	[B_USRWIDTH-1:0]	output	AXI slave write response channel user
axi_s_bvalid		output	AXI slave write response channel valid
axi_s_bready		input	AXI slave write response channel ready
cr_init_delay	[63:0]	input	Number of clock cycles to delay credit init
ubump	[NBMP-1:0]	inout	IO channel micro-bumps **(Begin AIB ports)**
ubump_aux	[1:0]	inout	AUX channel micro-bumps
tx_clk		input	AIB clock input

Port Name	Bit Range	Direction	Description
rx_clk		output	AIB clock output generated from micro-bumps
conf_done		input	Configuration complete from all AIB bridges in system
config_done		output	Configuration complete for this AIB bridge
device_detect_ovrd		input	DFT: Overrides device detect function
por_ovrd		input	DFT: POR override for water test (0=override)
pclk		input	APB clock
presetn		input	APB reset
paddr	[11:0]	input	APB address
pwrite		input	APB direction
psel		input	APB select
penable		input	APB enable
pwdata	[31:0]	input	APB write data
prdata	[31:0]	output	APB read data
pready		output	APB bus ready
tdi		input	Test data in (JTAG)
tck		input	Test clock
tms		input	Test mode select
tdo		output	Test data out
trstn_or_por_rstn		input	JTAG resetn or por_rstn
rx_error		output	Receive parity error
rx_error_data	[DWIDTH-1:0]	output	Receive parity error data
tx_rlen_error		output	Transmit read length error
tx_rlen_error_sz	[2:0]	output	Transmit read length error AXI size
tx_rlen_error_len	[7:0]	output	Transmit read length error AXI length
tx_wlen_error		output	Transmit write length error
tx_wlen_error_sz	[2:0]	output	Transmit write length error AXI size
tx_wlen_error_len	[7:0]	output	Transmit write length error AXI length
dig_test_sel		input	Enable digital test bus
dig_test_bus	[7:0]	output	Output DFT: Debug signals from AIB IO

4.3 Parameters Used in Master and Slave Integrated Blocks

Parameter Name	**Default Value**	**Description**
DWIDTH	80	AIB interface width may be 40 or 80
DISABLE_SDR	0	Don't operate in 40-bit mode when DWIDTH = 80 & ddr_en = 0
ADDR_DID_WIDTH	2	Width of address field used for DID. Max is 8.
CFG_ADDR_WIDTH	8	Width of address field used for config space. Min is 0. Max is 31
CFG_ADDR	255	Value that indicates config space address region
AR_AWIDTH	34	Read addr width, min is 8, max is 34 + ADDR_DID_WIDTH
AR_IDWIDTH	8	Read addr ID width maximum is 8
AR_USRWIDTH	8	Read addr user width maximum is 8
AW_AWIDTH	34	Write addr width, min is 8, max is 34 + ADDR_DID_WIDTH
AW_IDWIDTH	8	Write addr ID width maximum is 8
AW_USRWIDTH	8	Write addr user width maximum is 8
W_DWIDTH	128	Write data width must be 32, 64 or 128
W_USRWIDTH	8	Write data user width maximum is 8
R_IDWIDTH	8	Read data ID width maximum is 8
R_DWIDTH	128	Read data width must be 32, 64 or 128
R_USRWIDTH	5	Read data user width maximum is 5
B_IDWIDTH	8	Write response ID width maximum is 8
B_USRWIDTH	5	Write response user width maximum is 5
NUM_AXI_BURSTS	2	Num AXI bursts in transmit FIFO. Max is 16
NUM_CPI_FLITS	128	Num CPI flits in receive FIFO. Min is 128. Max is 256.
MAX_XFR_SIZE	256	Maximum AXI data transfer is 256 bytes.
SIZEFIT	0	Write data size is always equal to W_DWIDTH
CR_RTN_THRESH	32	Credit return threshold
AXI_BURST_WAIT	1	Start after all AXI burst transfers received
CPI_BURST_WAIT	0	Start after all CPI flits received
NDAT	80	Number of Sync Data uBumps in chan
NBMP	90	Total number of bumps
HNBMP	45	Half the # of uBumps in chan
FBMP	88	# of functional/logical uBumps (wo spares)
DLYW	10	Manual mode DLL adjust bit width

4.4 APG Register Write Sequence for Initialization

The sequence of writes required to initialize the master and slave DUT is described in the TLRB AIB PHY DESIGN PRELIMINARY SPECIFICATION spec, section 14.5. The overall scheme is:

a. Write POR register—poll for completion of POR
b. Write TX drive strength register
c. Write TX configuration/enable register
d. Write RX configuration/enable register
e. Write RX delay adjust
f. Write repair address
g. Write revision
h. Write config_done—poll for conf_done (all chiplets configured)
i. Release CPI bridge reset

4.5 JTAG Tests

The JTAG tests exercise the JTAG TAP controller and associated scan circuitry. These tests check the public BYPASS instruction and 15 private instructions described in the TLRB AIB PHY DESIGN PRELIMINARY SPECIFICATION, section 14.2. The JTAG tests also shift data through the 1080 boundary scan registers. There are 12 boundary scan registers associated with each of the 90 bumps.

5. Results

5.1 Hardware Issues Found/Fixed

a. The DUT was not throttling read data to master based on the master RREADY signal (CPI bridge). This was fixed with an RTL update from Micron.
b. The DUT was not keeping all READY signals low during reset (CPI bridge) (found by VIP assertions). This was fixed with an RTL update from Micron.
c. 256-byte packets were stalling in the CPI because there weren't enough credits in the 80-bit DWIDTH, SDR mode. This was fixed with an RTL update from Micron.

5.2 Test Environment Issues Found/Fixed

a. No code coverage or seed modifications were provided by the Vivado simulator. These issues were resolved by passing control from Vivado to the Mentor Questasim simulator. Questasim provided both code coverage and seed modification.
b. Parameters were not passing from the master VIP through the DUT to the slave VIP. These issues resolved after removing the unused trunks from the DUT interface.
c. The AXI VIP checkers were not properly checking responses for overlapping write/read frames. This issue was resolved by spitting the actual and expected transaction queues into two queues each, actual and expected queues for writes and actual and expected queues for reads.

5.3 Latency Measurement Results

Latency was measured as described in section 3.5 above. These are the measured latencies over 200 randomized, overlapping write/read transactions in system clock cycles:

Test Type	Write latency (min<avg<max) Clock Cycles	Read Latency (min<avg< max) Clock Cycles
DDR/80 bits	$8 \leq 31.6 \leq 71$	$10 \leq 30.1 \leq 92$
DDR/40 bits	$9 \leq 43.3 \leq 117$	$11 \leq 41.0 \leq 128$
SDR/80 bits	$9 \leq 47.5 \leq 111$	$11 \leq 39.6 \leq 128$
SDR/40 bits	$9 \leq 47.5 \leq 111$	$11 \leq 39.6 \leq 128$

These numbers show that during randomized, overlapping write/read activity, the latency value measured vary widely. Further analysis showed that if frames are widely spaced, the latency values approach the theoretical minimums (10 or 11 clock cycles). If frames are tightly packed, the latency values tend to increase. There is a trade-off between latency and system throughput. To ensure low latency on a given frame, the upstream process should make sure the frame of interest is appropriately isolated from potentially competing frames.

5.4 Coverage Results

Coverage results from the merging of 48, 200-transaction tests over DDR, DWIDTH and seed are summarized in the table, below. The cover groups and cover directives were written to provide additional monitoring of APB bus activity. Coverage classes are described in section 3.6.2.

Assertion	Statement	Branch	FSM State	FSM Trans	Cover Groups	Cover Directives
100.0%	100.0%	100%	100.0%	100%	100%	100%

After exclusions (3.6.3), the coverage of all metrics followed was 100%.