Blue Cheetah Analog Design
AIB Gen2 PHY Functional Specification

Version 1.23

February 2, 2024

Contents

Table of Figures

Table of Tables

1.Introduction

This document is the functional design specification for Blue Cheetah Analog Design's (BCA's) Advanced Interface Bus 2.0 (AIB2.0) PHY. It describes the PHY's high-level architecture and implementation details, including all run-time programming options available in its software accessible control and status registers (CSRs) and all procedures needed to reset, initialize and configure the PHY for functional operation.

1.1. PHY Overview

The PHY is delivered as a single, hard macro implemented in Intel 16 process technology. It is compliant to the new, AIB 2.0 specification supporting 24 channels, each with 40 transmit lanes and independent 40 receive lanes (also referred to as 80 IOs balanced), each lane operating at data rates of up to 4 Gbps.

The PHY is designed as a dual-mode interface, able to be configured as either a leader or a follower. It includes all the required and optional features of an AIB Plus configuration. The interface to its CSRs is Intel's Avalon Memory-Mapped Interface (AVMM). It implements a medium-density micro bump pitch (45µm) and the universal AIB channel height of 312.48um, leading to an AIB column height of 8025.57.

1.2. Feature List

Item	Feature	Description
1	AIB standard	AIB 2.0 (Gen1/Gen2)
2	AIB configuration	AIB Plus
3	Interface type	Dual-Mode
4	Number of channels	24
5	Maximum data IOs per channel	40/40 (TX/RX)
6	Maximum data rate per IO	4 Gbps
7	FIFO modes	1:1, 2:1, 4:1 (plus register mode)
8	Optional DCC	Included
9	Optional DLL	Included
10	CSR interface	AVMM
11	AUX channel connections	C4 bumps external to PHY
12	Micro bump pitch	45µm (medium-density)
13	IP delivery	Single, hard macro
14	Macro height	8025.57
15	Macro width	866.16
16	Process technology	Intel 16 (with enhanced metal stack)

Table 1: List of PHY Features

1.3. Differences Between Other Implementations

Item	Feature	Description
1	Clocking	The TX PHY uses a low-jitter clock generated by the system PLL and corrected by a calibrated DCC/DLL circuit to provide a low-jitter, low-DCD clock to the PHY and SoC. The RX PHY skews the forwarded clock using a calibrated DLL to provide robust timing margins at the receiver.
2	Location of redundancy	The multiplexing of signals to different micro bumps to work around package assembly defects is mostly done in the digital domain.

Table 2: Differences Between Other Implementations

1.4. Additional Information

Additional information regarding the AIB can be found in Intel's "*Advanced Interface Bus (AIB) Specification"* (1) (referred to herein as "***AIB Spec***") and their "*Advanced Interface Bus (AIB) Usage Note"* (2) ("***AIB Usage Note***"). We assume you have read and are familiar with everything in the ***AIB Spec***. The AVMM bus is similar to ARM's APB in both scope and complexity. Its protocol is defined in Intel's "Avalon Interface Specifications" (3) ("***AVMM Spec***"). JTAG boundary scan functionality is defined in the IEEE 1149.1-2001 standard "IEEE Standard Test Access Port and Boundary Scan Architecture" (4) ("***JTAG Std***").

2. Functional Description

2.1. AIB PHY

A block diagram of the PHY is shown in **Figure 1** below.

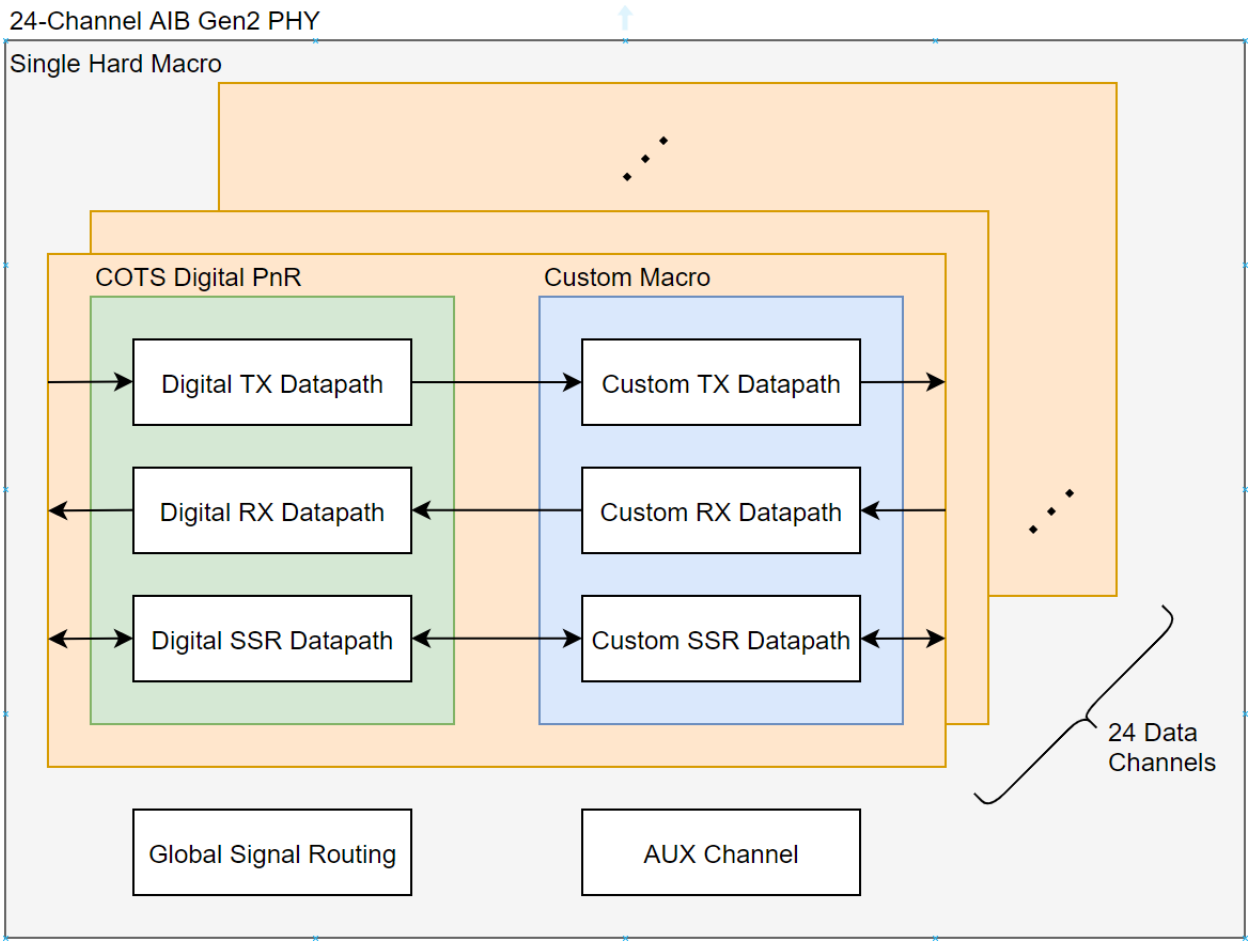

Figure 1: PHY Block Diagram

The PHY is delivered as a hard macro – a single, physical object that BCA's customers instantiate in their SoC netlist. As shown in the **Figure 1** there are 24 data channels, one AUX channel and some top-level, global routing to get from the ports of the PHY to the various channels and back. The orientation of the data busses implies the MAC is to the left of the above figure, towards the core of the SoC, and the external, AIB IO interface and its EMIB connection is to the right.

The ***AIB Spec*** (1) divides any AIB Plus data channel into two pieces: an AIB Adapter layer and an AIB IO layer.

2.1.1 24 Data Channels

The AIB PHY consists of a single AIB column. This column instantiates the maximum number of data channels allowed, 24. Each channel is identical (from a PnR perspective) to all the others. The AIB architecture is highly regular allowing specific blocks to be designed and implemented once and instantiated many times. This hierarchical design style is used often throughout the PHY.

2.1.2 One AUX Channel

The AIB column also instantiates the one auxiliary (AUX) channel, as well. Unlike the data channels, the AUX channel does not use any micro bumps. The two signals present in an AUX channel, *device detect* and *power_on_reset*, enter/leave the chiplet on standard C4 bumps. These bumps are located external to the PHY. All remaining structures and logic associated with these signals are located within the PHY, including their IO buffers and ESD protection.

2.1.3 AIB Data Channel

Each of the 24 data channels operate independently. They have their own AVMM slave interface, CSR address space and CSR registers thus, each can be configured uniquely. The *CONF_DONE* signal is used to exit the configuration phase across all channels of a given AIB interface at the same time, but this is also true for all AIB channels of a given module independent of how many chiplets are in the module and how many AIB interfaces are present in each chiplet. Calibration and the AIB link coming up occur independently for each channel. Thus, the channel should be viewed as the fundamental design entity of an AIB interface

A data channel consists of the following main components which are explained in subsequent sections:

Item	Component
1	Transmit (TX) Datapath
2	Receive (RX) Datapath
3	SSR TX Datapath
4	SSR RX Datapath
5	Reset and Initialization
6	AVMM Interface
7	JTAG interface

Table 3: Data Channel Components

2.1.4 Signal Direction Naming Convention

Since there are two PHYs in any AIB link, it is likely that one of these will be oriented in the opposite direction from what would be considered the normal data flow. This is especially true in IO links where there is usually a clear data direction, at least from the perspective of the overall system. This is shown below in **Figure 2** where a system-on-a-chip module is composed of two chiplets. Chiplet #1 is the main SoC device with AIB

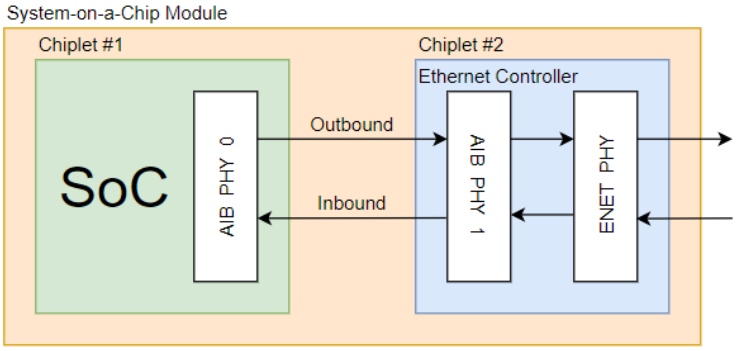

Figure 2: Signal Direction Naming Convention

PHY instance 0. Chiplet #2 is an IO device, in this example an Ethernet controller, with AIB PHY instance 1. From the perspective of the SoC, the top, outbound (also referred to as egress) path is the transmit data path and the bottom, inbound (ingress) path is the receive data path. These directions match those normally assigned to PHY 0, where the top path exits from its TX drivers and the bottom path enters its RX receivers. However, this is not true for PHY 1. The top, SoC TX path enters PHY 1's RX receivers and the bottom, SoC RX path exits PHY 1's TX drivers. Valid arguments could be made to call the top portion of PHY 1 TX or RX. Similarly for the bottom portion of PHY 1.

In this document, we will always refer to data directions with respect to the PHY itself and ignore where the PHY might sit in the larger system and the role it plays there. Thus, the top data path through PHY 0 and the bottom data path through PHY 1 are both called their TX data paths. Similarly, the bottom data path through PHY 0 and the top data path through PHY 1 are both RX data paths.

2.1.5 Data Paths and Clock Trees

In transmit path, the transmit clock, *m_ns_fwd_clk*, which is provided by SOC, is connected to AIB PHY IO analog logic where the clock phase can be calibrated by DCC/DLL block. Then, the adjusted clock is divided in Tx adapter clock divider and sent back to SoC where, in a possible implementation, it can be used for launching the FIFO data input (*data_in_f[319:0]*). Besides the divided clock *(ns_fwd_clk_div)*, AIB PHY provides a non-divided clock *(ns_fwd_clk)* which can be used on SoC for register mode to launch data input (*data_in[79:0]*). The main idea of providing a non divided clock is to avoid the introduction of an extra propagation delay in the clock tree caused by the clock divider logic.

In receiver path, DLL block generates the receive clock, *Rx clk adapt,* based on the received clock from the micro bump. The clock is used for writing the data received from analog IOs into Rx adapter FIFO. DLL block also generates *Rx clk soc* which is divided *(m_fs_fwd_clk)* in clock divider block and sent to SoC where it can be used for capturing the data in a possible implementation. Similarly, to transmit path, AIB PHY provides a non-divided clock *(fs_fwd_clk)* for register mode to SoC where the clock is not delayed by the clock divider logic. In an implementation option, this clock can be used for sampling data output (*data_out[79:0]*) provided by Rx adapter. The figure below shows AIB PHY data paths and AIB PHY clock trees.

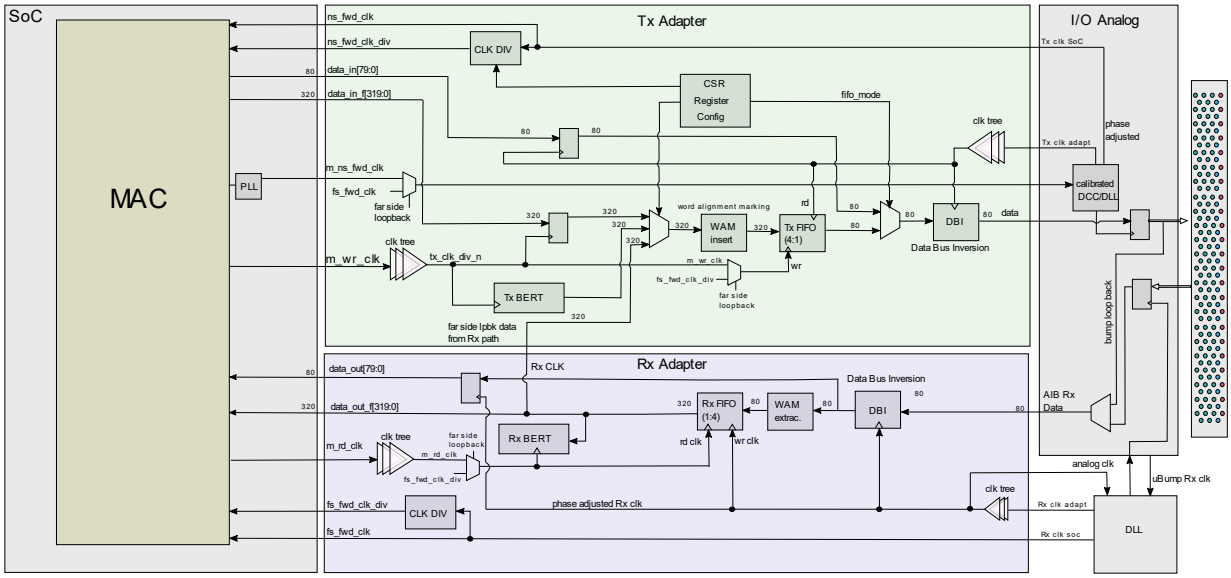

Figure 3: Data Paths and Clock Trees

2.1.6 TX Datapath

The TX data path consists of all the logic required to drive the high-speed AIB PHY outputs including: *tx[39:0], m_ns_fwd_clk* and *m_ns_rcv_clk*.

2.1.6.1 TX Clocking

Internally, the TX PHY will use a low-jitter, low-DCD clock derived from the system PLL (*m_ns_fwd_clk)* and passed through the calibrated DCC/DLL circuit. After being divided down to the appropriate frequency for the data rate, the PHY will drive the divided clock to the SoC as *ns_fwd_clk_div*. The SoC can decide whether to observe and / or utilize the *ns_fwd_clk_div* signal from all, some subset or only one of the AIB PHY channels. In other words, the SoC has full flexibility in the degree of channel bonding down to fully independent channels. A version of this clock, presumably after some amount of buffering

and distribution, can be driven back to the PHY as *m_wr_clk*. Data from the SoC will be launched in the *m_wr_clk* domain and captured in the PHY by *m_wr_clk*. This scheme is illustrated in **Figure 4**.

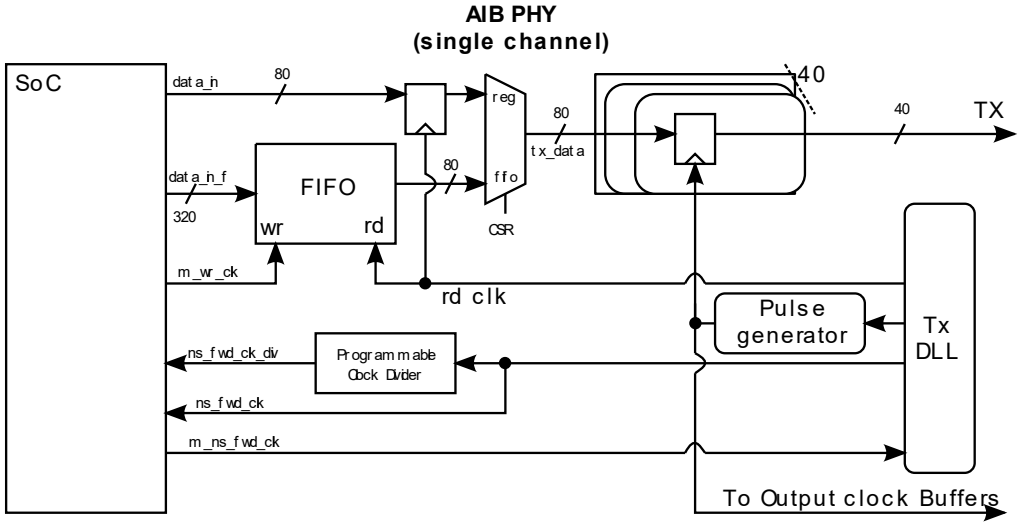

Figure 4: TX Clock Domains

There are slight differences between the implementation of the clock domains in the TX PHY and SoC as defined in the ***AIB Spec*** (1)***.*** Specifically, the FIFO with programmable read phase captures the incoming data using *rd clk* (see figure above) in FIFO modes and an additional divided clock (*ns_fwd_clk_div*) is driven to the SoC and can be used as the root of m_wr_clk in an implementation option. In register mode, SoC can use *ns_fwd_clk* clock (*m_ns_fwd_clk*) instead of *ns_fwd_clk_div* clock if required. There is no change to the functionality or implementation of m_ns_rcv_clk or m_fs_rcv_clk. The following table summarizes these changes for the TX.

AIB Spec.	BCA Impl.	Difference
m_wr_clk	*m_wr_clk*	None
m_ns_fwd_clk	*m_ns_fwd_clk*	None
N/A	*ns_fwd_clk_div*	Divided version of *m_ns_fwd_clk* driven to SoC.
N/A	*ns_fwd_clk*	Adjusted version of m_ns_fwd_clk driven to SoC.

Table 4: Correspondence Between TX Clocks

2.1.6.2 TX Digital

The fundamental components of the AIB Adapter layer are the data retiming register and the phase compensation FIFO. Both are implemented in the digital portion of the TX data path. This logic is shown below in **Figure 5**. The left side of the figure connects to the local AIB MAC layer. The right side to the Custom portion of the TX data path. The input on the bottom side is the far side loopback path from the RX data path.

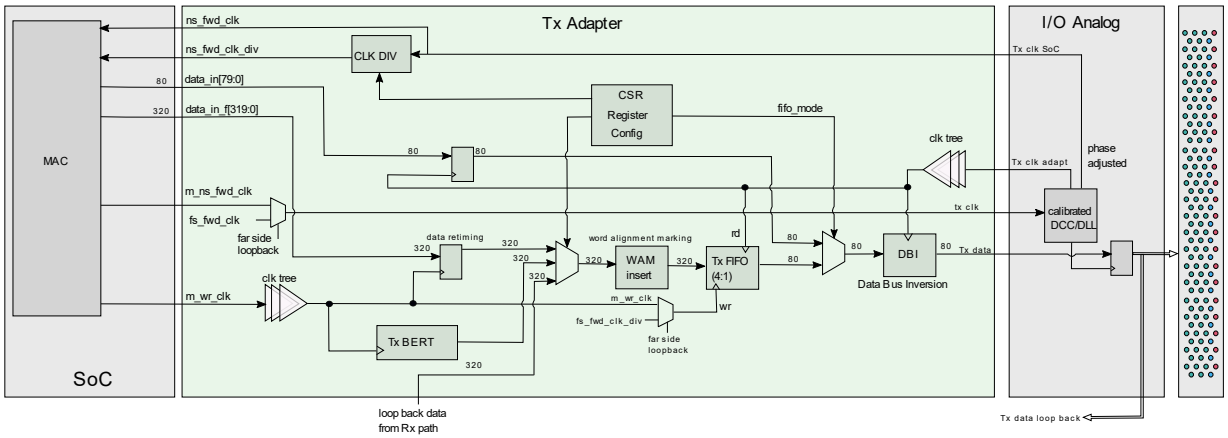

Figure 5: Digital Portion of TX Datapath

There are four sources of TX data. The first two sources are from the MAC; *data_in[79:0]*, intended for the data retiming register, and *data_in_f[319:0]*, intended for the phase compensation FIFO. The register operates down to half the rate of the AIB interface (when operating the AIB IO in DDR mode) and thus needs twice the number of data bits (80) for each lane (40). The FIFO operates down to one-eighth the rate (when operating the FIFO in 4:1 mode and the IO in DDR mode) needing eight times the number of data bits (320). Both register and FIFO mode support data bus inversion (DBI) and may need those symbols inserted. The serializer in the Custom portion of the TX data path implements the functionality of both register mode and FIFO mode. Thus, it is permissible to combine these two data paths together.

The third source is from an internal test pattern generator. This is similar to the optional test pattern extension described in the ***AIB Spec*** (1) and is documented in **Bit Error Rate Tester (BERT)** section of this specification.

The fourth source is from the RX data path and supports far side loopback. This is documented in **Section 2.3,**

Loopback Testing, of this specification.

The **tx_fifo_mode** field in **txadpcfg_0** register controls TX FIFO operation mode in terms of storage data rate or register mode:

Register Name	Field name	Register Bits	Operation
txadpcfg_0	***tx_fifo_mode***	*22:21*	*00-Transmit FIFO 1X Mode*
			01-Transmit FIFO 2X Mode
			10-Transmit FIFO 4X Mode
			11-Transmit Register Mode

Table 5: tx_fifo_mode field for the selection of the TX data

In SDR mode, the Digital block does not need to replicate valid SDR bits present on the even data bits to the previously unused odd data bits because the custom block running all TX outputs in DDR mode samples only the valid portion of data by disabling the odd clock path (see **Figure 11: Buffx1 TX Serializer**).

In the digital portion, TX FIFO can store multiple words (single, DWs or QWs) depending on the operation mode selected (FIFO 1:1, 2:1 or 4:1 mode). The Phase-compensation FIFOs will serialize the data into multiple, single words(80-bit) when operating in FIFO 2:1 or 4:1 mode.

2.1.6.3 TX Word Alignment Marking

Whenever the AIB link is programmed to operate in FIFO 2:1 or 4:1 mode, the MAC exchanges multiple 80-bit words with the PHY every clock period, in either a 160-bit doublewords (DWs) or a 320-bit quadwords (QWs). The AIB protocol requires that the alignment of the words within the DW or QW presented at the input to the TX PHY be maintained at the output of the RX PHY. Since the transmitter serializes DWs and QWs into words prior to transmission, there is not enough information available in the receiver to properly align the words back into DWs and QWs.

To enable this alignment, an in-band, word alignment marker (WAM) symbol is inserted into every word transmitted. The WAM symbol is a single bit that is 1 for the most significant word in any multi-word entity and 0 for all other words. Instead of being an additional bit added to the AIB interface, the WAM bit overwrites one of the existing data bits. The choice of which data bit to overwrite is configurable, but limited to the 5 bits: 39, 76, 77, 78 and 79. There is a CSR bit to enable WAM insertion (tx_wm_en) and a 5-bit CSR field to select which of the 5 data bits to replace within each word when operating in DDR mode (or 2 data bits in SDR mode).

Register Name	Field name	Register Bit	Operation
txadpcfg_0	***tx_wm_en***	*23*	*0-Disable word marking*
			1-Enable word marking

Table 6: CSR bit for Tx WAM enable

Register Name	Field name	Register Bit Set	Data bit used as WAM	Available for SDR
txadpcfg_0	**tx_marker_bit79**	20	79	No
	tx_marker_bit78	19	78	Yes
	tx_marker_bit77	18	77	No
	tx_marker_bit76	17	76	Yes
	tx_marker_bit39	16	39	No

Table 7: Word Alignment Marker Positions

For example, if operating in 2:1 mode and gen2 with WAM insertion enabled and the WAM bit position is set to the least significant possible bit ([39]), the DW will be written into the FIFO as below:

tx_fifo_word[159:0] = {{*data[159:120]*,1'b1,*data[118:80]*},{*data[79:40]*,1'b0,*data[38:0]*}}

Note, the WAM bits overwrite and replace *data[119,39]* from the MAC. And this choice of WAM position would not be appropriate for SDR mode as only the even data bits are transmitted in SDR mode.

If operating in 4:1 mode (gen2) with WAM enabled for the most significant bit ([79]), the QW present would be:

tx_fifo_word [319:0] =

{{1'b1,*data[318:240]*},{ 1'b0,*data[238:160]*},{1'b0,*data[158:80]*},{1'b0,*data[78:0]*}}

This choice would also not be appropriate for SDR mode.

If operating in 2:1 40-bit word mode (gen1) with WAM enabled, the WAM shall be configured in bit ([39]) only and DW will be written into the FIFO as below:

tx_fifo_word [159:0] =

{{40'bxx_xxxx_xxxx,1'b1,*data[78:40]*},{ 40'bxx_xxxx_xxxx ,1'b0,*data[38:0]*}}

Operation in 4:1 40-bit word mode (gen1) is not supported.

Word alignment marking consumes 1.25% of the link bandwidth in DDR mode and 2.5% in SDR mode. Although the initial alignment in the receiver is random, it remains constant between resets. Thus, once alignment is obtained in the receiver, there is no need for the transmitter to continue sending WAM symbols. The RX PHY communicates to the RX MAC that it has obtained alignment to the incoming data stream via the *m_rx_align_done* signal. If this information can be communicated back to the TX PHY and MAC, the transmitter may stop sending WAM symbols and recover the lost link bandwidth. How this communication occurs is outside the scope of this document and the ***AIB Spec*** (1).

See the section in this document on RX WAM functionality, 2.1.7.8 **RX Word Alignment** Marking, for information on the more difficult task of extracting the WAM symbols and using them to obtain word alignment.

The WAM functionality described above is as documented in the ***AIB Spec*** (1).

2.1.6.4 TX FIFO

The data size of each TX FIFO element and TX FIFO depth depend on operation mode. TX FIFO depths are 20, 10 and 5 for FIFO 1X, FIFO 2X and FIFO 4X mode, respectively, and TX FIFO data input widths are 80-bit, 160-bit and 320-bit for FIFO 1X, FIFO 2X and FIFO 4X mode, respectively. TX FIFO can operate with different clock rate between write clock and read clock depending on FIFO mode configuration. The data width and frequency of operation of its write and its read port are 4:1, 2:1 and 1:1 for FIFO 4x, FIFO 2x and FIFO 1x modes, respectively. It registers both its input write data and its output read data. It has a read latency which can be configured to add an integer number of additional clock periods of latency to increase its tolerance to clock uncertainty (see **Figure 7 TX FIFO read enable Generation**). When a certain set of flops are not being written by the hardware, its respective clocks are gated to save dynamic power. Basically, there is a clock gate for each 40 bits of the FIFO to enable the clock of those flops only when required.

The TX FIFO latency can be programmed by using the *tx_phcomp* field in *txadpcfg_0* register (see **Table 8: tx_phcomp field to program the TX FIFO latency)**.

Register Name	Field name	Register Bit	Functions
txadpcfg_0	**tx_phcomp**	31:28	4'b1011: 11 read cycle delay
			4'b1010: 10 read cycle delay
			4'b1001: 9 read cycle delay
			4'b1000: 8 read cycle delay
			4'b0111: 7 read cycle delay
			4'b0110: 6 read cycle delay
			4'b0101: 5 read cycle delay
			4'b0100: 4 read cycle delay
			4'b0011: 3 read cycle delay
			4'b0010: 2 read cycle delay

Table 8: tx_phcomp field to program the TX FIFO latency

In order to operate properly, *tx_phcomp* field shall be configured in the range described in the table below according to FIFO mode used. These constraints can guarantee that

the FIFO content in write clock domain is stable and with updated value to be read in the read domain clock.

tx_fifo_mode value	FIFO Mode	Allowed tx_phcomp value	Allowed read delay in TX FIFO
00	00-Transmit FIFO 1X Mode	4'b0010-4'b1011	2-11 read cycle delay
01	01-Transmit FIFO 2X Mode	4'b0011-4'b1011	3-11 read cycle delay
10	10-Transmit FIFO 4X Mode	4'b0101- 4'b1011	5-11 read cycle delay
11	11-Transmit Register Mode	N.A.	N.A.

Table 9: Allowed TX FIFO read delay

TX FIFO is 2D register array of flip-flops where their clocks can be enabled or disabled by a group or clock gates depending on the FIFO configuration and the value of write pointer. The **Figure 6: TX FIFO Block Diagram** shows in detail the TX FIFO implementation.

The tables below show the enabled and disabled clocks in TX FIFO according to FIFO mode. The variable “i” represents TX FIFO element and the index of clock enable refers to the bits of a FIFO element, for instance, tff_clk_39_0_en[3] refers to bits from 39 to 0 of TX FIFO element 3. For FIFO 1x and FIFO 2x modes, the enabled clock also depends on the clock cycle FIFO is operating in write side.

Clock signals	Clock cycle 0	Clock cycle 1	Clock cycle 2	Clock cycle 3
tff_clk_39_0_en[i]	**enabled**	**disabled**	**disabled**	**disabled**
tff_clk_79_40_en[i]	**enabled**	**disabled**	**disabled**	**disabled**
tff_clk_119_80_en[i]	**disabled**	**enabled**	**disabled**	**disabled**
tff_clk_159_120_en[i]	**disabled**	**enabled**	**disabled**	**disabled**
tff_clk_199_160_en[i]	**disabled**	**disabled**	**enabled**	**disabled**
tff_clk_239_200_en[i]	**disabled**	**disabled**	**enabled**	**disabled**
tff_clk_279_240_en[i]	**disabled**	**disabled**	**disabled**	**enabled**
tff_clk_319_280_en[i]	**disabled**	**disabled**	**disabled**	**enabled**

Table 10: Tx FIFO clock gating for FIFO 1x mode (gen2 mode).

Clock signals	Clock cycle 0	Clock cycle 1
tff_clk_39_0_en[i]	**enabled**	**disabled**
tff_clk_79_40_en[i]	**enabled**	**disabled**
tff_clk_119_80_en[i]	**enabled**	**disabled**
tff_clk_159_120_en[i]	**enabled**	**disabled**
tff_clk_199_160_en[i]	**disabled**	**enabled**
tff_clk_239_200_en[i]	**disabled**	**enabled**
tff_clk_279_240_en[i]	**disabled**	**enabled**
tff_clk_319_280_en[i]	**disabled**	**enabled**

Table 11: Tx FIFO clock gating for FIFO 2x mode (gen2 mode).

Clock signals	Enabled/Disabled
tff_clk_39_0_en[i]	**enabled**
tff_clk_79_40_en[i]	**enabled**
tff_clk_119_80_en[i]	**enabled**
tff_clk_159_120_en[i]	**enabled**
tff_clk_199_160_en[i]	**enabled**
tff_clk_239_200_en[i]	**enabled**
tff_clk_279_240_en[i]	**enabled**
tff_clk_319_280_en[i]	**enabled**

Table 12: Tx FIFO clock gating for FIFO 4x mode.

For gen1 mode, tff_clk_79_40_en[i], tff_clk_159_120_en[i], tff_clk_239_200_en[i] and tff_clk_319_280_en[i] are disabled during all the operation.

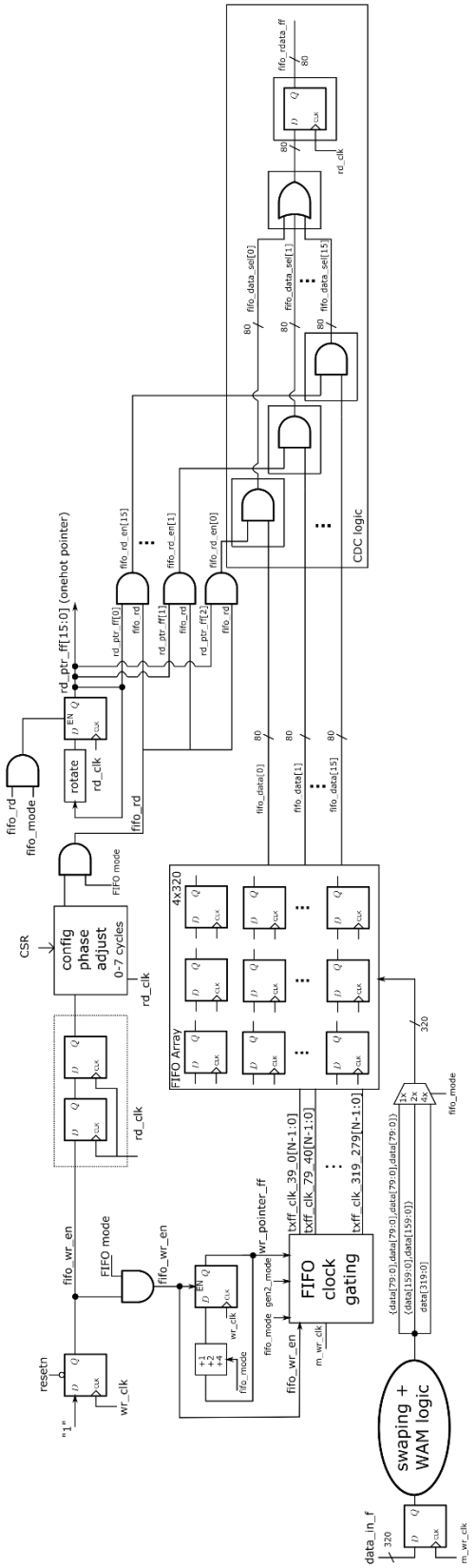

Figure 6: TX FIFO Block Diagram

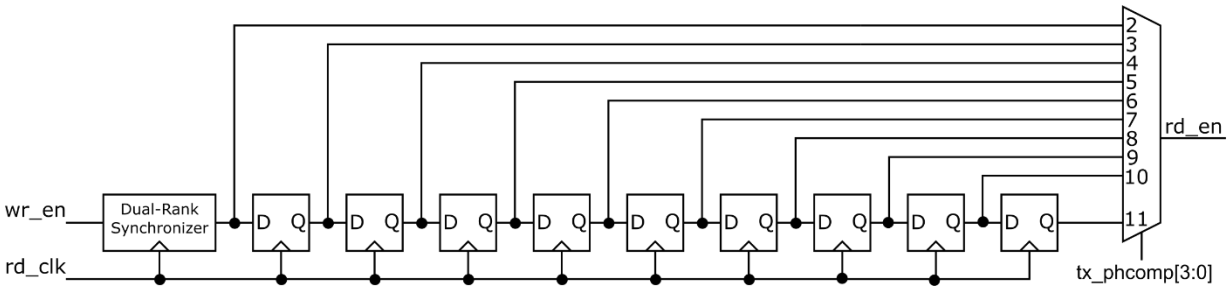

Figure 7 TX FIFO read enable Generation

2.1.6.5 TX FIFO clock drift

TX FIFO supports drift between read clock and write clock. The maximum clock drift depends on FIFO operation mode. **Table 13: TX FIFO maximum clock drift** shows the maximum clock Drift supported by Tx FIFO:

Mode	FIFO depth (w.r.t write side)	Maximum clock drift supported (ns_fwd_clk period)
1x	20	8
2x	10	8
4x	5	6

Table 13: TX FIFO maximum clock drift

The figures below show the effect of clock drift for 1x, 2x and 4x TX FIFO modes.

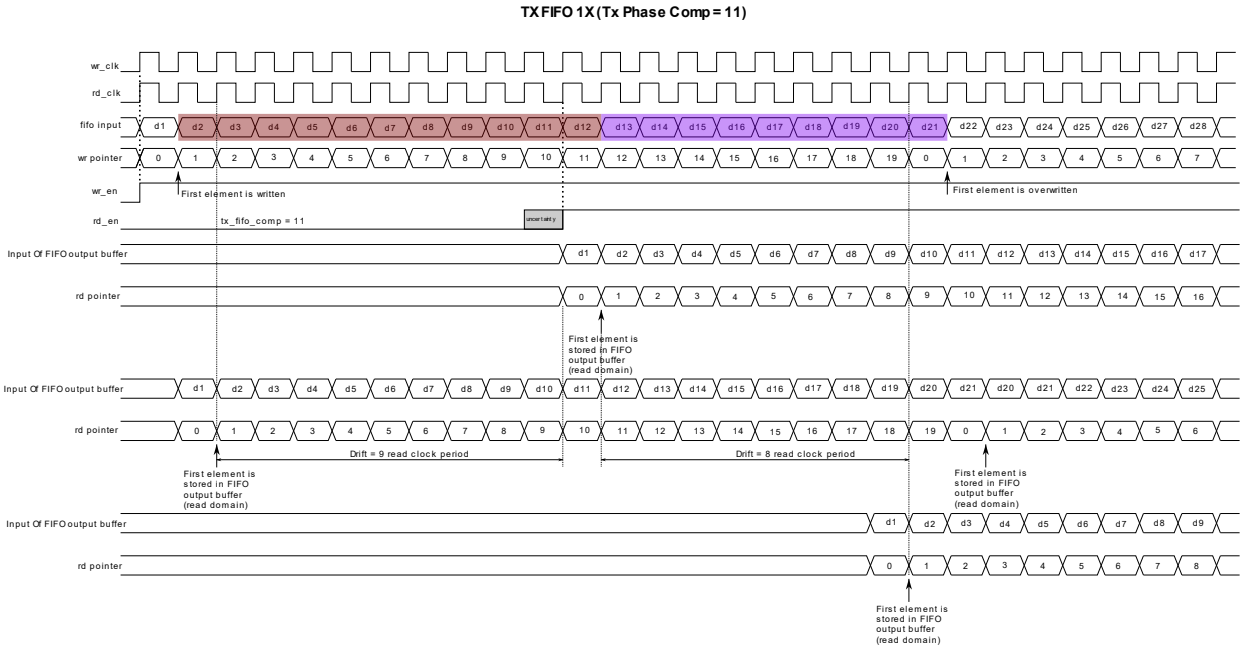

Figure 8: Clock drift in 1x TX FIFO mode

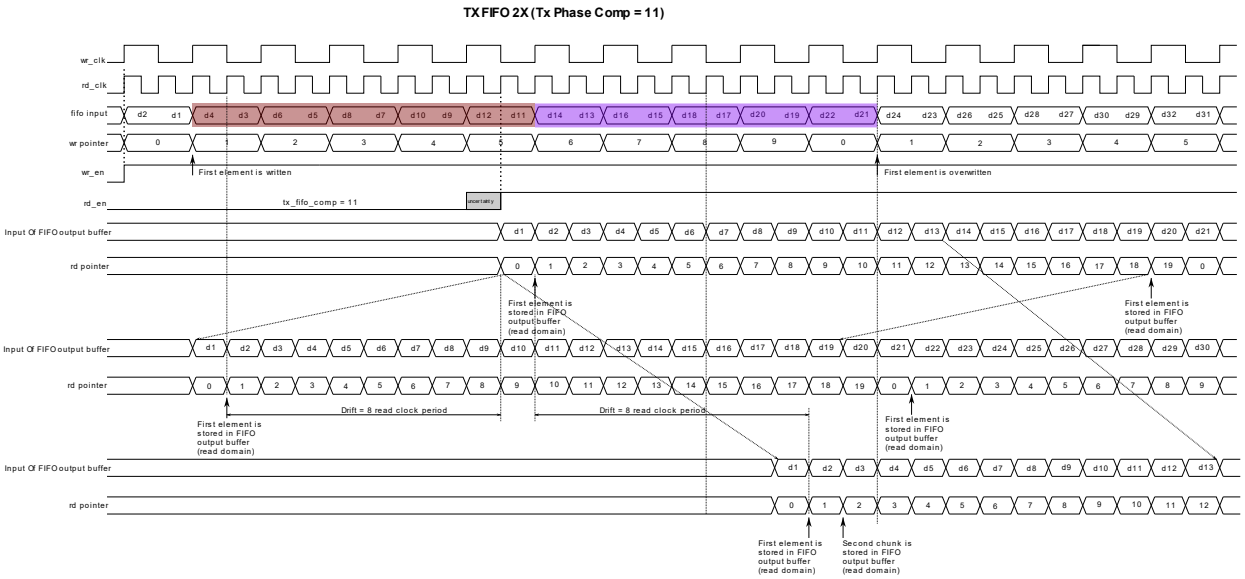

Figure 9: Clock drift in 2x TX FIFO mode

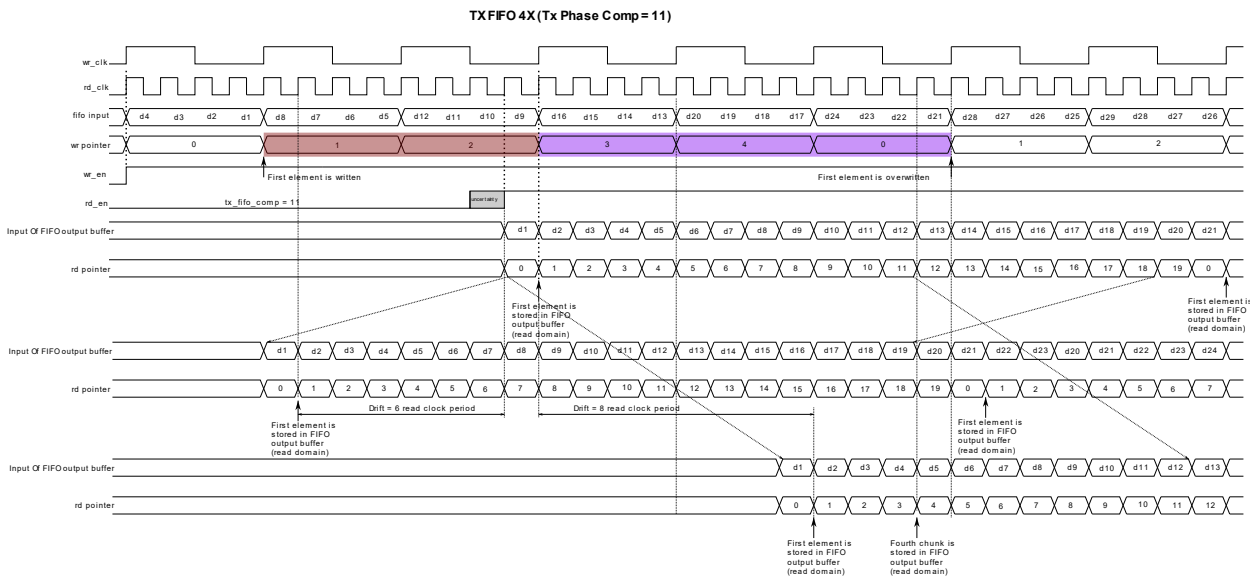

Figure 10: Clock drift in 4x TX FIFO mode.

2.1.6.6 TX Data Bus Inversion

Data bus inversion (DBI) is a new feature added to the Gen2 version of AIB. As with other DBI implementations, it is meant to reduce power. Since AIB is an unterminated bus, the only IO power dissipated is AC power due to the switching activity of the individual bits of the bus. Thus, power is saved when the number of bits which switch each cycle is reduced. This has the added benefit of reducing simultaneous switching noise (SSN) in the power delivery network (PDN). Depending on which version results in less switching activity, either the true version of the bus or the complement version of the entire bus (all bits of the bus are inverted) along with a symbol (the DBI bit) to indicate whether the bus is complemented (DBI=1) or not (DBI=0). Using this feature, the number of bits which switch each cycle can be kept less than or equal to half the width of the bus.

For backwards compatibility, DBI may be used only in Gen2 mode (implying only with DDR transfers). To improve performance, the 40-bit AIB interface is broken into two DBI groups – one for the lower half of the bus (*tx[19:0]*) and one for the upper half (*tx[39:20]*). Like the WAM bit, the DBI bits overwrite and replace existing data bits. Unlike the WAM bit, the location of the DBI bits is fixed to be the most significant in their group:

$$tx[39:0] = \{\{dbi[1],data[38:20]\},\{dbi[0],data[18:0]\}\}$$

Due to the way serial data is packed into words, DWs and QWs – the last serialization between SDR and DDR data occurs between even and odd bits of the data words, while the other levels of serialization occur between the multiple words within a DW or QW – the determination of what the sequential bits on a given lane is not obvious. The sequential bits of the lower half of the data bus are shown in the following tables (the

upper half is computed identically). *tx[n][m]* indicates bit *m* of the *tx* bus at unit interval (UI) *n*.

For register and FIFO 1:1 mode, the UI duration of the DDR data (*tx* and *dbi*) is half that of the SDR data (*data*). Thus, there are twice as many DDR UIs as SDR UIs, requiring two DDR UIs to document the sequence:

tx[2n][19:0] = {*dbi[2n][0],data[n][36,34,32,30,28,26,24,22,20,18,16,14,12,10,8,6,4,2,0]*}

tx[2n+1][19:0] = {*dbi[2n+1][0],data[n][37,35,33,31,29,27,25,23,21,19,17,15,13,11,9,7,5,3,1]*}

Table 14: DBI Sequential Bits in FIFO 1:1 Mode

For FIFO 2:1 mode, the UI ratio is four to one, requiring four UIs:

tx[4n][19:0] = {*dbi[4n][0],data[n][36,34,32,30,28,26,24,22,20,18,16,14,12,10,8,6,4,2,0]*}

tx[4n+1][19:0] = {*dbi[4n+1][0],data[n][37,35,33,31,29,27,25,23,21,19,17,15,13,11,9,7,5,3,1]*}

tx[4n+2][19:0] = {*dbi[4n+2][0],data[n][116,114,112,110,108,106,104,102,100,98,96,94,92,90,88,86,84,82,80]*}

tx[4n+3][19:0] = {*dbi[4n+3][0],data[n][117,115,113,111,109,107,105,103,101,99,97,95,93,91,89,87,85,83,81]*}

Table 15: DBI Sequential Bits in FIFO 2:1 Mode

For FIFO 4:1 mode, the UI ratio is eight to one, requiring eight UIs:

tx[8n][19:0] = {*dbi[8n][0],data[n][36,34,32,30,28,26,24,22,20,18,16,14,12,10,8,6,4,2,0]*}

tx[8n+1][19:0] = {*dbi[8n+1][0],data[n][37,35,33,31,29,27,25,23,21,19,17,15,13,11,9,7,5,3,1]*}

tx[8n+2][19:0] = {*dbi[8n+2][0],data[n][116,114,112,110,108,106,104,102,100,98,96,94,92,90,88,86,84,82,80]*}

tx[8n+3][19:0] = {*dbi[8n+3][0],data[n][117,115,113,111,109,107,105,103,101,99,97,95,93,91,89,87,85,83,81]*}

tx[8n+4][19:0] = {*dbi[8n+4][0],data[n][196,194,192,190,188,186,184,182,180,178,176,174,172,170,168,166,164,162,160]*}

tx[8n+5][19:0] = {*dbi[8n+5][0],data[n][197,195,193,191,189,187,185,183,181,179,177,175,173,171,169,167,165,163,161]*}

tx[8n+6][19:0] = {*dbi[8n+6][0],data[n][276,274,272,270,268,266,264,262,260,258,256,254,252,250,248,246,244,242,240]*}

tx[8n+7][19:0] = {*dbi[8n+7][0],data[n][277,275,273,271,269,267,265,263,261,259,257,255,253,251,249,247,245,243,241]*}

Table 16: DBI Sequential Bits in FIFO 4:1 Mode

In deciding whether to send the true or complement data group, the DBI symbol is not included in the computation. The least significant 19 bits of each transmitted 20-bit DBI group are compared to the corresponding bits in the row of the tables above that is to be transmitted next. If the number of bit positions which are different is greater than 9, then the DBI bit is asserted (1) and complement data is transmitted during the next UI, else the DBI bit is deasserted (0) and true data is transmitted.

DBI is computed after WAM insertion.

There is only one CSR bit for TX DBI functionality. This is the enable bit (tx_dbi_en).

Register Name	Field name	Register Bit	Operation

txadpcfg_0	***tx_dbi_en***	*1*	*0-Disable DBI*
			1-Enable DBI

Table 17: CSR bit for Tx DBI enable

2.1.6.7 TX Data Multiplexing

Immediately prior to exiting the Digital portion of the TX data path, there are four levels of 2-to-1 multiplexing to handle:

Item	TX Data Multiplexing
1	Word swapping within the DW in 40-bit, 2:1 FIFO mode
2	Two different word widths (80 versus 40)
3	Redundancy
4	JTAG

Table 14: TX Data Multiplexing

Each of these is discussed in detail below. The multiplexing must be done in the order listed.

2.1.6.8 Swapping Mode

The PHY supports word swapping mode when operating in Gen1 mode for FIFO 2X mode.

All bits, except for the most significant bit of each word, are swapped. The MSB is reserved for the word alignment marker. It is in the correct bit location within the DW and does not need to be swapped. This swapping logic can be implemented with the following 80-bit (78-bit) wide 2-to-1 multiplexer:

```
if (swap)
        data_out[79:0] = {data_in[79],data_in[38:0],data_in[39],data_in[78:40]};
else
        data_out[79:0] = data_in[79:0];
```

2.1.6.9 80-Bit Verses 40-Bit Word Widths

AIB Gen2 interfaces have 40 physical AIB lanes in each direction (40 TX and 40 RX). They also operate in DDR mode leading to an 80-bit word passing through the main PHY data paths each clock period. The maximum FIFO serialization is 4:1, leading to four-word wide quadword (QW) data paths of 320 bits (*data_gen2[319:0]*).

When operating in Gen1 mode, the number of physical lanes used reduces to 20, leading to a 40-bit word. The maximum Gen1 FIFO serialization is 2:1, leading to two-word wide doublewords (DWs) of 80 bits (*data_ gen1[79:0]*).

When in Gen1 mode and operating in register mode or FIFO 1:1 mode, the mapping between the Gen1 data present at the MAC-to-PHY interface and the connections to the AIB IO buffers (which are designed to support Gen2) is straightforward. All that is necessary to remember is that only half of the AIB lanes are used:

```
if (Gen1 && (reg_mode || fifo_1:1_mode))
        data_gen2[79:0] = {40'hXX_XXXX_XXXX,data_gen1[39:0]};
```

When in Gen1 mode and operating in FIFO 2:1 mode, the mapping is only slightly more complicated, requiring that the unused half of the AIB lanes are skipped over:

```
if (Gen1 && fifo_2:1_mode)
        data_gen2[159:0] = {40'hXX_XXXX_XXXX,data_gen1[79:40],40'hXX_XXXX_XXXX,data_gen1[39:0]};
```

The other FIFO mode, 4:1, is not possible in Gen1 and does not need to be considered here.

Thus, this word width conversion logic can be implemented in the following 160-bit (40-bit) wide 2-to-1 multiplexer:

```
if (Gen1 && fifo_2:1_mode)
        data_out[159:0] = {40'hXX_XXXX_XXXX,data_in[79:40],40'hXX_XXXX_XXXX,data_in[39:0]};
else
        data_out[159:0] = data_in[159:0];
```

2.1.6.10 Redundancy

Redundancy is described in more detail in **Section 2.4, Redundancy.** Here it is mentioned information to explain the final 2-to-1 multiplexer only.

Active redundancy is a way to improve yield by providing two spare micro bumps in each channel and a way to route signals away from faulty micro bumps to these two spares. Each channel is composed of 102 micro bumps – 100 functional, used micro bumps and the two spare, normally unused micro bumps. The micro bumps are named *aib[101:0]*. The two spares are located in the middle of the micro bump array at *aib[51:50]*.

Rerouting signals around a fault is implemented with four redundancy shift chains per channel, two which shift up from lower numbered micro bumps into the spares when the single fault is located within *aib[49:0]* and two which shift down from higher numbered micro bumps into the spares when the single fault is located within *aib[101:52]*. Even though there are two spare micro bumps, there is support for fixing only one fault.

The only noticeable effect of the two chains per direction is that signals jump two micro bumps towards the spares instead of the expected one. For example, if the faulty micro bump is *aib[56]*, then there are seven micro bumps that are modified:

		Signal Assigned to µBump	
NBR	µBump	Before Fault	After Fault
1	*aib[56]*	*fs_sr_clkb*	Fault
2	*aib[55]*	*fs_sr_data*	Unused
3	*aib[54]*	*fs_sr_load*	*fs_sr_clkb*
4	*aib[53]*	*fs_mac_rdy*	*fs_sr_data*
5	*aib[52]*	*fs_adapter_rstn*	*fs_sr_load*
6	*aib[51]*	Unused (*spare[1]*)	*fs_mac_rdy*
7	*aib[50]*	Unused (*spare[0]*)	*fs_adapter_rstn*

Table 18: Example of Redundancy

In almost all case, redundancy will reroute over the micro bump with the fault as well as the adjacent, next closest to the spares micro bump even though this micro bump is still functional (*e.g., aib[55]* in the example above). It means both spares will be used, leaving a functional micro bump unused in the non-spare micro bump array. The only exceptions to this are when the single fault is adjacent to one of the two spare micro bumps (*i.e.,* when the fault is at *aib[49]* or *aib[52]*).

After reviewing **Table 98: Micro bump Signal Assignment**, it can be concluded that there are three different classes of micro bumps:

Item	Description	µBumps
1	Micro bumps that connect to only one signal. These are the four micro bumps located at the ends of the four redundancy shift chains. Since shifting is unidirectional within each shift chain, it is not possible to shift anything onto the ends of the chains.	*aib[101], aib[100], aib[1], aib[0]*
2	Micro bumps that connect to three signals. These are the two spare micro bumps. These are the only micro bumps connected to two redundancy scan chains, both the one from above and the one from below. The three possible signals are unused and the one above and the one below. When unused, the signal assigned to these micro bumps is a don't care. We will assign this unused case to one of the two signals that are used, so that these connect to only two signals.	*aib[51], aib[50]*
3	Micro bumps that connect to two signals. These are the other 96 micro bumps.	*aib[99:52], aib[49:2]*

Table 19: Three Classes of Micro bumps

Since each micro bump connects to at most two used signals, redundancy can be implemented by another 2-to-1 multiplexer. In contrast to the other multiplexers in this section which have a common select for the entire bus, these multiplexers have a unique

select for each micro bump. When this select is logic-0, the functional, non-redundant signal is selected or, in the case of the spares, the signal associated with the lower numbered signal of the pair of possible signals. When this select is logic-1, the redundant signal is selected or, in the case of the spares, the signal associated with the higher numbered signal of the pair.

```
for (i=0; i<102; i++)
        if (redundancy_sel[i])
                data_out[i] = data_in_redundant[i];
        else
                data_out[i] = data_in_functional[i];
```

There are three registers which contain 102 multiplexer selects (**redund_0, redund_1, redund_2 and redund_3**). There is no enable bit for redundancy. The multiplexer selects together with the programming of the Standby mode of operation for each of the 102 micro bumps all that is required. The programming of these CSR bits is based on the presence or absence of a fault, and this is determined during manufacturing test. The results of these tests must be archived and made available whenever Configuration is performed, and the CSRs are programmed. How this is accomplished is outside the scope of this specification.

Register Name	Register Bit
redund_0	31:0
redund_1	31:0
redund_2	31:0
redund_3	5:0

Table 20: Redundancy Selection bits

The table below shows the values that should be configured in **redund_0**, **redund_1**, **redund_2** and **redund_3** registers according to the bump number with fault.

Bump number with fault	redund_0	redund_1	redund_2	redund_3
0	0xFFFF_FFFF	0x0003_FFFF	0x0000_0000	0x_0000_0000
1	0xFFFF_FFFE	0x0003_FFFF	0x0000_0000	0x_0000_0000
2	0xFFFF_FFFC	0x0003_FFFF	0x0000_0000	0x_0000_0000
3	0xFFFF_FFF8	0x0003_FFFF	0x0000_0000	0x_0000_0000

Bump number with fault	redund_0	redund_1	redund_2	redund_3
4	0xFFFF_FFF0	0x0003_FFFF	0x0000_0000	0x_0000_0000
5	0xFFFF_FFE0	0x0003_FFFF	0x0000_0000	0x_0000_0000
6	0xFFFF_FFC0	0x0003_FFFF	0x0000_0000	0x_0000_0000
7	0xFFFF_FF80	0x0003_FFFF	0x0000_0000	0x_0000_0000
8	0xFFFF_FF00	0x0003_FFFF	0x0000_0000	0x_0000_0000
9	0xFFFF_FE00	0x0003_FFFF	0x0000_0000	0x_0000_0000
10	0xFFFF_FC00	0x0003_FFFF	0x0000_0000	0x_0000_0000
11	0xFFFF_F800	0x0003_FFFF	0x0000_0000	0x_0000_0000
12	0xFFFF_F000	0x0003_FFFF	0x0000_0000	0x_0000_0000
13	0xFFFF_E000	0x0003_FFFF	0x0000_0000	0x_0000_0000
14	0xFFFF_C000	0x0003_FFFF	0x0000_0000	0x_0000_0000
15	0xFFFF_8000	0x0003_FFFF	0x0000_0000	0x_0000_0000
16	0xFFFF_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
17	0xFFFE_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
18	0xFFFC_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
19	0xFFF8_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
20	0xFFF0_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
21	0xFFE0_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
22	0xFFC0_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
23	0xFF80_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
24	0xFF00_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
25	0xFE00_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
26	0xFC00_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
27	0xF800_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
28	0xF000_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
29	0xE000_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
30	0xC000_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
31	0x8000_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000
32	0x0000_0000	0x0003_FFFF	0x0000_0000	0x_0000_0000

Bump number with fault	redund_0	redund_1	redund_2	redund_3
33	0x0000_0000	0x0003_FFFE	0x0000_0000	0x_0000_0000
34	0x0000_0000	0x0003_FFFC	0x0000_0000	0x_0000_0000
35	0x0000_0000	0x0003_FFF8	0x0000_0000	0x_0000_0000
36	0x0000_0000	0x0003_FFF0	0x0000_0000	0x_0000_0000
37	0x0000_0000	0x0003_FFE0	0x0000_0000	0x_0000_0000
38	0x0000_0000	0x0003_FFC0	0x0000_0000	0x_0000_0000
39	0x0000_0000	0x0003_FF80	0x0000_0000	0x_0000_0000
40	0x0000_0000	0x0003_FF00	0x0000_0000	0x_0000_0000
41	0x0000_0000	0x0003_FE00	0x0000_0000	0x_0000_0000
42	0x0000_0000	0x0003_FC00	0x0000_0000	0x_0000_0000
43	0x0000_0000	0x0003_F800	0x0000_0000	0x_0000_0000
44	0x0000_0000	0x0003_F000	0x0000_0000	0x_0000_0000
45	0x0000_0000	0x0003_E000	0x0000_0000	0x_0000_0000
46	0x0000_0000	0x0003_C000	0x0000_0000	0x_0000_0000
47	0x0000_0000	0x0003_8000	0x0000_0000	0x_0000_0000
48	0x0000_0000	0x0003_0000	0x0000_0000	0x_0000_0000
49	0x0000_0000	0x0002_0000	0x0000_0000	0x_0000_0000
50	-	-	-	-
51	-	-	-	-
52	0x0000_0000	0x0010_0000	0x0000_0000	0x_0000_0000
53	0x0000_0000	0x0030_0000	0x0000_0000	0x_0000_0000
54	0x0000_0000	0x0070_0000	0x0000_0000	0x_0000_0000
55	0x0000_0000	0x00F0_0000	0x0000_0000	0x_0000_0000
56	0x0000_0000	0x01F0_0000	0x0000_0000	0x_0000_0000
57	0x0000_0000	0x03F0_0000	0x0000_0000	0x_0000_0000
58	0x0000_0000	0x07F0_0000	0x0000_0000	0x_0000_0000
59	0x0000_0000	0x0FF0_0000	0x0000_0000	0x_0000_0000
60	0x0000_0000	0x1FF0_0000	0x0000_0000	0x_0000_0000
61	0x0000_0000	0x3FF0_0000	0x0000_0000	0x_0000_0000

Bump number with fault	redund_0	redund_1	redund_2	redund_3
62	0x0000_0000	0x7FF0_0000	0x0000_0000	0x_0000_0000
63	0x0000_0000	0xFFF0_0000	0x0000_0000	0x_0000_0000
64	0x0000_0000	0xFFF0_0000	0x0000_0001	0x_0000_0000
65	0x0000_0000	0xFFF0_0000	0x0000_0003	0x_0000_0000
66	0x0000_0000	0xFFF0_0000	0x0000_0007	0x_0000_0000
67	0x0000_0000	0xFFF0_0000	0x0000_000F	0x_0000_0000
68	0x0000_0000	0xFFF0_0000	0x0000_001F	0x_0000_0000
69	0x0000_0000	0xFFF0_0000	0x0000_003F	0x_0000_0000
70	0x0000_0000	0xFFF0_0000	0x0000_00FF	0x_0000_0000
71	0x0000_0000	0xFFF0_0000	0x0000_00FF	0x_0000_0000
72	0x0000_0000	0xFFF0_0000	0x0000_01FF	0x_0000_0000
73	0x0000_0000	0xFFF0_0000	0x0000_03FF	0x_0000_0000
74	0x0000_0000	0xFFF0_0000	0x0000_07FF	0x_0000_0000
75	0x0000_0000	0xFFF0_0000	0x0000_0FFF	0x_0000_0000
76	0x0000_0000	0xFFF0_0000	0x0000_1FFF	0x_0000_0000
77	0x0000_0000	0xFFF0_0000	0x0000_3FFF	0x_0000_0000
78	0x0000_0000	0xFFF0_0000	0x0000_7FFF	0x_0000_0000
79	0x0000_0000	0xFFF0_0000	0x0000_FFFF	0x_0000_0000
80	0x0000_0000	0xFFF0_0000	0x0001_FFFF	0x_0000_0000
81	0x0000_0000	0xFFF0_0000	0x0003_FFFF	0x_0000_0000
82	0x0000_0000	0xFFF0_0000	0x0007_FFFF	0x_0000_0000
83	0x0000_0000	0xFFF0_0000	0x000F_FFFF	0x_0000_0000
84	0x0000_0000	0xFFF0_0000	0x001F_FFFF	0x_0000_0000
85	0x0000_0000	0xFFF0_0000	0x003F_FFFF	0x_0000_0000
86	0x0000_0000	0xFFF0_0000	0x007F_FFFF	0x_0000_0000
87	0x0000_0000	0xFFF0_0000	0x00FF_FFFF	0x_0000_0000
88	0x0000_0000	0xFFF0_0000	0x01FF_FFFF	0x_0000_0000
89	0x0000_0000	0xFFF0_0000	0x03FF_FFFF	0x_0000_0000
90	0x0000_0000	0xFFF0_0000	0x07FF_FFFF	0x_0000_0000

Bump number with fault	redund_0	redund_1	redund_2	redund_3
91	0x0000_0000	0xFFF0_0000	0x0FFF_FFFF	0x_0000_0000
92	0x0000_0000	0xFFF0_0000	0x1FFF_FFFF	0x_0000_0000
93	0x0000_0000	0xFFF0_0000	0x3FFF_FFFF	0x_0000_0000
94	0x0000_0000	0xFFF0_0000	0x7FFF_FFFF	0x_0000_0000
95	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_0000
96	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_0001
97	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_0003
98	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_0007
99	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_000F
100	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_001F
101	0x0000_0000	0xFFF0_0000	0xFFFF_FFFF	0x_0000_003F

Table 21: Redundancy control register value according to bump number fault.

For the majority of the micro bumps, there is no one signal name that can be applied to that micro bump. There are usually two possibilities. For example, from ***Table 98: Micro bump Signal Assignment*** micro bump *aib[30]* will either be *m_ns_fwd_clk* (the functional input to the multiplexer) or *tx[10]* (the redundant input). The exceptions are the four micro bumps in row 1 of ***Table 19: Three Classes of*** which do have only one signal name. Thus, after this multiplexer, we will refer to the signal as aib[i] instead of its functional name.

The natural place to locate this 2-to-1 multiplexer is at the micro bumps. In this case, each of the 102 IO buffers is associated with a single signal and this signal is routed to two different micro bumps. There are issues with the analog multiplexer required in this methodology and the location of the multiplexer was moved back further into the core of the PHY. In BCA's PHY, it is moved all the way back into the Digital portion of the channel. This means there are 102 IO buffers, each associated with a single micro bump.

This decision leads to some new complications. Using the above example, the *aib[30]* signal sent from the Digital portion to the Custom portion is now sometimes a clock signal and other times a data signal. This is not as catastrophic as it might seem. In order to minimize the skew between clock and data in the AIB's source-synchronous links, clocks are transmitted as data. The circuitry in the last SDR-to-DDR serialization stage located in the buffx1 module looks identical for clocks and data. This is shown below in **Figure 11: Buffx1 TX Serializer**.

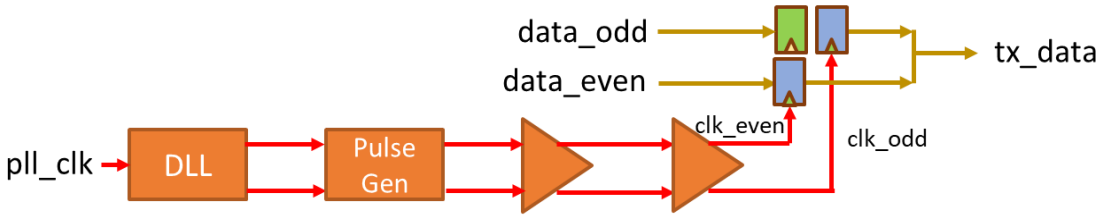

Figure 11: Buffx1 TX Serializer

There is a single clock, that is logically equivalent to the link clock, but that is not equivalent from a timing perspective, that goes to both serializers. The data serializer gets SDR data on its *data_even and data_odd* inputs that it converts to DDR data

In any given channel, whether redundancy is used or not, there are always two unused micro bumps. The driver and/or receiver associated with these unused micro bumps must be in Standby mode.

2.1.6.11 JTAG

Please, refer to AIB PHY integration guide.

2.1.6.12 Custom Support Logic

Some of the structures in the Custom block need support that is more easily provided via the COTS digital PnR design style. An example of this support is the finite-state machines (FSMs) that might be required to assist in the calibration of the DCC circuit. The logic for this support is provided in the Digital block.

2.1.6.13 TX Custom

Figure 12 illustrates a high-level, block diagram of the TX PHY data path (no clock domain information) only configured in 2:1 DDR mode. From this view, the TX serializes the data from the Digital portion of the TX data path and drives it out on the pins using either a high-voltage or a low-voltage, full swing driver depending on the mode of the link (Gen1 vs. Gen2). Post-serialized data is fed to the RX to perform near-side loopback. Data I/O redundancy is handled within the digital portion of the TX data path while clock I/O redundancy is handled by analog multiplexing.

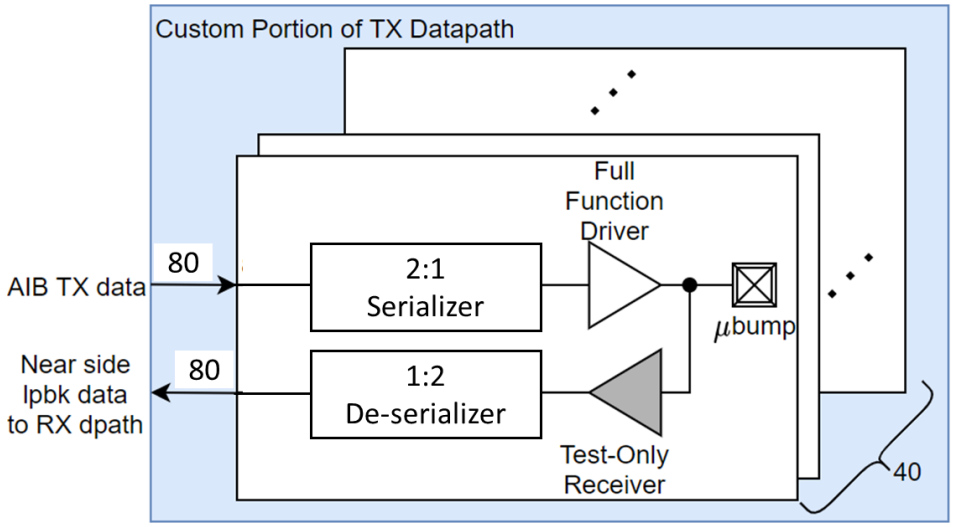

Figure 12: Custom Portion of TX Datapath

2.1.6.14 Serializer

In general, each PHY and SoC implementation can agree upon the maximum amount of drift allowed between m_wr_clk and ns_fwd_clk. The AIB PHY implements a TX FIFO which can accommodate any difference of phase between these two clocks.

The output driver supports both SDR and DDR signaling. It is also capable of driving high-voltage full-swing signals to provide compatibility with AIB Gen 1.

2.1.6.15 DCC/DLL

The PHY implements a DCC/DLL circuit to meet the duty-cycle specification in the TX across process, voltage and temperature. It also generates programmable clock phases for adapter and SoC to control latency in register mode.

2.1.7. RX Datapath

The RX data path consists of all the logic required to receive the high-speed AIB PHY inputs including: *rx[39:0],* m*_fs_fwd_clk* and *m_fs_rcv_clk*.

2.1.7.1 RX Clocking

Internally, the RX PHY will insert the forwarded clock (*fs_fwd_clk*) into a DLL that produces a clock (*analog clk*) that accurately samples the input data at the receiver. Further the buffered clock (*wr clk*) drives Rx FIFO. After being divided down to the appropriate frequency for the data rate, the PHY will drive the divided clock to the SoC as *fs_fwd_clk_div*. A version of this clock, presumably after some amount of buffering and distribution, can be driven to SoC in an optional implementation. Data from the PHY will

be launched in the *m_rd_clk* clock domain and captured in the SoC. This scheme is illustrated in **Figure 13**.

There are several differences in implementation of the forwarded clocks between the PHY and the MAC compared to AIB Spec. Specifically, *m_ns_fwd_clk* and *m_fs_fwd_clk* are not used by the AIB PHY in the same way as defined in the ***AIB Spec*** (1). Instead of using *m_fs_fwd_clk*, the AIB RX adapter receives *Rx clk Adapt* to clock one side of the FIFO. Also, there is no individual *m_fs_fwd_clk* driven to the SoC, instead both the TX and RX data paths send *ns_fwd_clk_div* and *fs_fwd_clk_div*, respectively, which can be used by the SoC internally and is used by PHY to launch data.

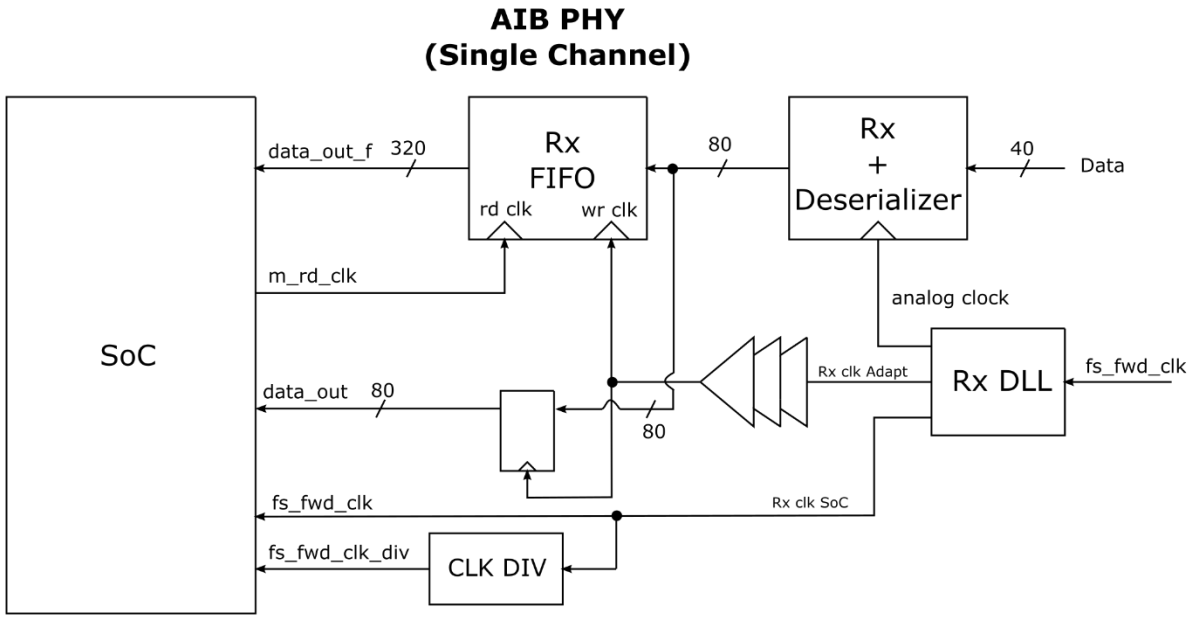

Figure 13: RX Clock Domains

The following table summarizes these changes for the RX.

AIB Spec.	BCA Impl.	Difference
m_rd_clk	*rclkSoC*	Launches *data_out_f*
m_fs_fwd_clk	*fs_fwd_clk_div*	Divided and delayed version of *m_fs_fwd_clk*
N/A	*fs_fwd_clk*	Delayed and non-divided version of m_*fs_fwd_clk*.

Table 22: Correspondence Between RX Clocks

2.1.7.2 RX Custom

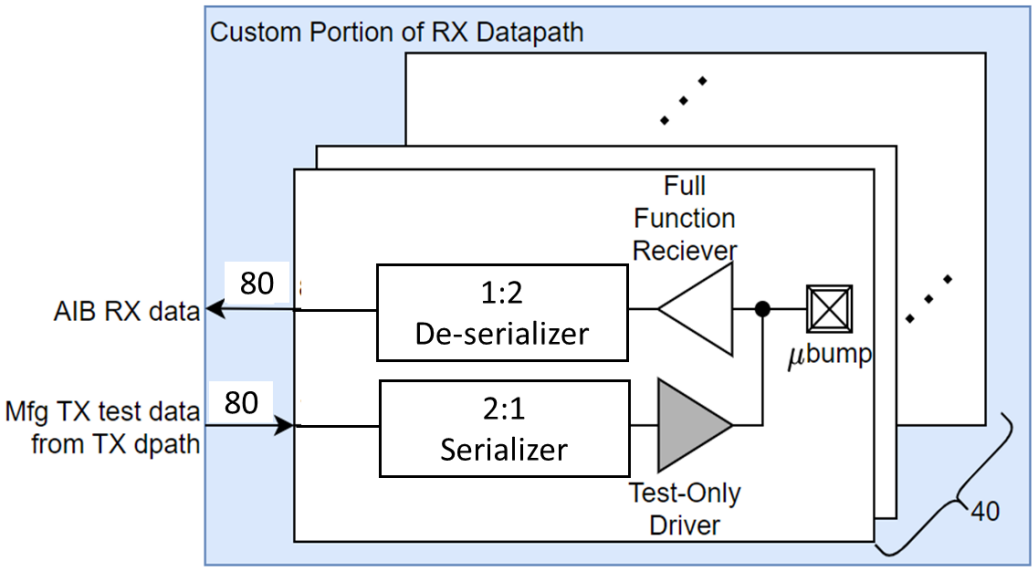

Figure 14: Custom Portion of RX Datapath

Figure 14 illustrates a high-level, block diagram of the RX PHY data path (no clock domain information) only configured in DDR mode. In this view, the RX captures the data and deserializes it before sending it to the Digital portion of the RX data path. Pre-serialized data from the TX is multiplexed into the received data stream in the RX to perform near-side loopback. Data I/O redundancy is handled within the digital portion of the RX data path and clock I/O redundancy is handled by analog multiplexing.

2.1.7.3 Input Receiver

The input receiver registers the incoming data using a skewed version of the forwarded clock produced by the local DLL. It supports both SDR and DDR signaling.

2.1.7.4 DLL

The PHY implements a DLL circuit to accurately sample the signal at the input of the receiver across process, voltage, and temperature. The Digital portion of the RX data path also implements a calibration mechanism for the DLL circuit. There is no configuration required nor available to the SoC and the implementation details are proprietary.

2.1.7.5 Deserializer

In general, each PHY and SoC implementation can agree upon the maximum amount of drift allowed between *rclkSoC* and *fs_fwd_clk_div*. AIB PHY implements a RX FIFO which can accommodate any difference of phase between these two clocks.

Like the TX data path, the data retiming register and phase compensation FIFO are implemented in the digital portion of the RX data path. The functionality implemented on the Digital portion is shown below in **Figure 15**. The left side connects to the local AIB SoC layer. The right side to the Custom portion of the RX data path. The output on the top side is the far side loopback path to the TX data path.

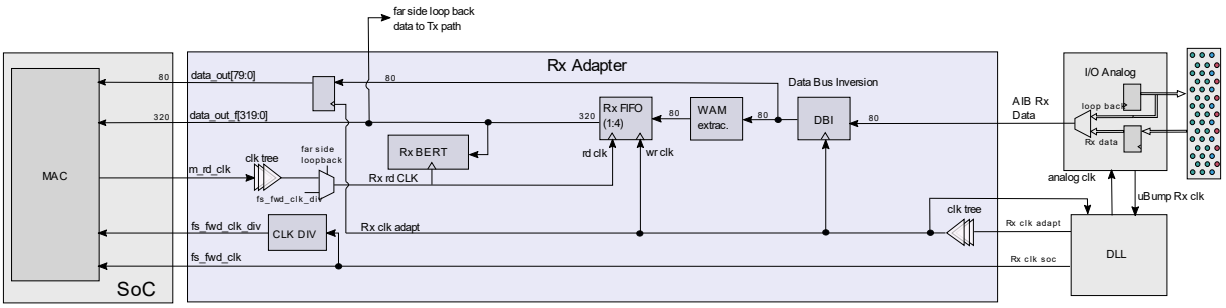

Figure 15: Digital Portion of RX Datapath

The data retiming register operates down to half the AIB interface rate (when operating the AIB IO in DDR mode) and thus needs twice the number of data bits (80) as lanes (40). The FIFO operates down to one-eighth the rate (when operating the FIFO in 4:1 mode and the IO in DDR mode) needing eight times the number of data bits (320). Both register and FIFO mode support data bus inversion (DBI) and may need those symbols extracted. The register does not need word alignment marking (WAM) symbols to be extracted from its data stream and thus bypasses this logic.

2.1.7.6 RX Data Multiplexing

Item	RX Data Multiplexing
1	JTAG
2	Redundancy
3	Loopback (requires its own redundancy)
4	Two different word widths (80 versus 40)
5	Word swapping within the DW in 40-bit, 2:1 FIFO mode

Table 23: RX Data Multiplexing

2.1.7.7 RX Data Bus Inversion

DBI extraction is like DBI insertion, just much simpler. When enabled (*rx_dbi_en* = 1), this logic examines the DBI symbol for each DBI group (the most significant bit of each 20-bit group = *rx[79,78,39,38]* for register and FIFO 1:1 modes, *rx[159,158,119,118,79,78,39,38]* for FIFO 2:1 mode and *rx[319,318,279,278,239,238,199,198,159,158,119,118,79,78,39,38]* for FIFO 4:1 mode)

and if it is asserted, then complements the remaining 19 bits of that DBI group. The DBI bit is passed through unchanged.

Register Name	Field name	Register Bit	Operation
rxadpcfg_0	***rx_dbi_en***	*1*	*0-Disable DBI*
			1-Enable DBI

Table 24: rx_dbi_en bit for Rx DBI enable

2.1.7.8 RX Word Alignment Marking

In contrast to DBI, WAM extraction is much more complicated than WAM insertion. There is still the enable bit *(rx_wa_en* in *rxadpcfg_1* register) and 5 bits in *rxadpcfg_1* register to indicate which of the data bits were replaced with the alignment marker (*rx_marker_bit79*, *rx_marker_bit78*, *rx_marker_bit77*, *rx_marker_bit76* and *rx_marker_bit39*) both of which are independent of their corresponding TX control bits. However, now there are the additional problems of locating the alignment markers in the incoming data stream, determining if the current assembly of words into DWs and QWs is correct. Once the word alignment has been done successfully, this success is communicated to SoC via the *m_rx_align_done* signal. RX side shall continue monitoring the alignment markers to verify if the data remains aligned.

Register Name	Field name	Register Bit	Operation
rxadpcfg_1	***rx_wa_en***	*0*	*0-Disable word marking*
			1-Enable word marking

Table 25: CSR bit for Rx WAM Enable

Register Name	Field name	Register Bit	Data bit used as WAM
rxadpcfg_1	**rx_marker_bit79**	7	79
	rx_marker_bit78	6	78
	rx_marker_bit77	5	77
	rx_marker_bit76	4	76
	rx_marker_bit39	3	39

Table 26: CSR bit for Rx data bits used as WAM

These new problems are solved with the finite state machine (FSM) described in detail below. But first we describe how words arrive unaligned and what must happen in the RX FIFO to realign them.

Using a DW example similar to that from the TX WAM section (**TX Word Alignment Marking**), but in this case the WAM bit position is the most significant bit of the word ([79]), the transmitter sends:

tx[159:0] = {{1'b1,*data[158:80]*},{1'b0,*data[78:0]*}}

During serialization, the transmitter sends the least significant words first (AIB is a Little-Endian protocol). Thus, the receiver will always receive least significant words first and write them into the RX FIFO first (recalling the first level of deserialization from DDR to SDR data occurs in the AIB IO prior to the RX FIFO):

FIFO Entry	WAM	Data
i	0	*data[n][78:0]*
i+1	1	*data[n][158:80]*
i+2	0	*data[n+1][78:0]*
i+3	1	*data[n+1][158:80]*

Table 27: RX FIFO in 2:1 Mode

The read port of this FIFO will assemble two adjacent entries (two words) into each DW output. There are conceptually two different ways to do this (all other valid combinations are variations of these two). The aligned way:

rx[159:0] = {FIFO[i+1],FIFO[i]} = {{1'b1,*data[n][158:80]*},{1'b0,*data[n][78:0]*}}

and the unaligned way:

rx[159:0] = {FIFO[i+2],FIFO[i+1]} = {{1'b0,*data[n+1][78:0]*},{1'b1,*data[n][158:80]*}}

Clearly, the unaligned way is wrong, but it is wrong for two reasons. The obvious, first mistake is the DBI bits are reversed in the DW. If this were the only problem, then it could be fixed by simply swapping the two words within the DW. However, there is a second mistake and that is the two words within this RX DW come from two different TX DWs. FIFO entries i and i+1 come from the same TX DW (indicated by UI *[n]*), similarly for FIFO entries i+2 and i+3 (indicated by UI *[n+1]*). RX DWs must maintain this association. Combining FIFO entries i+1 and i+2 will never result in a valid RX DW. The problem is the RX FIFO does not know this.

The only entity which does know this is the WAM extract block. It knows the relative position of the words based on the alignment process and thus the incoming data can be

stored in the FIFO in the correct order. The state diagram of the FSM which is at the heart of the WAM extract block is:

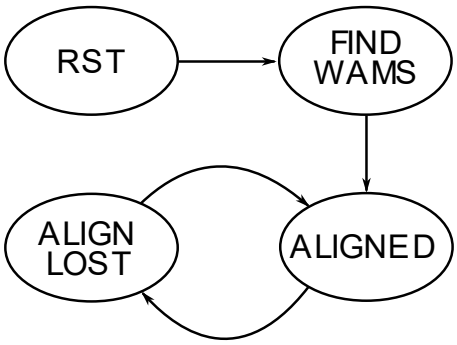

Figure 16: WAM Extract FSM State Diagram

The purpose of each state and their transitions are:

State	Description
RST	Reset: Entered whenever an interface or adapter reset is asserted. Go to FIND_WAMS state when (all resets are deasserted), (FIFO 2:1 or 4:1 mode) and (WAM enabled).
FIND_WAMS	Find word alignment markers: Search for all possible valid combinations of 2-bit or 4-bit (based on FIFO mode) WAMs in the programmed WAM bit position. If a certain number of 2-bit or 4-bit WAMs, which is configured in *rx_align_threshold* field of *rxadpcfg_1* register, is found, the words are considered aligned, and FSM goes to ALIGNED state. See **Figure 17: WAM Extract FSM State Diagram**.
ALIGNED	Words are properly aligned in their DWs or QWs. While in this state, *m_rx_align_done* is asserted to SoC. In each cycle, it compares the current WAMs to the expected combination (based on FIFO mode configuration). If WAMs are different from the expected values and ***rx_wa_mode*** is cleared, the FSM goes to ALIGN_LOST state and *m_rx_align_done* is deasserted. When ***rx_wa_mode*** bit is set, the FSM remains in the ALIGNED state regardless the detection of word alignment marker and *m_rx_align_done* is not deasserted.
ALIGN_LOST	Although alignment had been done previously, based on the current WAMs, it has been lost. While in this state, *m_rx_align_done is deasserted*. If current WAMs imply alignment, then increment a counter, else reset counter. If counter reaches the programmable threshold *rx_align_threshold*, the FSM go back to ALIGNED state.

Table 28:: WAM Extract FSM State

After the FSM is able to detect the proper word alignment based on CSR configuration, the complete data, including the WAM symbols, is written into RX FIFO and then sent to SoC.

The FSM considers the word mark alignment complete only if the configured number of consecutive patterns 2'b10 (FIFO 2x mode) or 4'b1000 (FIFO 4x mode) is detected in FIND_WAMS state:

rx_align_threshold[4:0]	Pattern for FIFO 2x mode	Pattern for FIFO 4x mode	Number of consecutive patters to be aligned (rx_align_threshold + 1)
0	2'b10	4'b1000	1
1	2'b10	4'b1000	2
2	2'b10	4'b1000	3
3	2'b10	4'b1000	4
…	…	…	…
31	2'b10	4'b1000	32

Table 29: Number of WAMs to be aligned

The figure below shows additional details about the WAMs detection logic present in FIND_WAMS state and an example illustrating the operation of FSM to complete the alignment process in FIND_WAMS state.

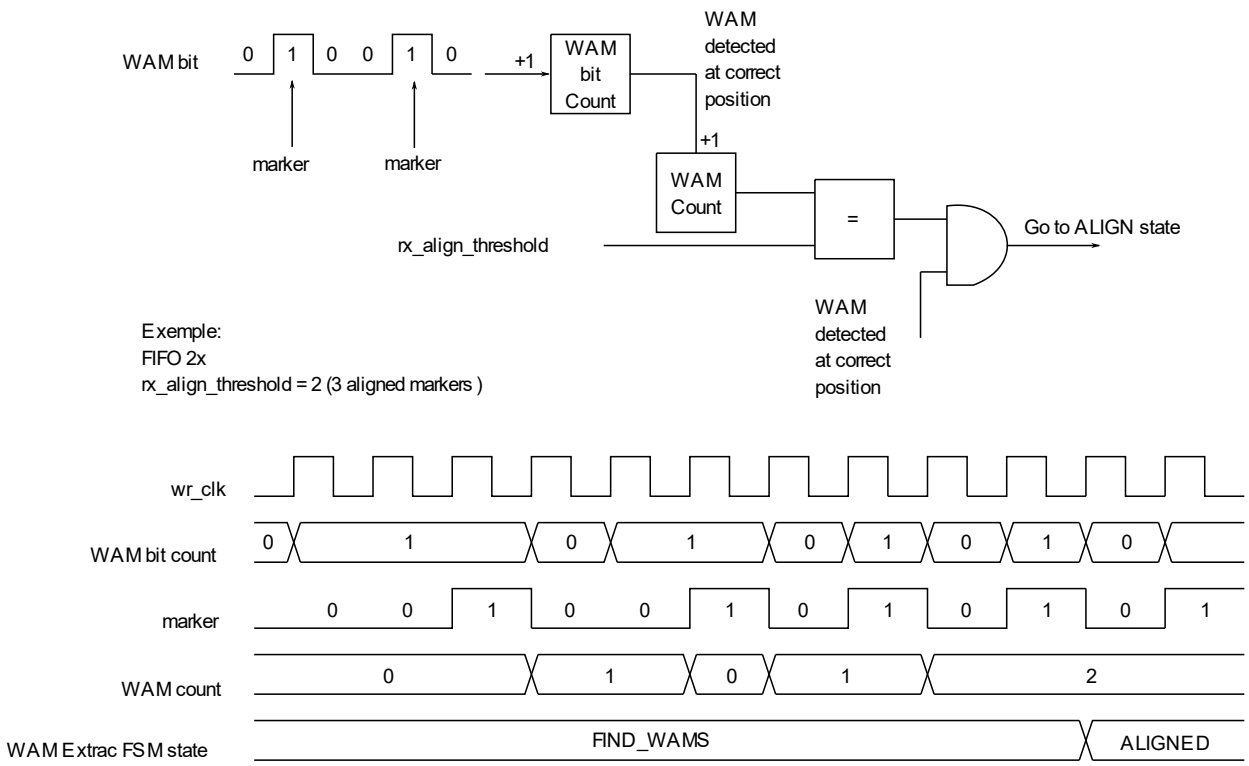

Figure 17: WAM Extract FSM State Diagram

The logic which detects the word alignment is composed basically of two counters. The first, *WAM bit count*, monitors the bit stream received in the digital interface to detect a correct marker alignment of a full word, according to the mode ({0,1} is expected for FIFO 2x mode and {0,0,0,1} is expected for FIFO 4x). If the marker is not detected at the correct position, the counter is reset. The second counter, *WAM count*, is incremented when a new valid sequence of bits is detected by *WAM bit count*. If an incorrect marker is detected by *WAM bit count, WAM count* is reset as well to restart the alignment process. Once the configured number of consecutive sequences of aligned markers is found, the WAM FSM goes to the state ALIGNED and the data starts being stored in RX FIFO and *m_rx_align_done* is asserted to SoC.

When ***rx_wa_mode*** bit is zero, the WAM FSM keep monitoring the word alignment after the alignment criteria are met, and if the word markers are not detected at the correct position, m_*rx_align_done* is deasserted by the PHY, although the data will continue being received in RX FIFO. Therefore, up to the software to issue a functional reset to restart the WAM FSM if required. If the user needs to keep m_*rx_align_done* asserted after the alignment process regardless the presence of word alignment markers, the rx_wa_mode bit should be set.

2.1.7.9 RX FIFO

RX FIFO data input is 80-bit wide and RX FIFO depth is 24. The total number of bits that can be read at one time in read domain are 80 bits, 160 bits and 320 bits for FIFO 1X, FIFO 2X and FIFO 4X mode, respectively. RX FIFO is a standard asynchronous FIFO that can operate with different clock rate between write clock and read clock depending on FIFO mode configuration. The data width and frequency of operation of its write port and its read port are 1:4, 1:2 and 1:1 for FIFO 4x, FIFO 2x and FIFO 1x modes, respectively. It registers both its input write data and its output read data. RX FIFO has a read latency which can be configured to add an integer number of additional clock periods of latency to increase its tolerance to clock uncertainty (see **Figure 18: RX FIFO read enable Generation**). When a certain set of flops are not being written by the hardware, its respective clocks are gated to save dynamic power. Basically, there is a clock gate for each 40 bits of FIFO to enable the clock of those flops only when required. For each write clock, clock is enabled for a certain group of 80 flops in gen2 mode or group of 40 flops in gen1 mode.

The RX FIFO latency can be programmed by using the *rx_phcomp* field in *rxadpcfg_0* register (see **Table 30: rx_phcomp field to program the RX FIFO latency)**.

Register Name	Field name	Register Bit	Functions
rxadpcfg_0	**rx_phcomp**	27:24	4'b1011: 11 read cycle delay
			4'b1010: 10 read cycle delay
			4'b1001: 9 read cycle delay
			4'b1000: 8 read cycle delay
			4'b0111: 7 read cycle delay
			4'b0110: 6 read cycle delay
			4'b0101: 5 read cycle delay
			4'b0100: 4 read cycle delay
			4'b0011: 3 read cycle delay
			4'b0010: 2 read cycle delay

Table 30: rx_phcomp field to program the RX FIFO latency

In order to operate properly, r*x_phcomp* field shall be configured in the range described in the table below according to FIFO mode used. These constraints can guarantee that the FIFO content in write clock domain is stable and with updated value to be read in the read domain clock.

rx_fifo_mode value	FIFO Mode	Allowed rx_phcomp value	Allowed read delay in RX FIFO
00	00- Receive FIFO 1X Mode	4'b0010-4'b1011	2-11 read cycle delay
01	01- Receive FIFO 2X Mode	4'b0010-4'b1010	2-10 read cycle delay
10	10- Receive FIFO 4X Mode	4'b0010-4'b0101	2-5 read cycle delay
11	11-Receive Register Mode	N.A.	N.A.

Table 31: Allowed Rx phase compensation according to Rx FIFO mode.

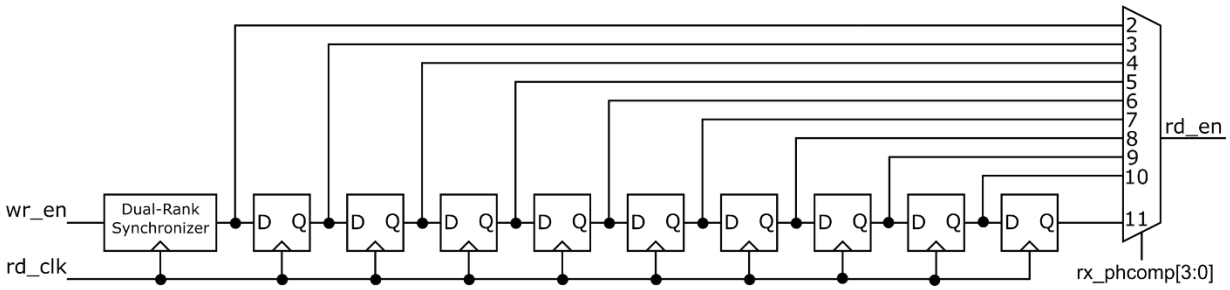

Figure 18: RX FIFO read enable Generation

RX FIFO is 2D register array of flip-flops where their clocks can be enabled or disabled by a group of clock gates depending on the FIFO configuration and the value of write pointer. The **Figure 19: RX FIFO Block Diagram** shows in detail the RX FIFO implementation.

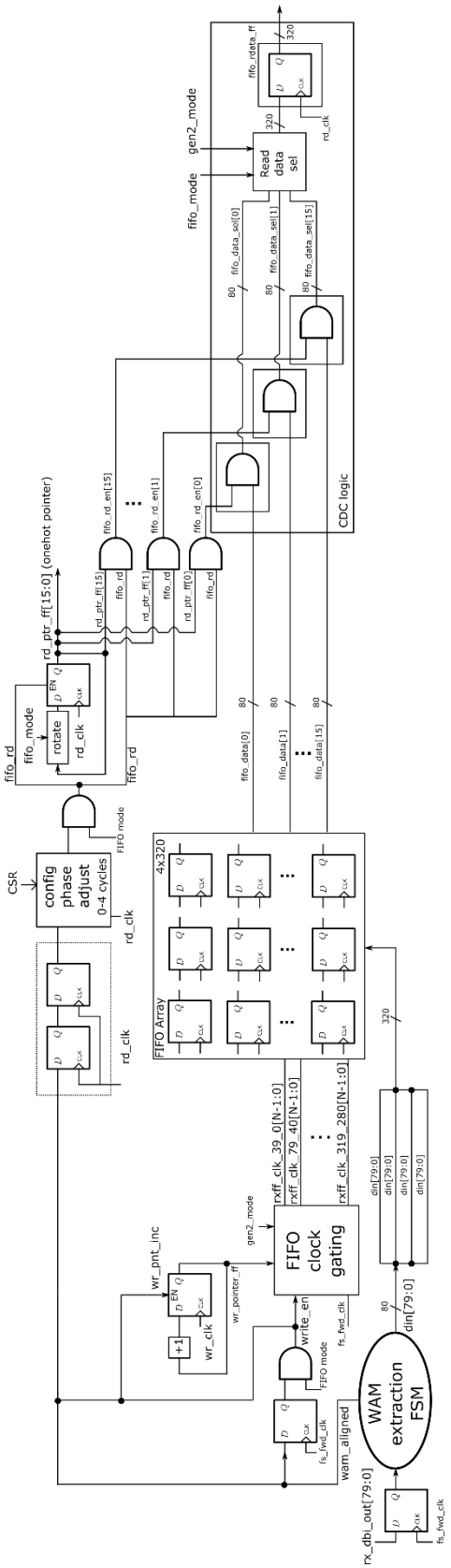

Figure 19: RX FIFO Block Diagram

2.1.7.10 RX FIFO Clock Drift

RX FIFO supports drift between read clock and write clock. The maximum clock drift depends on FIFO operation mode. **Table 32: Maximum RX FIFO clock drift** shows the maximum clock Drift supported by RX FIFO:

Mode	FIFO depth (w.r.t write side)	Maximum clock drift supported (fs_fwd_clk period)
1x	24	9
2x	24	9
4x	24	7

Table 32: Maximum RX FIFO clock drift

The figures below show the effect of clock drift for 1x, 2x and 4x TX FIFO modes.

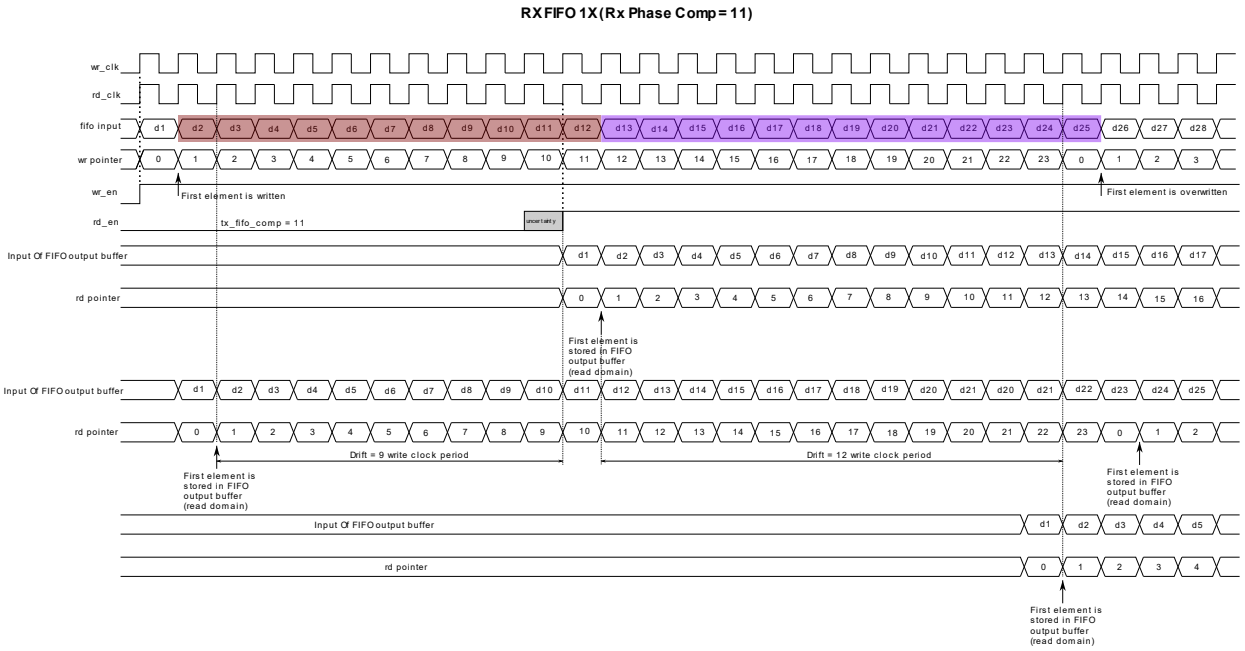

Figure 20: Clock drift in 1x RX FIFO mode

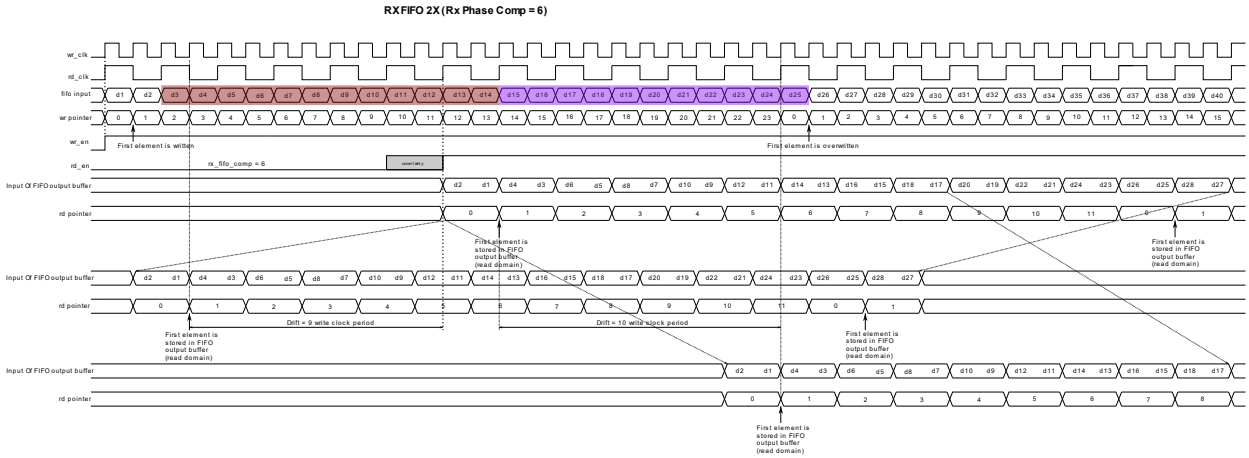

Figure 21: Clock drift in 2x RX FIFO mode

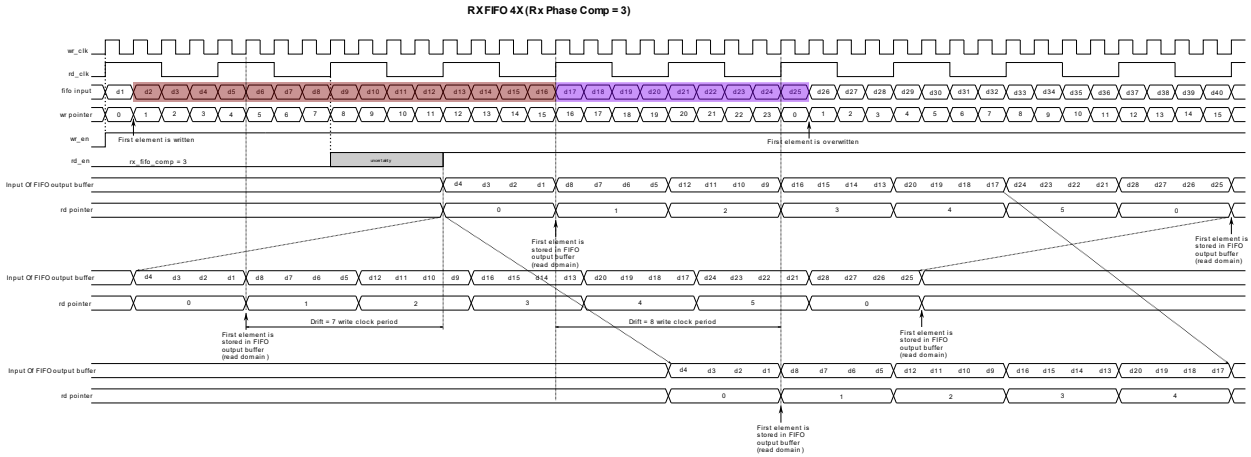

Figure 22: Clock Drift in 4x RX FIFO mode.

2.1.7.11 Custom Support Logic

Some of the structures in the Custom block need support that is more easily provided via the COTS digital PnR design style. An example of this support is the finite-state machines (FSMs) that might be required to assist in the calibration of the DLL circuit. The logic for this support is provided in the Digital block.

2.1.8 SSR RX Datapath and SSR TX Datapath

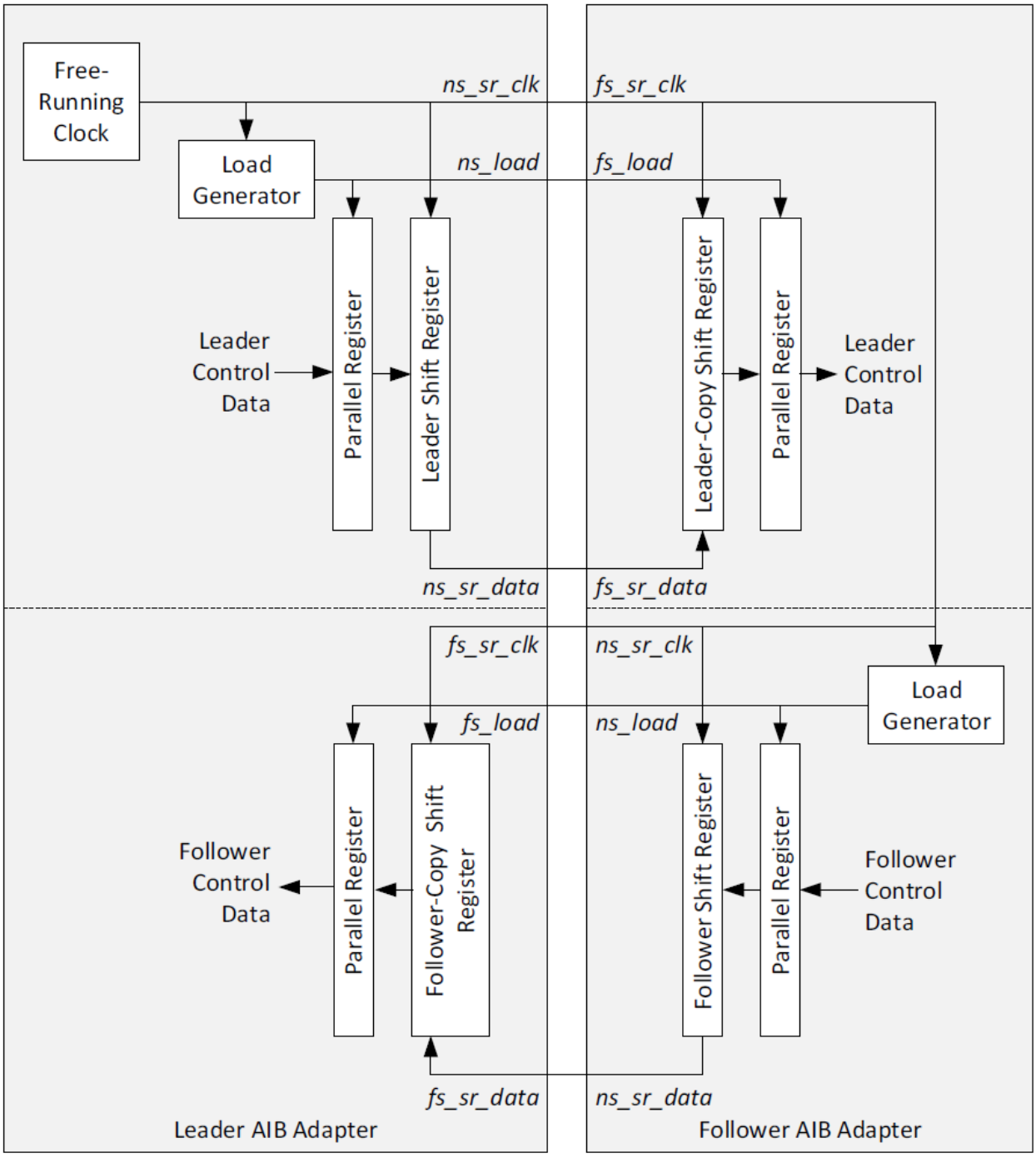

Figure 23: SSR TX and RX Datapath

2.1.9 Reset and Initialization

Initialization refers to the process of bringing the AIB link up to a functional state after the first application of power. It consists of five relatively independent phases as shown in

Figure 25. This is one more than the four phases described in the ***AIB Spec*** (1) which is shown in **Figure 24**.

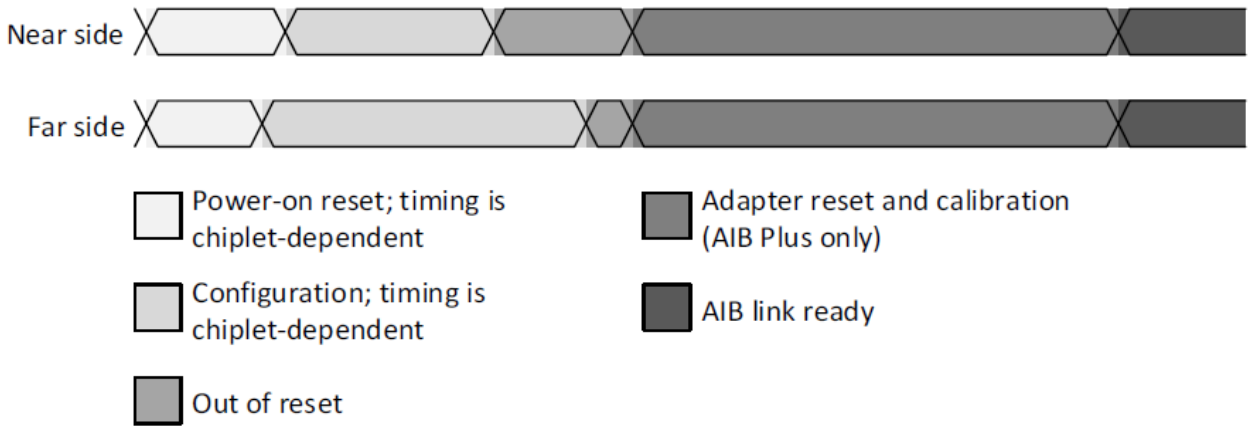

Figure 24: AIB Spec Initialization Phases

Figure 25: Extended Initialization Phases

The one new phase introduced here is call the Alignment phase. As explained below, it represents the time between the end of the Calibration phase, when high-speed data can first be reliably exchanged across the AIB link, and the beginning of the Ready phase, when application data can first be exchanged. It consists of the MAC and PHY on both sides of the link transferring word alignment markers until the receivers are able to assemble words into higher level DWs or QWs with the same alignment the transmitters send them. This phase is not required for operation in register or FIFO 1:1 mode.

2.1.9.1 PowerOnReset Phase

As shown in ***Table 33: PHY Voltage Rails***

, there are three voltage rails in the AIB PHY.

Rail	Name	Description
1	VDDC2	The digital core voltage. Supplies power to all non-IO drive circuits in the PHY. In any given AIB link, there are two VDDC rails, one for the Leader and one for the Follower, which come up independently.
2	VDDC1	This is low noise power rail used to power the clock circuits. This is shorted to VDDC2 at the board level.

3	VDDTX	Supplies power to the high-speed data channel buffers when operating in Gen1/Gen2 mode.

Table 33: PHY Voltage Rails

The PowerOnReset phase starts when the first of these rails starts to ramp indicating that power is first being applied to the PHY. The PowerOnReset phase ends when the AIB Follower deasserts *power_on_reset*. The purpose of this phase is to allow all voltages and clocks input to the PHY to become valid and stable and to provide a minimum reset pulse after these inputs are stable.

VDDC1, VDDC2 and *i_cfg_avmm_clk* are the voltage and clock inputs used in the next, Configuration phase and are required to be stable before the end of the PowerOnReset phase. Although VDDTX and the other clock inputs are not required during Configuration, we expect they will also become stable during PowerOnReset. With clock gating, it is possible that all clocks will not be running at this time. The clocks must not violate any of their timing requirements (in particular, their minimum pulse width requirements).

The only AIB interface signals which are active during the PowerOnReset phase are the two auxiliary (AUX) signals, *device_detect* and *power_on_reset*. All remaining AIB signals, both inputs and outputs, must be in Standby mode. *device_detect* is driven from the Leader to the Follower. *power_on_reset* is driven from the Follower back to the Leader. Since this is a dual-mode PHY, meaning it can operate as both a Leader and a Follower, these functionally unidirectional signals are implemented as bidirectional IOs.

Architecturally, only a single, power-on reset signal is necessary for the AIB link. This reset is generated by the application layer in the Follower. This is communicated to the Follower's PHY via its *i_m_power_on_reset* input. The Follower's PHY forwards this reset over to the Leader's PHY via the AIB signal *power_on_reset*. The Leader's PHY communicates this reset to its application layer via its *o_m_power_on_reset* output.

The Leader is able to indicate its presence to the Follower via the AIB signal *device_detect*. The Follower's PHY communicates the presence of a valid Leader to its application layer via its *m_device_detect* output. The Follower must not begin the process of exiting the PowerOnReset phase until it sees an active *m_device_detect*. An example of when this is used is an application where there is no Leader attached to the AIB link. In this example, the AIB *device_detect* signal floats and is pulled inactive by the pull-down resistor present at the Follower's input receiver. In a similar manner, in any application where there is no Follower attached to the AIB link, the *power_on_reset* signal floats and is pulled active by the pull-up resistor present at the Leader's input receiver.

Note: For proper working of analog CBBs, the VDDC supply has to be ramped up always before VDDTX supply.

2.1.9.2 Reset Signals

There are two main effects of reset:

1. To control the operation of the AIB IO buffers.
2. To initialize the state of the synchronous logic inside the PHY.

The first is used to quiesce the IO buffers, both the inputs and the outputs, when they are not in use. This non-active state of the IO buffers is referred to as their Standby mode of operation.

The normal mode of operation of the IO buffers is referred to as their Active mode.

The second effect of reset is to initialize the state of the synchronous logic inside the PHY. summarizes the state of each reset domain within the AIB PHY during the various phases of reset.

	1	2	3	4	5	6	7	8
1	"Reset" Signal	Reset CSRs	Standby AIB IO	Reset AIB Adapter	Reset OSC FSM	RX/TX CAL FSM	Reset Side Band Bus Logic	Reset Non-Standard calibration FSMs[1]
2	m_gen2_mode=1	No	No	No	No	No	No	No
3	power_on_reset=1	Yes[2]	Yes	Yes	Yes	Yes	Yes	Yes
4	i_cfg_avmm_rst_n=0	Yes	No	Yes	Yes	Yes	Yes	Yes
5	i_conf_done=0	No	Yes	Yes	Yes	Yes	Yes	Yes
6	ns_mac_rdy=0	No	No	No	No	No	No	No
7	fs_mac_rdy=0	No	No	No	No	No	No	No
8	ns_adapter_rstn=0	No	No	Yes	No	Yes	No	No
9	fs_adapter_rstn=0	No	No	Yes	No	Yes	No	No

Figure 26: State of each domain during reset.

[1] CDR machine is not affected by *i_conf_done* and *i_cfg_avmm_rst_n*.

[2] CSRs shared with all the channels (**adc0, adc1, adc2, adc3, adc4, pvta0, pvtb0, pvta1, pvtb1, pvta2, pvtb2, pvta3, pvtb3, pvta4, pvtb4 and auxch**) are affected only by *i_cfg_avmm_rst_n* reset. If these registers need to be reset after applying any reset different from *i_cfg_avmm_rst_n*, the user should write 0 into all the writable (RW) fields.

NOTE: BCA PHY implementation has an AIB 2.0 protocol non-conformance regarding the way ns_mac_rdy affects IO state. When ns_mac_rdy input is deasserted, BCA PHY does not put the IO pads corresponding to RX, TX, ns_fwd_clk/b, fs_fwd_clk/b and, for Gen 1, ns_rcv_clk/b and fs_rcv_clk/b in stand by mode. The impact of having a not stable clock within the describe condition is minimum since application needs to hold the adapter in reset until ns_mac_rdy and fs_mac_rdy are asserted and thus clock calibration will not start during the period when ns_mac_rdy is deasserted.

2.1.9.3 Reset Logic

AIB PHY has several reset sources which can come from different clock domains. Therefore, there is synchronization logic implemented on PHY to release the reset according to destination clock domain of the logic affected by a certain reset source. Moreover, some reset sources need to be mixed since more than one source can affect the same logic. The figure below shows the main logic implemented to deal with different reset sources.

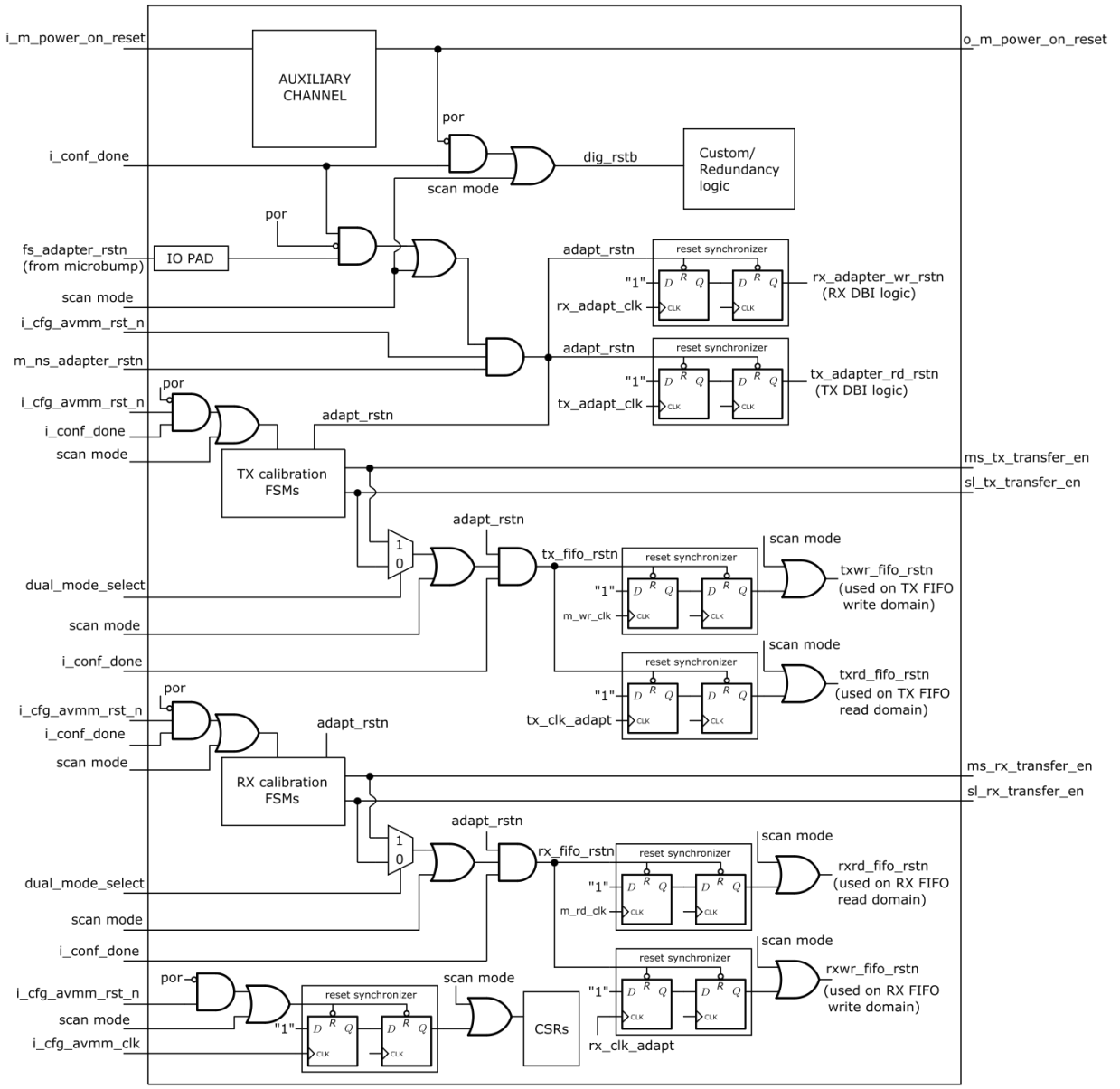

Figure 27: Reset logic.

BCA PHY implements the standard calibration machines described in AIB 2.0 protocol specification. The figure below shows how the different clock sources affect these machines.

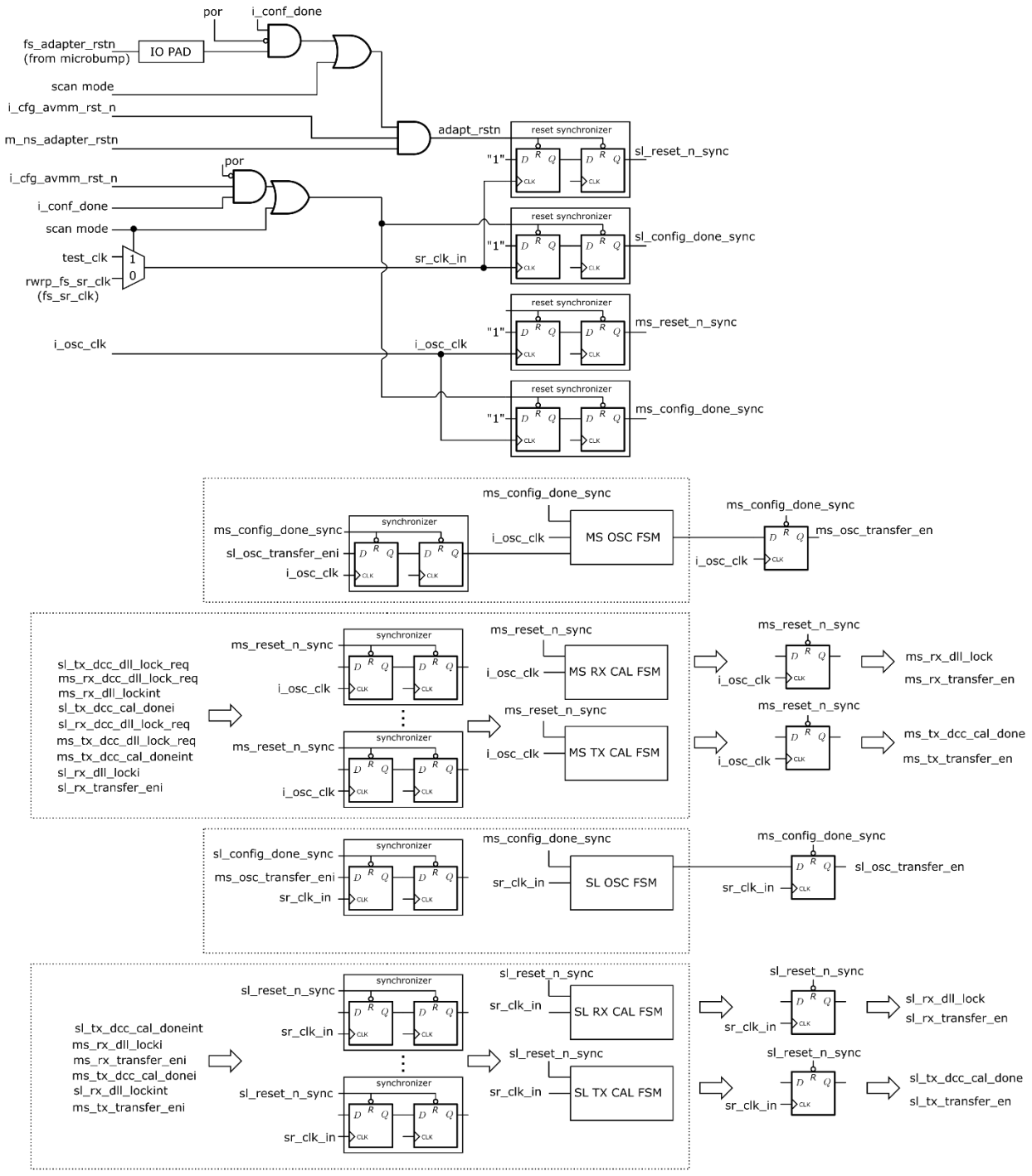

Figure 28: Reset logic of calibration state machines.

Besides calibration FSMs, reset logic affects leader and follower logic which implements side band logic. The figures below show how the different source affect the side band logic for leader and follower, respectively.

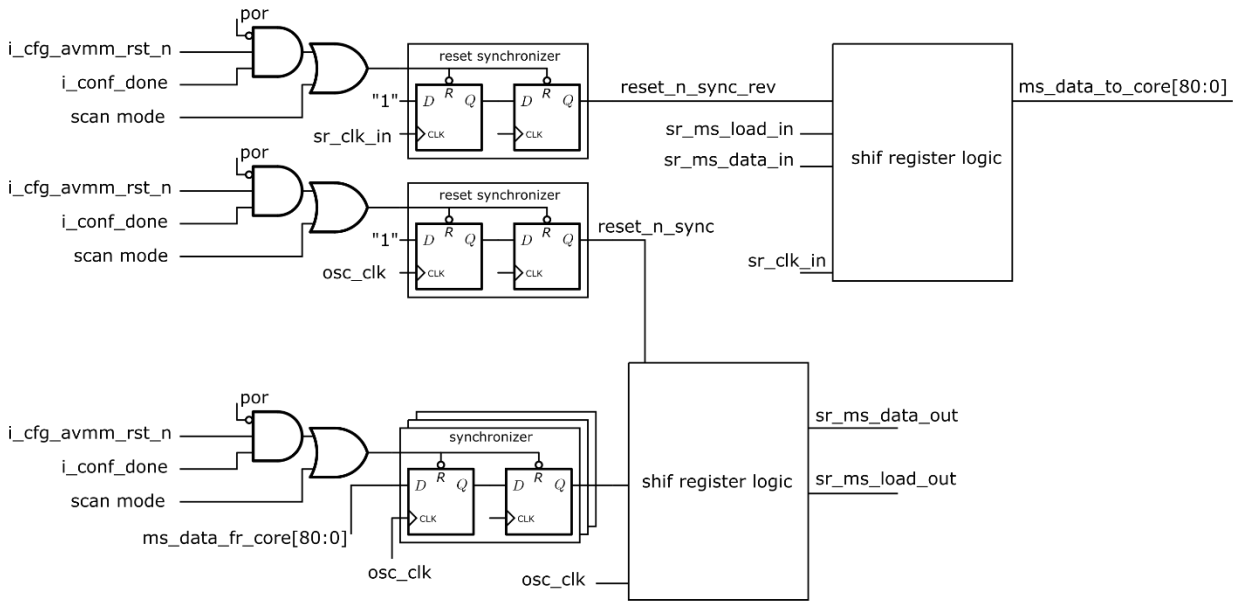

Figure 29: Reset logic of leader sideband.

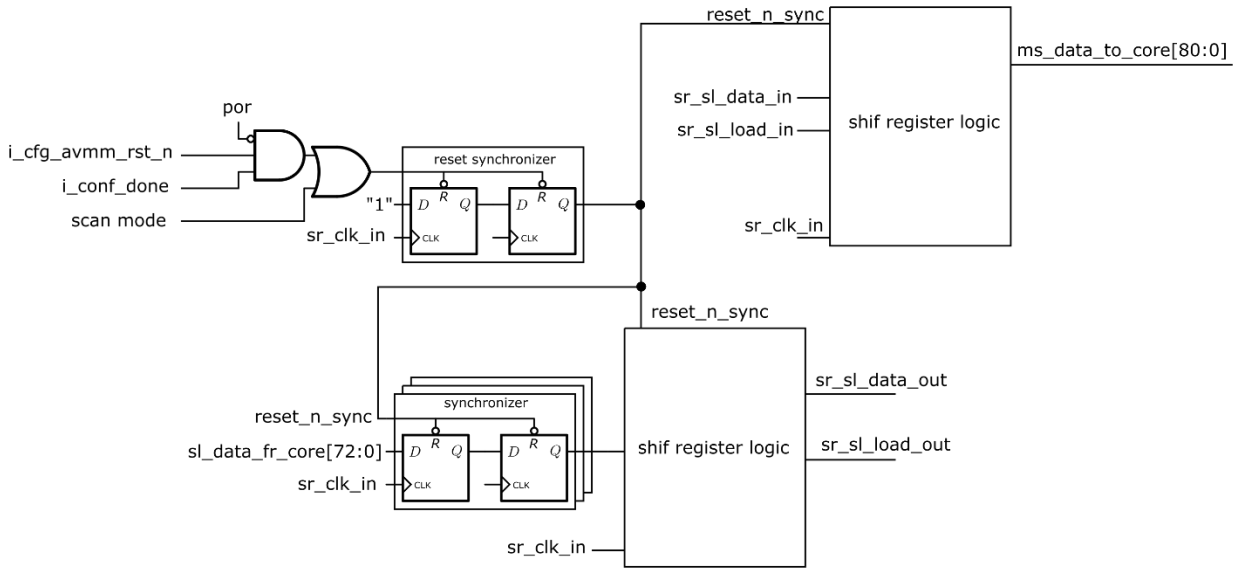

Figure 30: Reset logic of follower sideband.

2.1.9.4 Configuration Phase

The Configuration phase starts when the Follower deasserts *power_on_reset*. The Configuration phase ends when the module-wide *CONF_DONE* signal is asserted. The purpose of this phase is to allow the See **Section 3.2.2.3.5: Configuration Completion Signals** of the ***AIB Spec*** (1).

Although the programming of the CSRs is independent for the Leader and Follower in a given link, *CONF_DONE* forces transition to the next phase to wait until the slowest completes. This is true not just for the two PHYs in a given AIB link. It is true for all the AIB PHYs in the module (package).

2.1.9.5 Calibration

The user should run the following software routine to configure the proper phase of TX and RX internal clocks and to initialize some registers used by the hardware to control pull-up mechanism in analog/custom implementation. The routine below should be run only when PHY goes from configuration phase (*i_conf_done* = 0) to calibration phase (*i_conf_done* = 1). If there is an adapter reset when i_conf_done is already asserted, the routine below does not need to be run.

During configuration phase:

1. Set **vcalcode_ovrd** bit of **calvref** register.

After leaving configuration phase (i_conf_done = 1):

1. Poll the **rx_soc_clk_lock** bit of **rxdll2** register until the bit is set by the hardware.
2. Read **rx_soc_clkph_code[3:0]** field of **rxdll2** register.
3. If the value read in step 3 is less than 2:
   Write the value read in step 3 + 14 into **rxpi_socclk_code[3:0]** field of **rxdll2** register (only the four LSBs of result are used).
   Otherwise:
   Write the value read in step 3 minus 2 into **rxpi_socclk_code[3:0]** field of **rxdll2** register (only the four LSBs of result are used).
4. Read **rx_adp_clkph_code[3:0]** field of **rxdll2** register.
5. Write the value read in step 5 plus 6 into **rxpi_adpclk_code[3:0]** field of **rxdll2** register (only the four LSBs of sum are used).
6. Read **tx_adp_clkph_code[3:0]** field of **txdll2** register.
7. Write the value read in step 7 plus 8 into **txpi_aclk_code[3:0]** field of **txdll1** register.
8. Read **tx_soc_clkph_code[3:0]** field of **txdll2** register.
9. If the value read in step 9 is less than 2:

Write the value read in step 9 + 14 into **txpi_socclk_code[3:0]** field of **txdll2** register.

Otherwise:

Write the value read in step 9 minus 2 into **txpi_socclk_code[3:0]** field of **txdll1** register**.**

10. Set **rxpi_sclk_code_ovrd**, **rxpi_aclk_code_ovrd, rxsoc_lock_ovrd** and **rxadp_lock_ovrd** bits of **rxdll2** register.
11. Set **txpi_aclk_code_ovrd**, *txadp_lock_ovrd*, **txpi_sclk_code_ovrd** and **txsoc_lock_ovrd** bits of **txdll2** register.
12. Clear **vcalcode_ovrd** bit of **calvref** register.

The figure below shows the flow chart of calibration process including the point in time when the steps above to adjust clock phase should be performed.

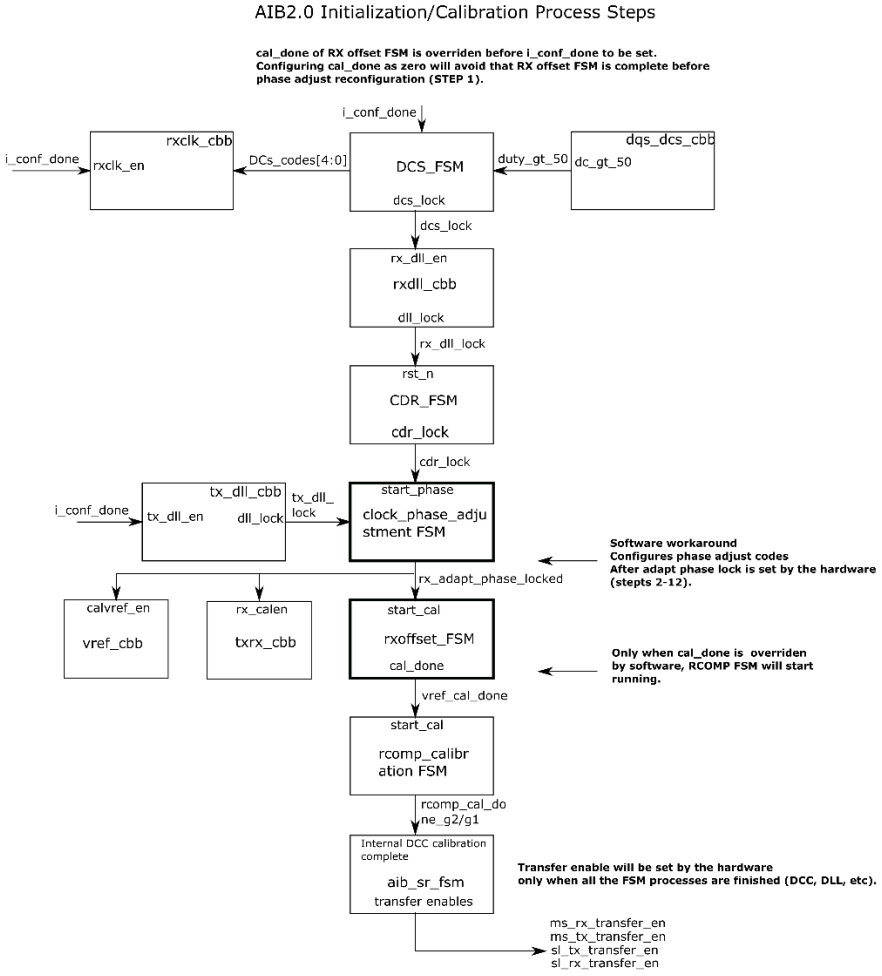

Figure 31: Calibration flow chart.

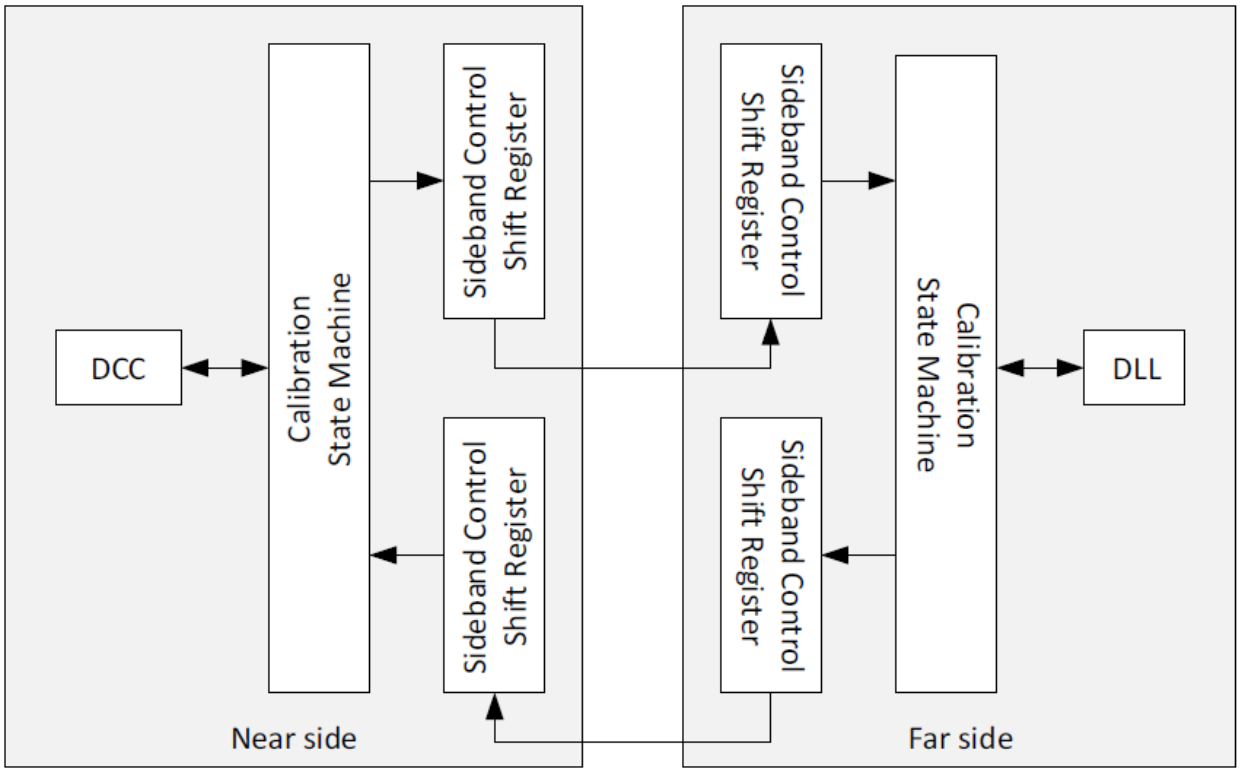

Figure 32: Calibration Architecture

The figure below shows how sideband data is transferred from leader to follower and from follower to leader depending on dual_mode_select port value. More information about which data is transfer through side band can be found in AIB 2.0 spec.

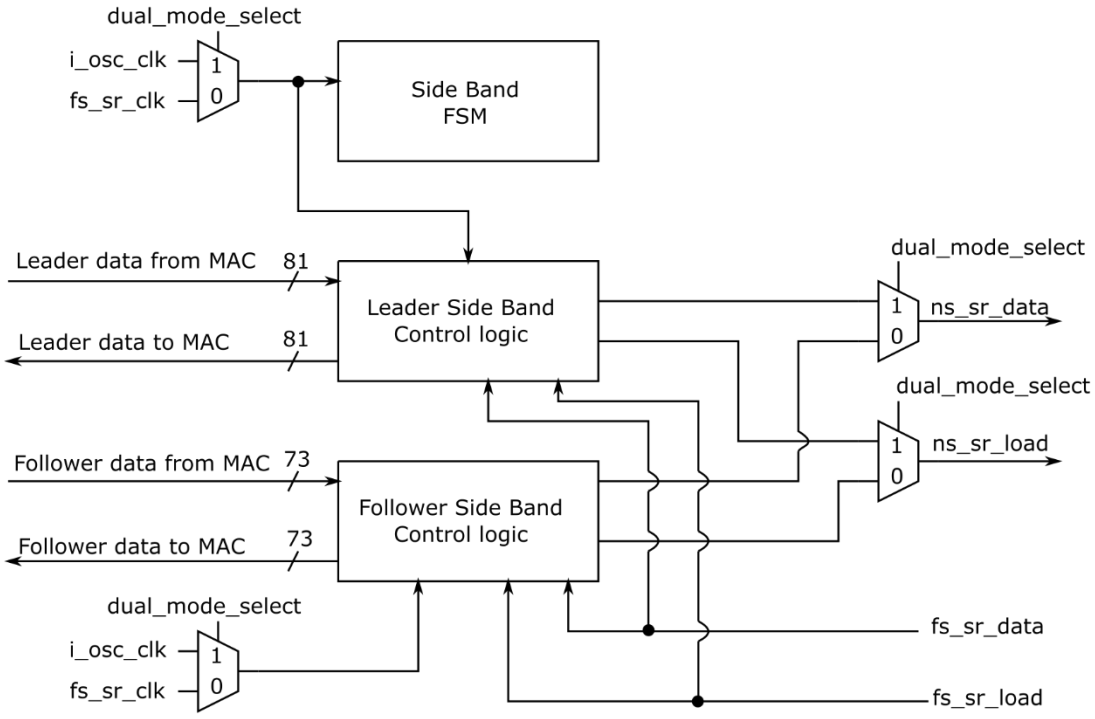

Figure 33: Sideband data transfer flow.

Leader and follower sideband control logic is basically composed of a shift register which can deserialize the data coming from the bumps and provide a data bus to MAC.

2.1.9.6 Clock Calibration

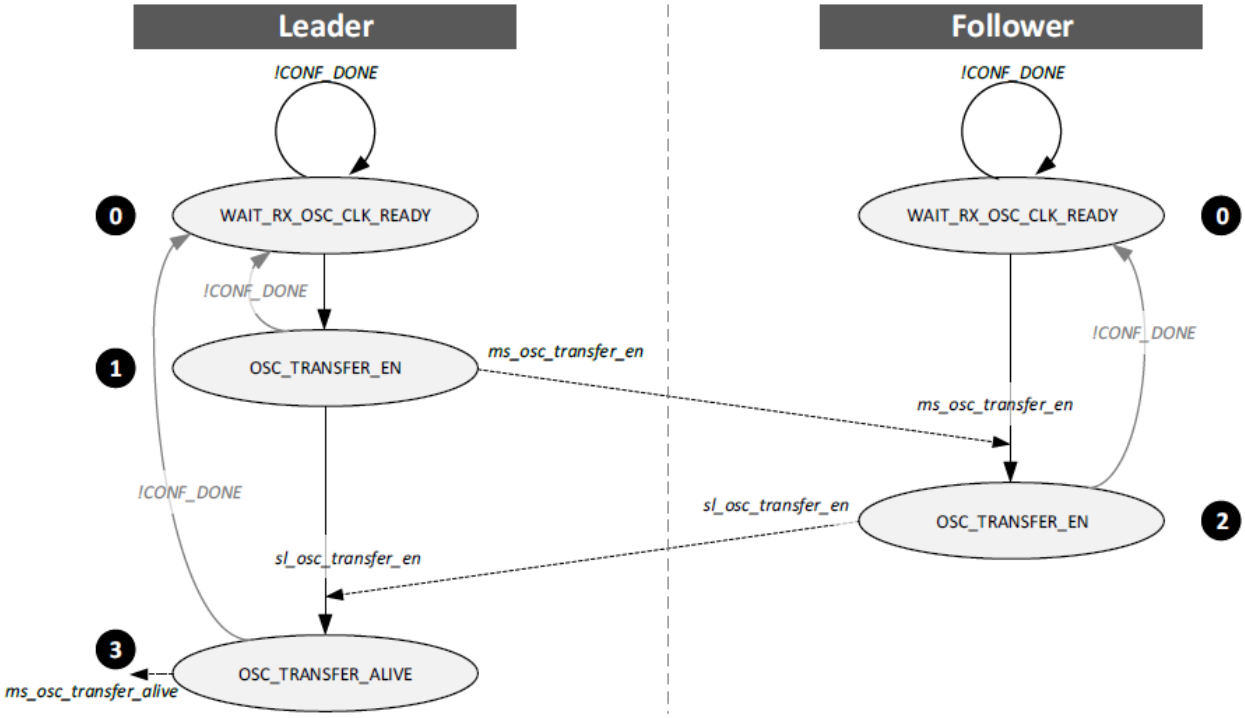

Figure 34: Clock Calibration FSM

2.1.9.7 Leader-to-Follower Calibration

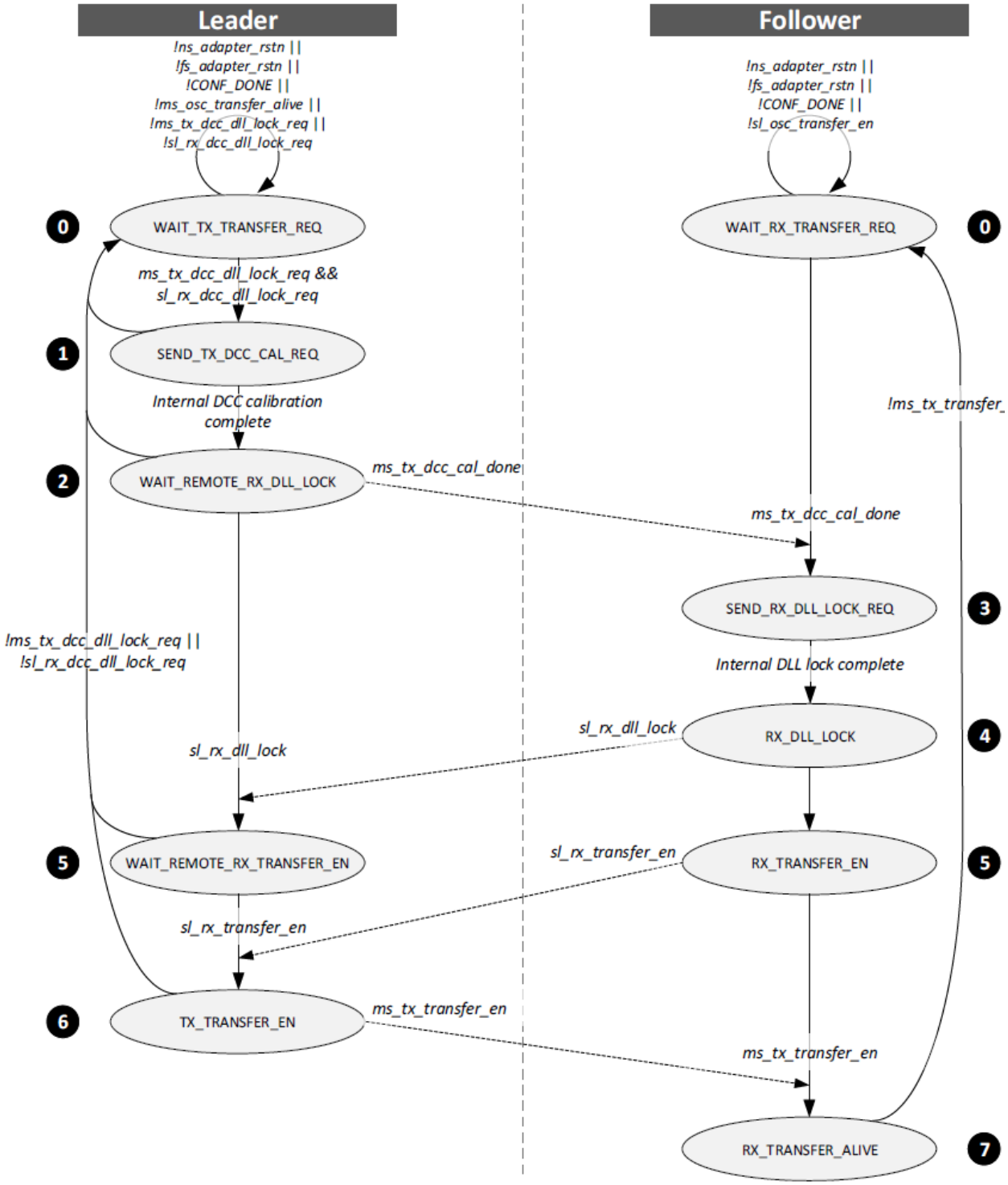

Figure 35: Leader-to-Follower Calibration FSM

2.1.9.8 Follower-to-Leader Calibration

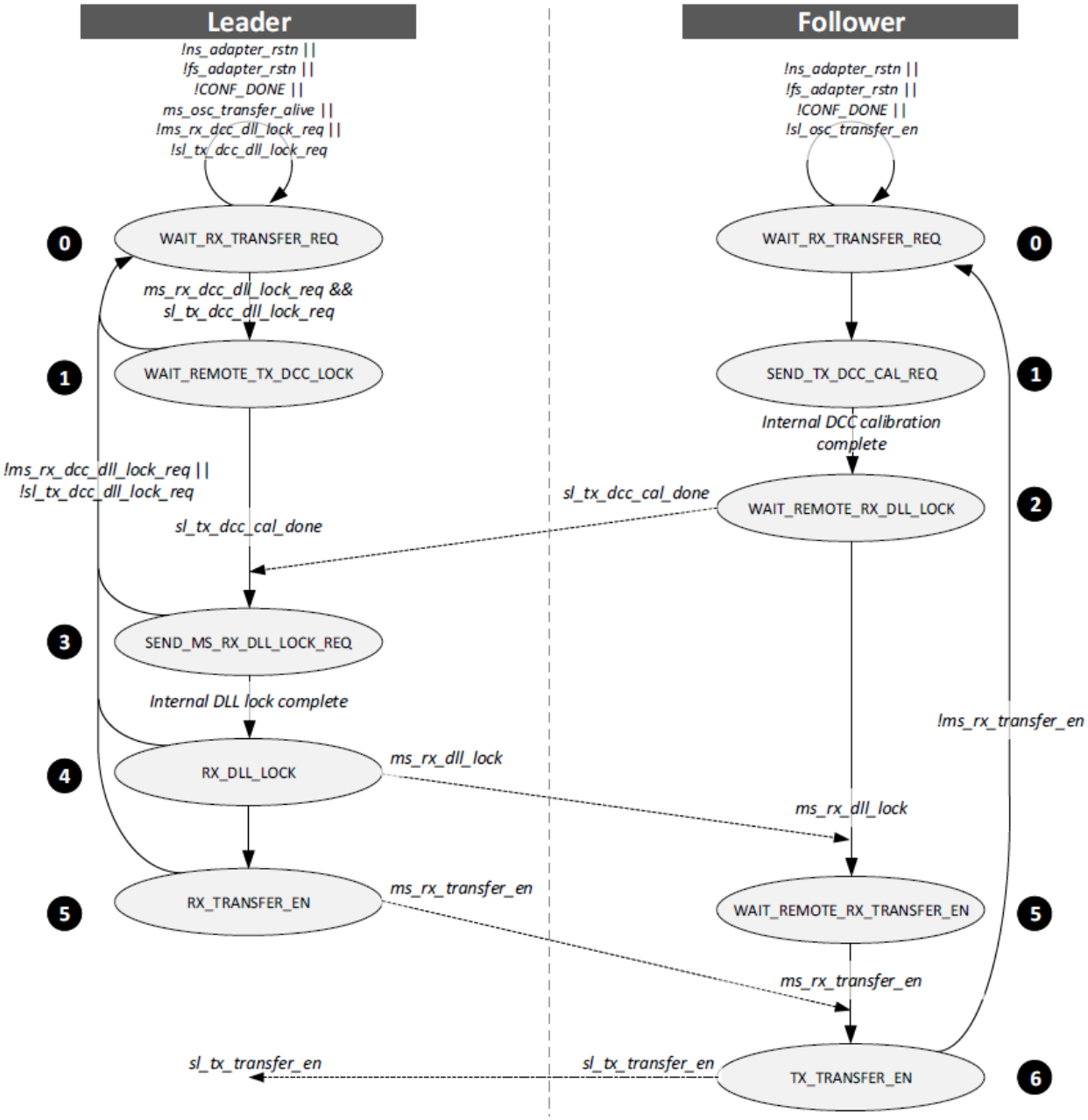

Figure 36: Follower-to-Leader Calibration FSM

2.1.10 AVMM Interface

Avmm interface is used for CSRs access. The maximum frequency of operation for i_cfg_avmm_clk clock is 200 MHz

Port Name	Width	Direction
i_cfg_avmm_clk	1	Input
i_cfg_avmm_rst_n	1	Input
i_cfg_avmm_addr	[15:0]	Input
i_cfg_avmm_byte_en	[3:0]	Input
i_cfg_avmm_read	1	Input
i_cfg_avmm_write	1	Input
i_cfg_avmm_wdata	[31:0]	Input
o_cfg_avmm_rdatavld	1	Output
o_cfg_avmm_rdata	[31:0]	Output
o_cfg_avmm_waitreq	1	Output

Table 34: AVMM Interface Signals

2.2 Bit Error Rate Tester (BERT)

BERT logic is basically composed of pattern generators in transmit side and pattern checkers in receive side. Its purpose is to provide capability of transmitting a sequence of bits in a certain lane and check the bits in the corresponding lane at receive side. The hardware implements four pattern generators whose output can be selected independently for each transmit lane, and four pattern checkers which can be used for checking simultaneously up to four receive lanes. BERT feature is not supported in register mode.

Instead of generating transmit data after Tx FIFO, just before the lanes, each pattern generator provides data to Tx FIFO input in a way that the bit sequence goes to the selected transmit lane. The same method is used on receive side where the data is checked after Rx FIFO instead of just after receiving lanes. The figure below shows transmit data path when pattern generators are used for transmitting data through the lanes.

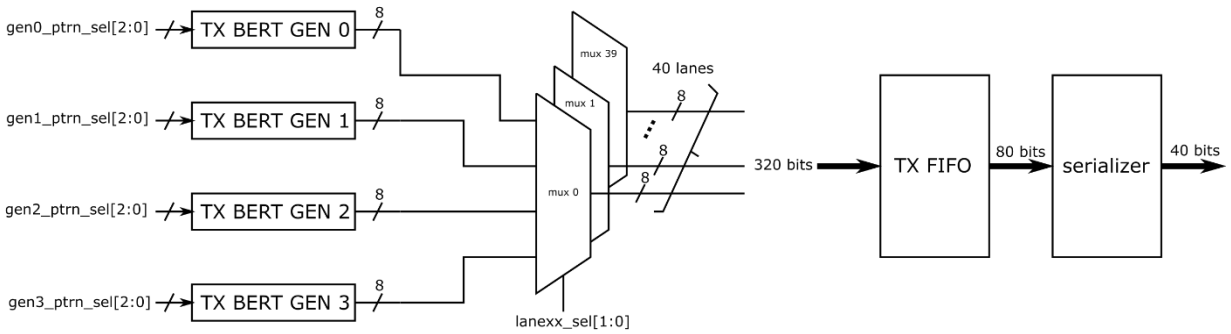

Figure 37: TX BERT Data Generators.

The figure below shows receive data path when data is received from the lanes and checked by pattern checkers.

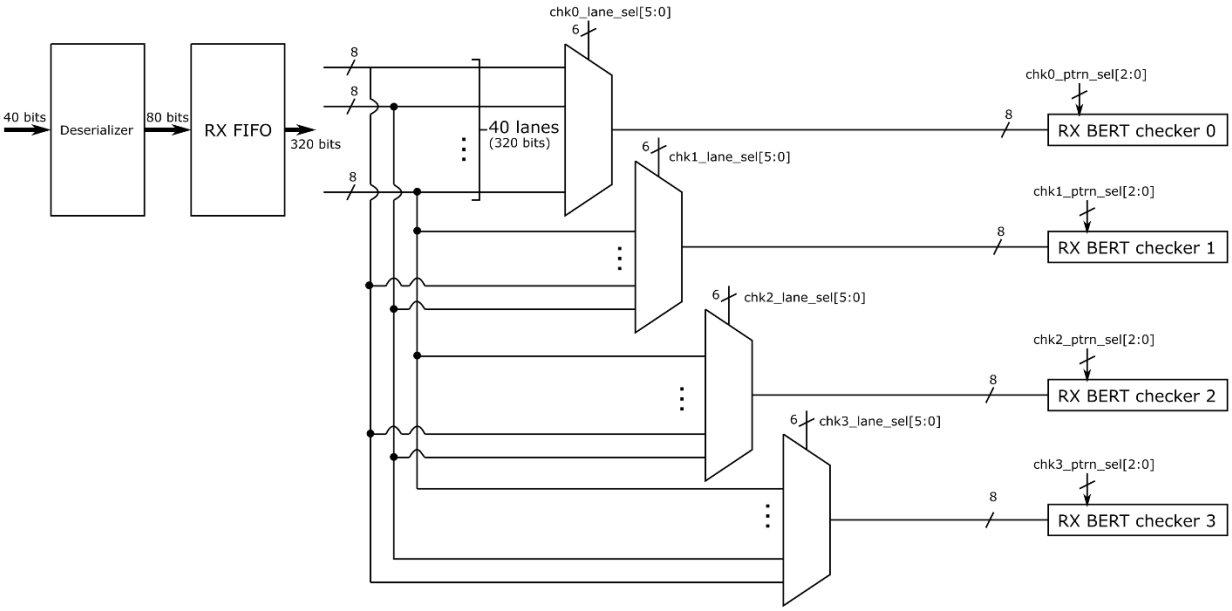

Figure 38: RX BERT Data Checkers.

Each TX BERT block can generate both pseudo-random data and deterministic data, and each RX BERT block can check both the types of data. The implementation allows to start up TX BERT generators exactly at the same time, giving total control over when the data is transmitted and what is transmitted in the lanes. All the control and status registers used on BERT feature can be accessed through host interface (Avalon).

As Avalon interface operates in a different clock domain than RX FIFO and TX FIFO, two clock domain crossing (CDC) interfaces are implemented. One for crossing data from Avalon CSR domain to TX FIFO domain and another for crossing data from Avalon CSR domain to RX FIFO domain. Both CDC interfaces are capable of transferring date in both directions, from Avalon clock domain to BERT clock domain and from BERT domain to Avalon clock domain. In order to save area, most of BERT CSRs are implemented only

on BERT domain and access is done in an indirect way through an access control register (***bert_areq***) which requests the access in Avalon domain to BERT domain and a data register (***bert_wdata***) which provides data to be written into BERT registers. In order to read data from TX BERT domain and RX BERT domain, ***txbert_rdata*** and ***rxbert_rdata*** register shall be used by the host, respectively. A status bit (***acc_rq_p*** bit in ***bert_areq*** register) indicates when the operation is pending and thus the host shall wait for the end of the transaction to read data or perform a new access in BERT domain registers. So that bit coherence in data transfer between the clock domains is kept, BERT CDC interfaces implement a handshake mechanism. The figure below shows BERT CDC interfaces.

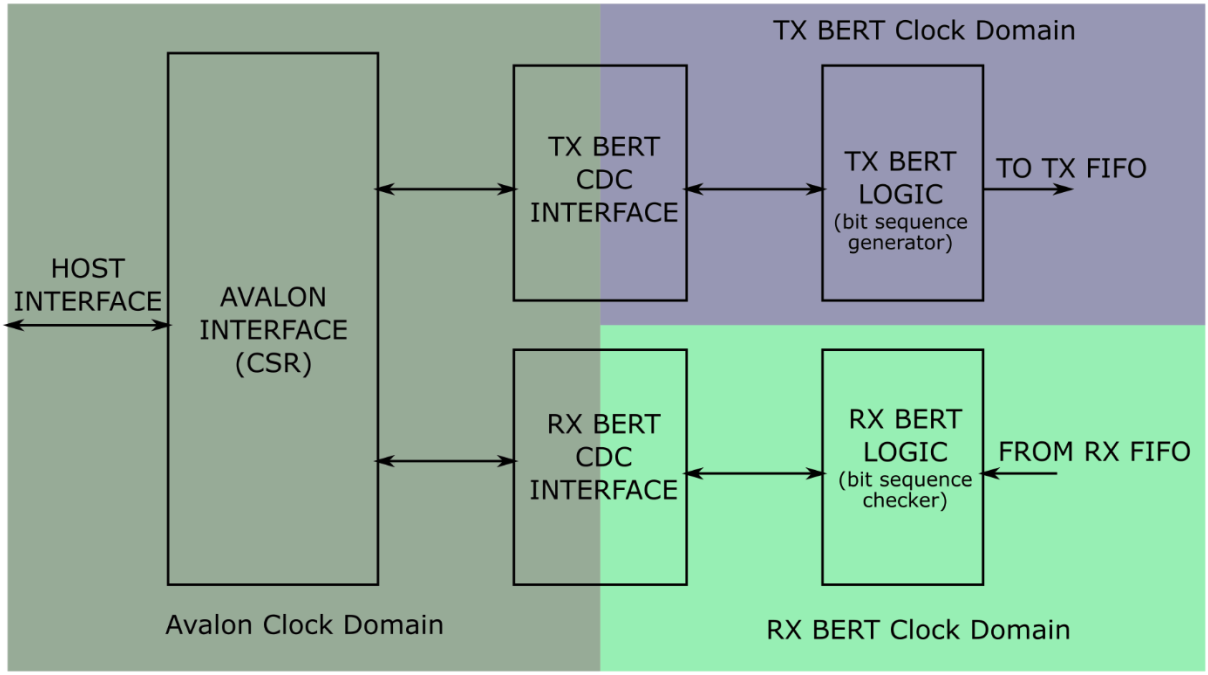

Figure 39: BERT CDC Interfaces

2.2.9 Pseudo-Random Data Generation and Data Checking

There are four different pseudo-random bit sequence (PRBS) options in each TX BERT generator block. Each of these options is capable of generating predictable data using linear feedback shift registers (LFSRs) (Fibonacci). The four polynomials used on data generators are listed in ***Table 35: PRBS Definitions***.

Organization / Standard	Sequence	Characteristic Polynomial
ITU-T O.150 [7]	PRBS-31	$x^{31} + x^{28} + 1$
ITU-T O.150 [7]	PRBS-23	$x^{23} + x^{18} + 1$
ITU-T O.150 [7]	PRBS-15	$x^{15} + x^{14} + 1$
OEIS [8]	PRBS-7	$x^{7} + x^{6} + 1$

Table 35: PRBS Definitions

Each n-bit PRBS generator can be initialized with an arbitrary seed (different from zero). In the receive side, PRBS checkers are initialized from the incoming RX data stream generated by the TX BERT block configured with the same PRBS definition.

2.2.10 Deterministic Data Generation and Data Checking

In addition to the four PRBS patters, there is also a single 128-bit data register in each TX BERT and RX BERT that can be circularly rotated to generate a deterministic sequence of bits and check the receive data. There are two ways of operating the register, The first uses all the 128 bits available in the buffer to generate a 128-bit long bit sequence and the second uses 127 bits in the register (bit 0 of buffer is not used by the hardware) to generate a 127-bit long sequence. This allows a completely arbitrary 127-bit pattern or 128-bit pattern to be repeatedly transmitted, received and checked for correctness. As the length of PRBS-7 is 127 (2^n -1), it is possible to configure the buffer to generate the same bit sequence. Although there is redundance, this allows to check the PRBS-7 transmitted bits with the 127-bit buffer checker or vice versa (it is important to note that PRBS-7 is not able to generate or check all the possible sequences of 127 bits as deterministic buffer can do). The deterministic data checker buffer is initialized using the incoming data from TX BERT as performed for PRBS data.

2.2.11 BERT configuration

BERT supports FIFO mode only, and thus register mode shall not be used. Marker alignment shall be disable during BERT operation in receive side and transmit side. In addition, DBI and swapping feature shall be disabled in both transmit side and receive side. The following steps should be performed by the host to configure the BERT to generate the bit sequence on transmit side and check the respective bits on receive side:

For TX BERT configuration:

1) During BCA PHY configuration phase, set ***tx_bert_en*** bit in ***txadpcfg_1*** register, to select TX BERT generator as data source of TX FIFO.
2) After configuration phase (CONF_DONE = 1) and calibration phase (all transfer enable signals are set by the hardware), write into ***bert_wdata*** register the seed bits according to **Table 36: BERT seed bits vs bit pattern**:

Sequence	Seed bits (x = 0 to 3)
PRBS-31	seedp3_x[31:1]
PRBS-23	seedp3_x[31:9]
PRBS-15	seedp3_x[31:17]
PRBS-7	seedp3_x[31:25]
127-bit pattern	{seedp3_x[31:0], seedp2_x[31:0], seedp1_x[31:0], seedp0_x[31:1]}
128-bit pattern	{seedp3_x[31:0], seedp2_x[31:0], seedp1_x[31:0], seedp0_x[31:0]}

Table 36: BERT seed bits vs bit pattern

Then, write ***acc_rdwr***, ***acc_addr*** and ***acc_req*** fields in ***bert_areq*** register to perform a write access into seedp3_x in BERT domain.

The host shall wait the hardware to clear ***acc_rx_p*** field in ***bert_areq*** register to indicate there is no pending transaction.

If it is required, repeat step 1 for seedp***0_x***, ***seedp1_x***, ***seedp2_x*** registers.

For 127-bit and 128-bit patterns, the seeds correspond exactly to the transmitted value (first transmit bit is on the left).

3) Configure ***GEN_SEL0*****,** ***GEN_SEL1*** and ***GEN_SEL2*** **TX** BERT domain registers using ***bert_wdata*** and ***bert_areq*** registers, similarly as described in the previous step, to select one TX BERT generator for each lane (there are four independent TX BERT generators).
4) Configure ***GEN_PTRN_SEL*** register using ***bert_wdata*** and ***bert_areq*** registers to select the transmit pattern option for the four TX BERT generators.
5) Write one into the bits of ***tx_start*** field in ***txbert_ctrl*** register that correspond to each TX BERT generator used. It is important to write simultaneously all the bits into ***tx_start*** field so that all the TX BERT generators start generating the sequence of bits at the same time.

If it is necessary to the reconfigure TX BERT generators, ***tx_rst*** field in ***txbert_ctrl*** register can be used. Tx BERT status register (**txbert_sta**) provides information about the status of each TX BERT generator.

In order to read any TX BERT register which is implemented on TX BERT domain, the host should write the field in ***bert_areq*** register and read the value of ***txbert_rdata*** after ***acc_rx_p*** field in ***bert_areq*** register indicates there is no pending transaction.

For RX BERT configuration:

1) Configure ***RX_LANE_SEL*** register using ***bert_wdata*** and ***bert_areq*** registers to select the lanes which will be connected to each RX BERT checkers. As there are four checkers, the maximum number of lanes that can be checked at a time is four.
2) Configure ***CHK_PTRN_SEL*** register using ***bert_wdata*** and ***bert_areq*** registers to select bit sequence pattern to be checked on each RX BERT checker.
3) After TX BERT is already sending data, write one into the bits of ***seed_in*** field in ***rxbert_ctrl*** register for the used checkers. This procedure will initialize the RX BERT side with TX BERT seeds.
4) Wait for a period of time equivalent to 560 *m_ns_fwd_clk* clock periods.
5) Write one into the bits of ***rx_start*** field in ***rxbert_ctrl*** register that correspond to each RX BERT checker used. It is important to write simultaneously all the bits in ***rx_start*** field so that all RX BERT checkers start checking the sequence of bits at the same time.

The host shall wait the hardware to clear ***acc_rx_p*** field in ***bert_areq*** register to indicate there is no pending transaction before executing any new transaction to access the registers in TX BERT domain and RX BERT domain.

If it is necessary to the reconfigure RX BERT checkers, ***rx_rst*** field in ***rxbert_ctrl*** register can be used. RX BERT status register (**rxbert_sta**) provides information about the status of each RX BERT checker.

In order to read any RX BERT register which is implemented on RX BERT domain, the host should also write the fields in ***bert_areq*** register and wait the hardware to clear ***acc_rx_p*** field in ***bert_areq*** register to indicate there is no pending transaction. Then, the register value can be read in ***rxbert_rdata*** register.

The field description and address of TX BERT domain and RX BERT domain registers can be found in **Table 92: TX BERT Domain registers** and **Table 93: RX BERT Domain registers**, respectively.

2.2.12 BERT Counters and Receive Bits

BERT feature implements a unique bit counter for all the checkers and one bit error counter per each RX BERT checker. Bit counter is 49-bit wide, thus it is able to count bits during about 39 hours when the bit rate is 4 Gbps without wrapping around. The value stored in bit counter represents the total number of bits received in a single checker. Therefore, the user should multiply the value stored in the bit counter by the total number

of checked lanes to have the total number of bits. The bit error counters, ***RXBERT0_ ERRCNT, RXBERT1_ ERRCNT, RXBERT2_ ERRCNT*** and ***RXBERT3_ ERRCNT***, are 16-bit wide and their count is saturated in the value 0xFFFF.

The host can read the ***biterr*** bit of ***rxbert_sta*** register to check if errors were detected by RX BERT checkers. The error counters (***RXBERT0_ ERRCNT, RXBERT1_ ERRCNT, RXBERT2_ ERRCNT*** and ***RXBERT3_ ERRCNT*** registers) shall be read using ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter.

As bit count is 49-bit wide and register interface is 32-bit wide, the host shall perform two access in the following order:

1) Read ***RXBERT_ BITCNT_LO*** register through ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter (The value of ***RXBERT_ BITCNT_LO*** register is stored in a temporary buffer to be read in step 2).
2) Read ***RXBERT_ BITCNT_HI*** register through ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter to access the remaining bits of the register.

BERT implementation also stores the last 128 bits received in each checker that can be read using ***bert_areq and rxbert_rdata.*** The host shall perform four access in the following order to access the complete data received in RX BERT checker 0:

1) Read ***RXBERT0_ DATA0*** register through ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter (The values of ***RXBERT0_ DATA1, RXBERT0_ DATA2 and RXBERT0_ DATA3*** registers are stored in a temporary buffer to be read in step 2, step 3 and step 4).
2) Read ***RXBERT0_ DATA1*** register through ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter to access the remaining bits of the register.
3) Read ***RXBERT0_ DATA2*** register through ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter to access the remaining bits of the register.
4) Read ***RXBERT0_ DATA3*** register through ***bert_areq*** and ***rxbert_rdata*** registers as explained in **2.2.11** chapter to access the remaining bits of the register.

The same steps can be done to access **RXBERT1_DATAx, RXBERT2_DATAx, RXBERT3_DATAx** registers.

2.3 Loopback Testing

2.3.1 Near Side Loopback

The near side loop is implemented using the test-only receivers present in each TX cell and test-only transmitters presented in each RX cell. During the loop back mode same data is driven to both TX and RX pads. This results in 2 sets of loopback data, one from TX cells and other from RX cells. A config register controls the multiplexing between these 2 data sets allowing individually check the TX and RX pads.

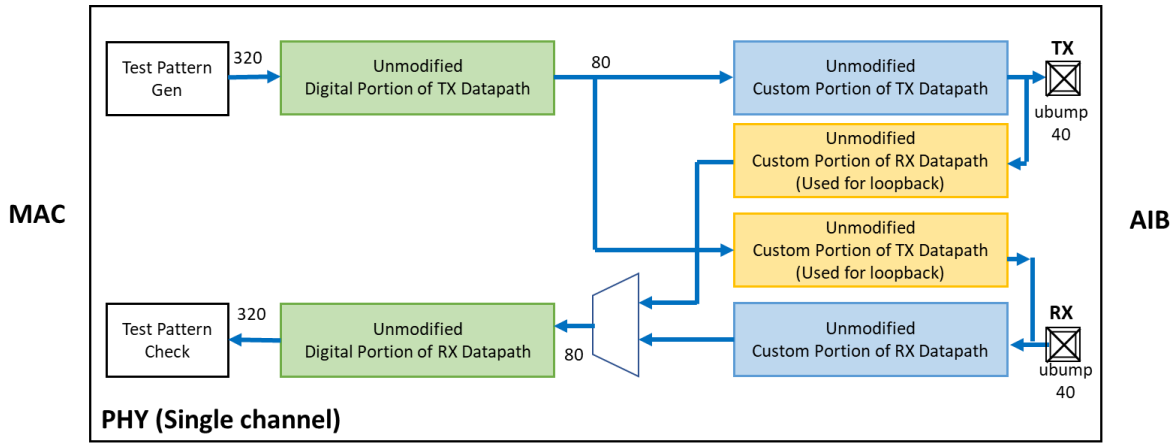

Figure 40: Near Side Loopback Path BCA Implementation

The clock for TX follows the same path as normal operation i.e., output of TX DCC/DLL is driven to both TX pads and test-only TXs. The receive clock path has 2 options. One option is used the receivers as used in normal mode since the receive clock pads are driven by test-only driver present on the RX clock pads allowing the receiver clock to propagate from RX DLL to all the RXs. The second option is to use the TX clock from TX DCC/DLL for RX DLL. A config register allows to select between these 2 modes. The clock alignment in both the cases it achieved in a similar manner as the normal operation.

PHY shall be configured as master when operating in near side loopback (PHY configured as slave is not supported). In order to operate properly in nearside loopback, other PHYs connected to the same micro bumps of BCA PHY shall keep their PADs in high impedance mode (high-Z).

Field Name			Near side loopback mode
loopback_mode	**rx_lpbk_en**	**tx_lpbk_en**	
2'b10	**1'b0**	**1'b0**	Transmit data is driven in TX IO pads and RX IO pads. Receive data comes from TX IO pads.
2'b00	**1'b1**	**1'b0**	Transmit data is driven in RX IO pads. Receive data comes from RX IO pads.
2'b00	**1'b0**	**1'b1**	Transmit data is driven in TX IO pads. Receive data comes from TX IO pads.

Table 37: Near side loopback mode.

2.3.2 Far Side Loopback

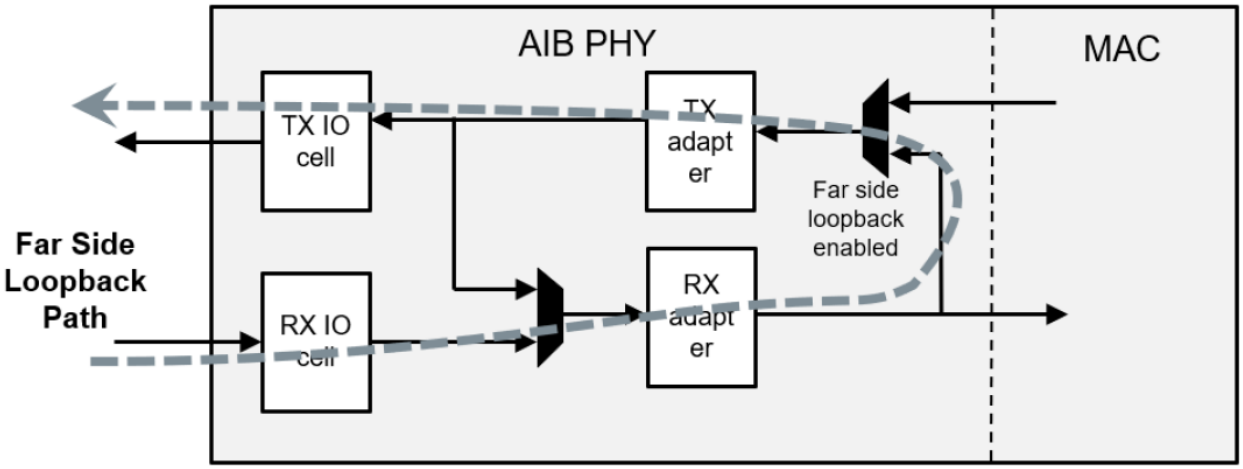

Figure 41: Far Side Loopback Path from AIB Spec

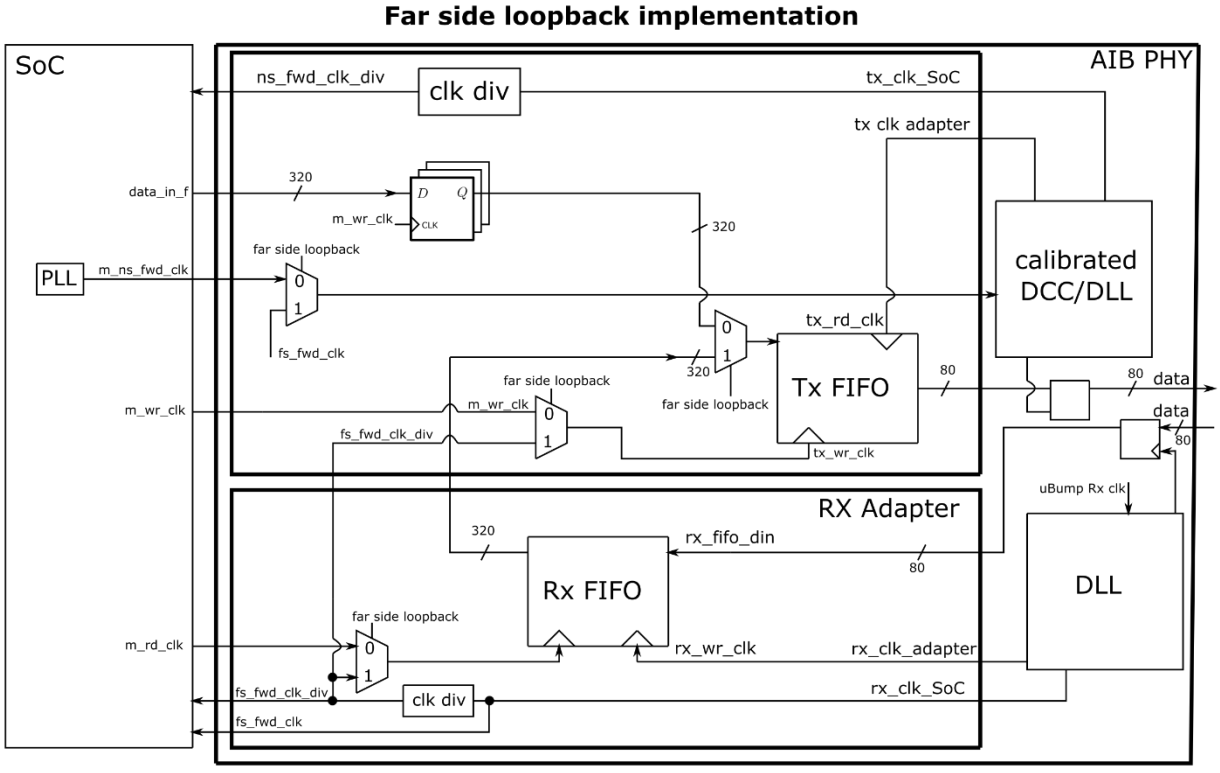

Figure 42: Far Side Loopback Path BCA Implementation

Far side loopback is as documented in the ***AIB Spec*** (1). The only interesting detail is loopback data must go through the two (one RX and one TX) digital FIFOs. Far side

loopback in register mode is not supported. In far side loopback, SoC shall still provide m_ns_fwd_clk since some internal PHY machines use the clock (divided down) to run calibration process. In far side loopback, the nominal frequency of m_ns_fwd_clk should be the same as the nominal frequency of far side clock.

2.3.3 Forwarded Clock Observability

In near side loopback mode, it is not possible to loopback ns_fwd_clk to fs_fwd_clk since the clock source is not sampled at the output of IO buffer in custom implementation. In order to check clock IO buffer logic, BERT logic can be used. For test purpose, it is possible to use TX BERT pattern generate to drive data into IO buffer input and use RX BERT checker to monitor the same data in IO buffer output. The figure below shows how to use BERT mechanism to check the clock path of *ns_fwd_clk*, *ns_fwd_clkb*, *fs_fwd_clk* and *fs_fwd_clkb*.

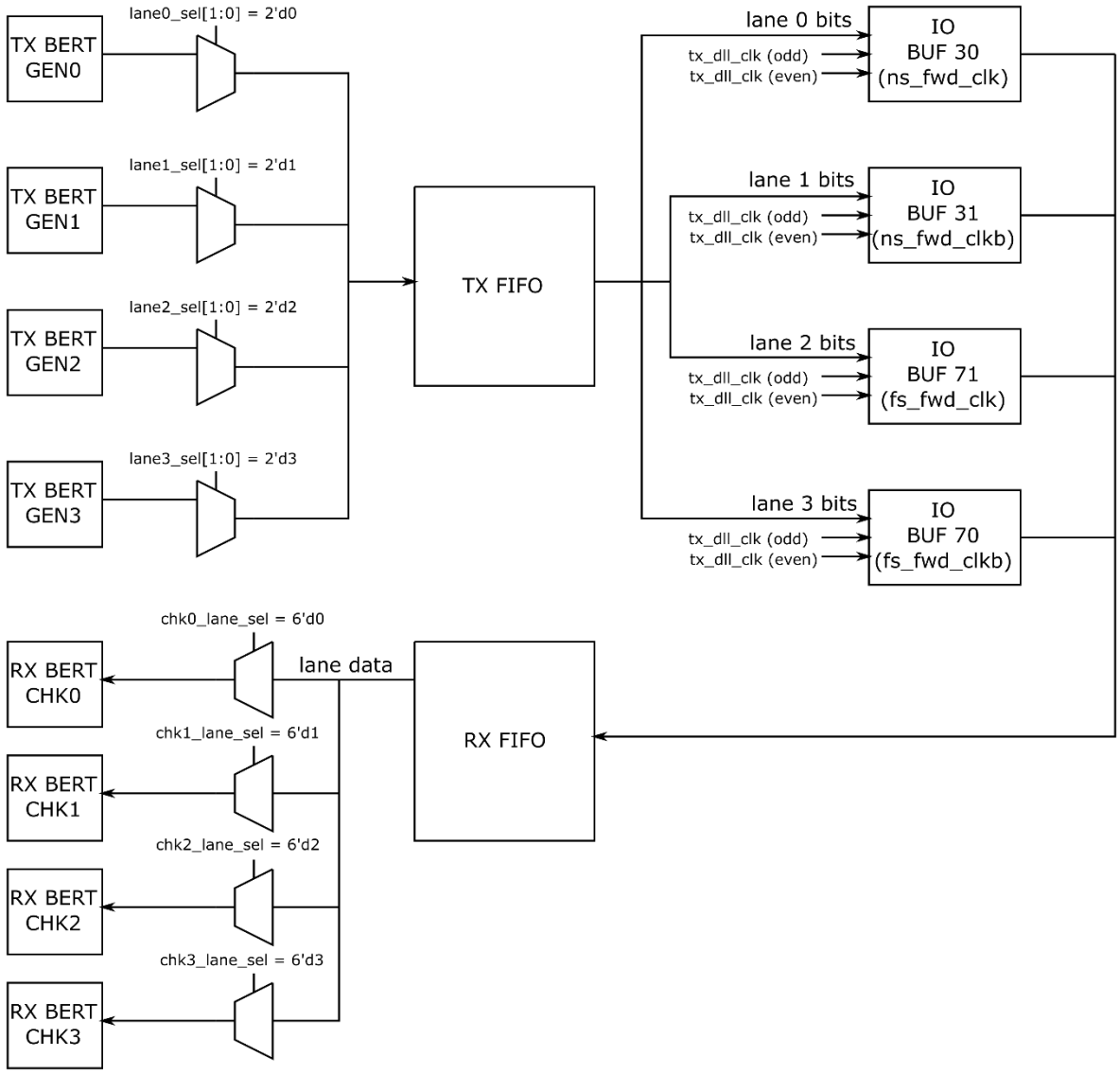

Figure 43: BERT logic used for forwarded clock test.

The following steps should be done to check forwarded clock paths:

1) During configuration phase, set the ***loopback_mode*** field of the ***txadpcfg_1*** register to 2'b10 to enable near side loopback.
2) During configuration phase, set the ***tx_bert_en*** bit of the ***txadpcfg_1*** register to select TX BERT data as input source of TX FIFO.
3) Wait for the end of configuration phase (CONF_DONE = 1) and calibration phase (all transfer enable signals are set by the hardware).

4) Set the ***fwd_clk_test*** field of the ***txadpcfg_1*** register to select TX BERT generator as data source to forwarded clock IO buffer and IO buffer output as data source to RX BERT checker.
5) Configure ***lane0_sel***, ***lane1_sel***, ***lane2_sel*** and ***lane3_sel*** fields of ***GEN_SEL0*** register with 2'd0, 2'd1, 2'd2 and 2'd3, respectively.
6) Configure ***chk0_lane_sel***, ***chk1_lane_sel***, ***chk2_lane_sel*** and ***chk3_lane_sel*** fields of ***RXLANE_SEL*** register with 6'd0, 6'd1, 6'd2 and 6'd3, respectively.
7) Configure **gen0_ptrn_sel**, **gen1_ptrn_sel**, **gen2_ptrn_sel**, **gen3_ptrn_sel**, **chk0_ptrn_sel**, **chk1_ptrn_sel**, **chk2_ptrn_sel** and **chk3_ptrn_sel** with 6'd5.
8) Set **seed_0_0** = **seed_1_0** = **seed_2_0** = **seed_3_0** = 0x5555_5555.
9) Set **seed_0_1** = **seed_1_1** = **seed_2_1** = **seed_3_1** = 0xAAAA_AAAA.
10) Set **seed_0_2** = **seed_1_2** = **seed_2_2** = **seed_3_2** = 0x5555_5555.
11) Set **seed_0_3** = **seed_1_3** = **seed_2_3** = **seed_3_3** = 0xAAAA_AAAA.
12) Write one into the bits of ***tx_start*** field in ***txbert_ctrl*** register that correspond to each TX BERT generator used.
13) Write one into the bits of ***seed_in*** field in ***rxbert_ctrl*** register for the used checkers. This procedure will initialize the RX BERT side with TX BERT seeds.
14) Wait for a period of time at least equal to 5100 *m_ns_fwd_clk* clock periods.
15) Write one into the bits of ***rx_start*** field in ***rxbert_ctrl*** register that correspond to each RX BERT checker used.
16) Read ***RXBERT0_ ERRCNT, RXBERT1_ ERRCNT, RXBERT2_ ERRCNT*** and ***RXBERT3_ ERRCNT*** registers using the procedure described in **Bit Error Rate Tester (BERT)** to check bit errors.
17) Read ***RXBERT0_ DATA0, RXBERT0_ DATA0, RXBERT0_ DATA0 and RXBERT0_ DATA0*** registers using the procedure described in **Bit Error Rate Tester (BERT)** to check if the expected data is being received in each RX BERT checker.

2.4 Redundancy

Redundancy storage is external to PHY and must be programmed into the PHY during configuration.

2.4.1 Active Redundancy

102 2-to-1 multiplexers each with an independent select bit.

2.4.2 Passive Redundancy

There is no passive redundancy in the PHY (AUX signals enter/leave on C4 bumps not micro bumps)

2.5 Weak pull-down and Weak pull-up

BCA has the following CSRs to control IO pads regarding weak pull-down and weak pull-up:

Register Name	Field name	Register Bit	Affected IOs [Bump ID]
io_ctrl1	tx_wkpu	31	Sets weak pull-up for IOs from 0 to 49.
	tx_wkpd	30	Sets weak pull-down for IOs from 0 to 49.
	rx_wkpu	29	Sets weak pull-up for IOs from 52 to 101.
	rx_wkpd	28	Sets weak pull-down for IOs from 52 to 101.

Table 38: Weak pull-down and weak-up control registers.

The hardware sets automatically weak pull-down in the following cases:

- All IO pads are in weak pull-down during power-on reset or during configuration phase (*conf_done* = 0).
- Spares IO pads are automatically configured in weak pull-down if there is no fault (redundancy is unused).
- In GEN2 mode, IO pads corresponding to the following clocks are automatically configured in weak pull-down: *ns_rcv_clk, ns_rcv_clk_b, ns_sr_clk_b, fs_sr_clk_b, fs_rcv_clk* and *fs_rcv_clk_b*.
- In Gen1 mode, unused IO pads are automatically configured in weak pull-down.
- A faulty or unused IO pad is automatically configured in weak-pull according to redundancy configuration.

2.6 DFT

Please, refer to AIB PHY integration guide to find information about DFT related ports.

2.6.1 JTAG Boundary Scan

Support only the AIB equivalent of EXTEST and INTEST

2.6.2 ATPG

Please, refer to AIB PHY integration guide.

3 Control and Status Registers

3.3 Address Map

Register Name	Offset Address	Access Type
Rx Adapter Configuration 0 (rxadpcfg_0)	0x208 + (channel_nb*0x800)	RW
Rx Adapter Configuration 1 (rxadpcfg_1)	0x210 + (channel_nb*0x800)	RW
Tx Adapter Configuration 0 (txadpcfg_0)	0x218 + (channel_nb*0x800)	RW
Tx Adapter Configuration 1 (txadpcfg_1)	0x21C + (channel_nb*0x800)	RW
BERT Access Request (bert_areq)	0x220 + (channel_nb*0x800)	RW
BERT Write data (bert_wdata)	0x224 + (channel_nb*0x800)	RW
TX BERT Control (txbert_ctrl)	0x228 + (channel_nb*0x800)	WO
TX BERT Status (txbert_sta)	0x22C + (channel_nb*0x800)	RO
TX BERT Read Data (txbert_rdata)	0x230 + (channel_nb*0x800)	RO
RX BERT Control (rxbert_ctrl)	0x234 + (channel_nb*0x800)	WO
RX BERT Status (rxbert_sta)	0x238 + (channel_nb*0x800)	RO
RX BERT Read Data (rxbert_rdata)	0x23C + (channel_nb*0x800)	RO
AIB Redundancy 0 (redund_0)	0x320 + (channel_nb*0x800)	RW
AIB Redundancy 1 (redund_1)	0x324 + (channel_nb*0x800)	RW
AIB Redundancy 2 (redund_2)	0x328 + (channel_nb*0x800)	RW
AIB Redundancy 3 (redund_3)	0x32C + (channel_nb*0x800)	RW
AIB Analog Control 1 (anactrl1)	0x330 + (channel_nb*0x800)	RW
AIB Analog Control 2 (anactrl2)	0x334 + (channel_nb*0x800)	RW
AIB Voltage Reference Code register (vrefcode)	0x338 + (channel_nb*0x800)	RW
AIB Calibration Voltage Reference register (calvref)	0x33C + (channel_nb*0x800)	RW
AIB Rx DLL1 (rxdll1)	0x340 + (channel_nb*0x800)	RW
AIB Rx DLL2 (rxdll2)	0x344 + (channel_nb*0x800)	RW
AIB CDR (cdr)	0x348 + (channel_nb*0x800)	RW
AIB Tx DLL1 (txdll1)	0x34C + (channel_nb*0x800)	RW
AIB Tx DLL2 (txdll2)	0x350 + (channel_nb*0x800)	RW
AIB Rx Clock (rxclk)	0x358 + (channel_nb*0x800)	RW
AIB DCS1 (dcs1)	0x364 + (channel_nb*0x800)	RW

AIB DCS2 (dcs2)	0x368 + (channel_nb*0x800)	RW
AIB IO control1 (io_ctrl1)	0x37C + (channel_nb*0x800)	RW
AIB IO control2 (io_ctrl2)	0x380 + (channel_nb*0x800)	RW
AIB FSMDIV (fsmdiv)	0x384 + (channel_nb*0x800)	RW
AIB RCOMP1 (rcomp1)	0x388 + (channel_nb*0x800)	RW
AIB RCOMP2 (rcomp2)	0x38C + (channel_nb*0x800)	RW
AIB NTL1 (ntl1)	0x390 + (channel_nb*0x800)	RW
AIB NTL2 (ntl2)	0x394 + (channel_nb*0x800)	RW
AIB ADC0 (adc0)	0xC000	RW
AIB ADC1 (adc1)	0xC004	RW
AIB ADC2 (adc2)	0xC008	RW
AIB ADC3 (adc3)	0xC00C	RW
AIB ADC4 (adc4)	0xC010	RW
AIB Auxiliary Channel (auxch)	0xC018	RW
AIB PVTA0 (pvta0)	0xC024	RW
AIB PVTB0 (pvtb0)	0xC028	RW
AIB PVTA1 (pvta1)	0xC034	RW
AIB PVTB1 (pvtb1)	0xC038	RW
AIB PVTA2 (pvta2)	0xC044	RW
AIB PVTB2 (pvtb2)	0xC048	RW
AIB PVTA3 (pvta3)	0xC054	RW
AIB PVTB3 (pvtb3)	0xC058	RW
AIB PVTA4 (pvta4)	0xC064	RW
AIB PVTB4 (pvtb4)	0xC068	RW

Table 39: Address Map (channel_nb from 0 to 23)

Note: All the registers are 32-bit wide and register space is 2KB for each AIB channel (0x000 to 0x4FF). The access types are:
RO – Read-only register: register bits are read-only and cannot be written by software.
RW – Read-write register: register bits can be read or write by software
Read access in reserved addresses return zero.
WO – Write-only register: register bits can be written, but read operation always returns zero.

3.4 Register Definitions

3.4.1Rx Adapter Configuration 0 (rxadpcfg_0)

Rx Adapter Configuration 0 (rxadpcfg_0) – 0x208 + (channel_nb*0x800)				
Bits	Name	Type	Reset	Description
31:28	reserved	RO	4′b0	Reserved.
27:24	rx_phcomp	RW	4′b010	Defines phase compensation to read receive FIFO: 4′b1100-1111: reserved. 4′b1011: 11 read cycle delay. 4′b1010: 10 read cycle delay. 4′b1001: 9 read cycle delay. 4′b1000: 8 read cycle delay. 4′b0111: 7 read cycle delay. 4′b0110: 6 read cycle delay. 4′b0101: 5 read cycle delay. 4′b0100: 4 read cycle delay. 4′b0011: 3 read cycle delay. 4′b0010: 2 read cycle delay. 4′b000x: reserved. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
23:4	reserved	RO	22′b0	Reserved.
3:2	rx_clk_div	RW	2′b0	Receive clock divider selection: 00: clock disabled. 01: clock frequency is divided by 1. 10: clock frequency is divided by 2. 11: clock frequency is divided by 4. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
1	rx_dbi_en	RW	1′b0	Enables Rx data bus inversion (DBI): 0: DBI disabled 1: DBI enabled **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).** **NOTE: When BERT is used, rx_dbi_en shall be disabled.**

0	rxswap_en	RW	1′b0	Enables Rx data swapping used in GEN 1.0 mode and FIFO 2x: 0: Data swapping disabled. 1: Data swapping enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).** **NOTE: When BERT is used, rxswap_en shall be disabled.**

Table 40: Rx Adapter Configuration 0

3.4.2 Rx Adapter Configuration 1 (rxadpcfg_1)

Rx Adapter Configuration 1 (rxadpcfg_1) – 0x210 + (channel_nb*0x800)				
Bits	Name	Type	Reset	Description
31	rx_wa_mode	WR	1′b0	This sticky bit defines the behavior of RX FIFO WAM state machine. Once the bit is set by the user, only a hard reset can clear it (write once bit). 0: Once word alignment is detected, *m_rx_align_done* is deasserted if word markers are not aligned. 1: Once word alignment is detected, *m_rx_align_done* keeps asserted if word markers are not aligned.
30:13	reserved	RO	18′b0	Reserved.

12:8	rx_align_threshold	RW	5′b10	Defines the minimum number of consecutive aligned word marks that should be detected before starting to receive data in Rx FIFO: Word mark threshold = rx_align_threshold + 1 Note: used only for FIFO 2x mode and 4x FIFO mode. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
7	rx_marker_bit79	RW	1′b0	Configures the marker at bit 79 (count in AIB2.0 full rate): 0: marker bit is not at bit 79. 1: marker bit is at bit 79. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
6	rx_marker_bit78	RW	1′b0	Configures the marker at bit 78 (count in AIB2.0 full rate): 0: marker bit is not at bit 78. 1: marker bit is at bit 78. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
5	rx_marker_bit77	RW	1′b0	Configures the marker at bit 77 (count in AIB2.0 full rate): 0: marker bit is not at bit 77. 1: marker bit is at bit 77. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
4	rx_marker_bit76	RW	1′b0	Configures the marker at bit 76 (count in AIB2.0 full rate): 0: marker bit is not at bit 76. 1: marker bit is at bit 76. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

3	rx_marker_bit39	RW	1′b0	Configures the marker at bit 39 (count in AIB2.0 used for GEN 1.0 mode): 0: marker bit is not at bit 39. 1: marker bit is at bit 39. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
2:1	rx_fifo_mode	RW	2′b0	Selected operation mode: 00: Receive FIFO 1x mode. 01: Receive FIFO 2x mode. 10: Receive FIFO 4x mode. 11: Receive register mode. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).** **NOTE: When BERT is used, rx_fifo_mode field shall be different from 11 (register mode).**
0	rx_wa_en	RW	1′b0	Enables receive word alignment for receive FIFO phase compensation: 0: Receive word alignment is disabled. 1: Receive word alignment is enabled. **NOTE: word alignment shall be disabled when BERT feature is used.**

Table 41: Rx Adapter Configuration 1

3.4.3 Tx Adapter Configuration 0 (txadpcfg_0)

Tx Adapter Configuration 0 (txadpcfg_0) – 0x218 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:28	tx_phcomp	RW	4′b010	Defines phase compensation to read transmit FIFO: 4′b1100-1111: reserved. 4′b1011: 11 read cycle delay. 4′b1010: 10 read cycle delay. 4′b1001: 9 read cycle delay. 4′b1000: 8 read cycle delay. 4′b0111: 7 read cycle delay. 4′b0110: 6 read cycle delay.

				4′b0101: 5 read cycle delay. 4′b0100: 4 read cycle delay. 4′b0011: 3 read cycle delay. 4′b0010: 2 read cycle delay. 4′b000x: reserved. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
27:26	reserved	RO	2′b0	Reserved.
25:24	tx_clk_div	RW	2′b0	Transmit clock divider selection: 00: clock is disabled. 01: clock frequency is divided by 1. 10: clock frequency is divided by 2. 11: clock frequency is divided by 4. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
23	tx_wm_en	RW	1′b0	Enables transmit word alignment marker for transmit FIFO phase compensation: 0: Word marker is disabled for transmission. 1: Word marker is enabled for transmission. **NOTE: word alignment marker shall be disabled when BERT feature is used.**
22:21	tx_fifo_mode	RW	2′b0	Selected operation mode: 00: Transmit FIFO 1x mode. 01: Transmit FIFO 2x mode. 10: Transmit FIFO 4x mode. 11: Transmit register mode. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).** **NOTE: When BERT is used, tx_fifo_mode field shall be different from 11 (register mode).**
20	tx_marker_bit79	RW	1′b0	Inserts the marker at bit 79 (count in AIB2.0 full rate): 0: marker bit is not at bit 79. 1: marker bit is at bit 79. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

19	tx_marker_bit78	RW	1′b0	Inserts the marker at bit 78 (count in AIB2.0 full rate): 0: marker bit is not at bit 78. 1: marker bit is at bit 78. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
18	tx_marker_bit77	RW	1′b0	Inserts the marker at bit 77 (count in AIB2.0 full rate): 0: marker bit is not at bit 77. 1: marker bit is at bit 77. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
17	tx_marker_bit76	RW	1′b0	Inserts the marker at bit 76 (count in AIB2.0 full rate): 0: marker bit is not at bit 76. 1: marker bit is at bit 76. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
16	tx_marker_bit39	RW	1′b0	Inserts the marker at bit 39 (count in AIB2.0 used for GEN 1.0 mode): 0: marker bit is not at bit 39. 1: marker bit is at bit 39. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
15:2	Reserved	RO	14′b0	Reserved.
1	tx_dbi_en	RW	1′b0	Enables Tx data bus inversion (DBI): 0: DBI disabled 1: DBI enabled **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).** **NOTE: When BERT is used, tx_dbi_en shall be disabled.**
0	txswap_en	RW	1′b0	Enables Tx data swapping used in GEN 1.0 mode and FIFO 2x: 0: Data swapping disabled. 1: Data swapping enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated). **NOTE: When BERT is used, txswap_en shall be disabled.**

Table 42: Tx Adapter Configuration 0

3.4.4 Tx Adapter Configuration 1 (txadpcfg_1)

Tx Adapter Configuration 1 (tx1) – 0x21C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	sdr_mode	RW	1'b0	Enables single data rate mode: 0: PHY operates in double data rate mode. 1: PHY operates in single data rate mode. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
30	pad_en	RW	1'b1	Enables IO pads: 0: disables IO pads (high impedance). 1: enables IO pads. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
29:16	reserved	RO	15'b0	Reserved.
15:14	loopback_mode	RW	2'b0	Configures loop back mode: 00: no loop back 01: Far side loop back 10: Near side loop back 11: Reserved **NOTE**: ***rx_clk_div*** **and** ***tx_clk_div*** **fields should be configured with the same value when the PHY is operating in far side loopback mode.** **NOTE: PHY shall be configured as master when operating in near side loopback.** **NOTE: For near side loopback (loopback_mode = 2'b10), Transmit data is driven in both TX pads and RX pads, and data is read in TX pads.**

13:10	Reserved	RO	4′b0	Reserved.
9	fwd_clk_test	RW	1′b0	Enables the use of BERT logic to test clock paths of ns_fwd_clk, ns_fwd_clkb, fs_fwd_clk and fs_fwd_clkb. **NOTE: Used Only for test purpose. Near side loop back and tx_bert_en shall be set to use this feature.**
8	tx_bert_en	RW	1′b0	Enables the selection of TX BERT data on transmit FIFO data path. **NOTE: If tx_bert_en is set, loopback_mode field shall be different from 2′b01 (Far side loopback).** **NOTE: BERT feature does not support register mode.**
7:0	Reserved	RO	8′b0	Reserved.

Table 43: Tx Adapter Configuration 1

3.4.5 BERT Access Request (bert_areq)

BERT Access Request Register (bert_areq) – 0x220 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	acc_rq_p	RO	1′b0	Indicates BERT register access is pending.
30	acc_req	WO	1′b0	Access request for registers in BERT domain. The acc_rdwr and acc_addr fields should be set together acc_req. It means the hardware only captures the values of acc_rdwr and acc_addr fields during the host access whose acc_req bit is being set. **NOTE: acc_req bit must not be set when there is an access pending (acc_rq_p = 1).**
29	acc_rdwr	WO	1′b0	Access read/write control bit: 0: write access 1: read access
28:6	Reserved	RO	9′b0	Reserved.
5:0	acc_addr	WO	6′b0	Address used for accessing register in BERT domain.

Table 44: BERT Access register

3.4.6 BERT Write data (bert_wdata)

BERT Write Data Register (bert_wdata) – 0x224 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:0	wdata	RW	32'b0	Write data for BERT domain register access. **NOTE: wdata register must not be written when there is an access pending (acc_rq_p = 1).**

Table 45: BERT Write Data register

3.4.7 **TX BERT Control (txbert_ctrl)**

TX BERT Control Register (txbert_ctrl) – 0x228 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:20	Reserved	RO	12'b0	Reserved.
19:16	tx_start	WO	4'b0	Writing one starts transmission of TX BERT bit sequence: tx_start[0] starts transmission of BERT transmitter 0 tx_start[1] starts transmission of BERT transmitter 1 tx_start[2] starts transmission of BERT transmitter 2 tx_start[3] starts transmission of BERT transmitter 3
15:4	Reserved	RO	12'b0	Reserved.
3:0	tx_rst	WO	4'b0	Writing one resets TX BERT sequence generator block and stops bit sequence transmission: tx_rst[0] resets BERT transmitter 0. tx_rst[1] resets BERT transmitter 1. tx_rst[2] resets BERT transmitter 2. tx_rst[3] resets BERT transmitter 3. **NOTE: seedp0_0, seedp1_0, seedp2_0, seedp3_0, seedp0_1, seedp1_1, seedp2_1, seedp3_1, seedp0_2, seedp1_2, seedp2_2, seedp3_2, seedp0_3, seedp1_3, seedp2_3, seedp3_3 BERT and txbert_run register fields are affected by rx_rst.**

Table 46: TX BERT Control register.

3.4.8 **TX BERT Status (txbert_sta)**

TX BERT Status Register (txbert_sta) – 0x22C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	Reserved	RO	1'b1	Reserved.
30:4	Reserved	RO	27'b0	Reserved.

3:0	txbert_run	RO	4′b0	When set by the hardware, indicates which TX BERT sequence generator is running: txbert_run[0]: Indicates if Tx BERT0 generator is running. txbert_run[1]: Indicates if Tx BERT1 generator is running. txbert_run[2]: Indicates if Tx BERT2 generator is running. txbert_run[3]: Indicates if Tx BERT3 generator is running.

Table 47: TX BERT Status register.

3.4.9 **TX BERT Read Data (txbert_rdata)**

TX BERT Read Data Register (txbert_rdata) – 0x230 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:0	tx_rdata	RO	32′b0	Read data for register access in TX BERT domain. **NOTE: tx_rdata register must be read only when there is no access pending (acc_rq_p = 0).**

Table 48: TX BERT Read Data register.

3.4.10 **RX BERT Control (rxbert_ctrl)**

RX BERT Control Register (rxbert_ctrl) – 0x234 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:28	Reserved	RO	4′b0	Reserved.
27:24	seed_in	WO	4′b0	Writing one enables self seeding for RX PRBS checkers: seed_in[0] starts seeding Rx BERT 0 checker. seed_in[1] starts seeding Rx BERT 1 checker. seed_in[2] starts seeding Rx BERT 2 checker. seed_in[3] starts seeding Rx BERT 3 checker.
23:20	Reserved	RO	4′b0	Reserved.
19:16	rx_start	WO	4′b0	Writing one starts transmission of TX BERT bit sequence: rx_start[0] starts bit comparison on Rx BERT0 checker. rx_start[1] starts bit comparison on Rx BERT1 checker. rx_start[2] starts bit comparison on Rx BERT2 checker. rx_start[3] starts bit comparison on Rx BERT3 checker.

15:4	Reserved	RO	12′b0	Reserved.
3:0	rx_rst	WO	4′b0	Writing one resets RX BERT sequence checker block and disables bit sequence check: rx_rst[0] resets Rx BERT0 checker. rx_rst[1] resets Rx BERT1 checker. rx_rst[2] resets Rx BERT2 checker. rx_rst[3] resets Rx BERT3 checker. **NOTE: rxbert_run, biterr, rxbert0_bitcnt_lo, rxbert0_bitcnt_hi, rxbert0_errcnt, rxbert1_errcnt, rxbert2_errcnt, rxbert3_errcnt, rx_bert0_data0, rx_bert0_data1, rx_bert0_data2, rx_bert0_data3, rx_bert1_data0, rx_bert1_data1, rx_bert1_data2, rx_bert1_data3, rx_bert2_data0, rx_bert2_data1, rx_bert2_data2, rx_bert2_data3, rx_bert3_data0, rx_bert3_data1, rx_bert3_data2 and rx_bert3_data3 register fields are affected by rx_rst.**

Table 49: RX BERT Control register

3.4.11 RX BERT Status (rxbert_sta)

RX BERT Status Register (rxbert_sta) – 0x238 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:20	Reserved	RO	12′b0	Reserved.
19:16	biterr	RO	4′b0	When set by the hardware, indicates at least one bit error has been detected in RX BERT checkers: biterr[0] bit error detected in RX BERT0 checker. biterr[1] bit error detected in RX BERT1 checker. biterr[2] bit error detected in RX BERT2 checker. biterr[3] bit error detected in RX BERT3 checker.
15:4	Reserved	RO	12′b0	Reserved.
3:0	rxbert_run	RO	4′b0	When set by the hardware, indicates which RX BERT checker is checking the receive bits: txbert_run[0]: Indicates if RX BERT0 checker is running. txbert_run[1]: Indicates if RX BERT1 checker is running. txbert_run[2]: Indicates if RX BERT2 checker is running. txbert_run[3]: Indicates if RX BERT3 checker is running.

Table 50: RX BERT Status register.

3.4.12 **RX BERT Read Data (rxbert_rdata)**

RX BERT Read Data 0 Register (rxbert_rdata1) – 0x23C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:0	rx_rdata	RO	32'b0	Read data for register access in RX BERT domain. **NOTE: rx_rdata register must be read only when there is no access pending (acc_rq_p = 0).**

Table 51: RX BERT Read Data0 register.

3.4.13 **AIB Redundancy 0 (redund_0)**

More details about redundancy can be found in **Redundancy** section.

AIB Redundancy 0 (redund_0) – 0x320 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	redund_31	RW	1'b0	redundancy enable for AIB IO pad 31: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
30	redund_30	RW	1'b0	redundancy enable for AIB IO pad 30: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
29	redund_29	RW	1'b0	redundancy enable for AIB IO pad 29: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
28	redund_28	RW	1'b0	redundancy enable for AIB IO pad 28: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
27	redund_27	RW	1'b0	redundancy enable for AIB IO pad 27: 0: redundancy shift disabled.

				1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
26	redund_26	RW	1'b0	redundancy enable for AIB IO pad 26: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
25	redund_25	RW	1'b0	redundancy enable for AIB IO pad 25: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
24	redund_24	RW	1'b0	redundancy enable for AIB IO pad 24: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
23	redund_23	RW	1'b0	redundancy enable for AIB IO pad 23: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
22	redund_22	RW	1'b0	redundancy enable for AIB IO pad 22: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
21	redund_21	RW	1'b0	redundancy enable for AIB IO pad 21: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
20	redund_20	RW	1'b0	redundancy enable for AIB IO pad 20: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

19	redund_19	RW	1'b0	redundancy enable for AIB IO pad 19: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
18	redund_18	RW	1'b0	redundancy enable for AIB IO pad 18: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
17	redund_17	RW	1'b0	redundancy enable for AIB IO pad 17: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
16	redund_16	RW	1'b0	redundancy enable for AIB IO pad 16: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
15	redund_15	RW	1'b0	redundancy enable for AIB IO pad 15: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
14	redund_14	RW	1'b0	redundancy enable for AIB IO pad 14: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
13	redund_13	RW	1'b0	redundancy enable for AIB IO pad 13: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
12	redund_12	RW	1'b0	redundancy enable for AIB IO pad 12: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
11	redund_11	RW	1'b0	redundancy enable for AIB IO pad 11: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
10	redund_10	RW	1'b0	redundancy enable for AIB IO pad 10: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
9	redund_9	RW	1'b0	redundancy enable for AIB IO pad 9: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
8	redund_8	RW	1'b0	redundancy enable for AIB IO pad 8: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
7	redund_7	RW	1'b0	redundancy enable for AIB IO pad 7: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
6	redund_6	RW	1'b0	redundancy enable for AIB IO pad 6: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
5	redund_5	RW	1'b0	redundancy enable for AIB IO pad 5: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
4	redund_4	RW	1'b0	redundancy enable for AIB IO pad 4:

				0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
3	redund_3	RW	1'b0	redundancy enable for AIB IO pad 3: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
2	redund_2	RW	1'b0	redundancy enable for AIB IO pad 2: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
1	redund_1	RW	1'b0	redundancy enable for AIB IO pad 1: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
0	redund_0	RW	1'b0	redundancy enable for AIB IO pad 0: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

Table 52: AIB Redundancy 0

3.4.14 AIB Redundancy 1 (redund_1)

More details about redundancy can be found in **Redundancy** section.

AIB Redundancy 1 (redund_1) – 0x324 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	redund_63	RW	1'b0	redundancy enable for AIB IO pad 63: 0: redundancy shift disabled. 1: redundancy shift enabled.
30	redund_62	RW	1'b0	redundancy enable for AIB IO pad 62:

				0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
29	redund_61	RW	1'b0	redundancy enable for AIB IO pad 61: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
28	redund_60	RW	1'b0	redundancy enable for AIB IO pad 60: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
27	redund_59	RW	1'b0	redundancy enable for AIB IO pad 59: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
26	redund_58	RW	1'b0	redundancy enable for AIB IO pad 58: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
25	redund_57	RW	1'b0	redundancy enable for AIB IO pad 57: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
24	redund_56	RW	1'b0	redundancy enable for AIB IO pad 56: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
23	redund_55	RW	1'b0	redundancy enable for AIB IO pad 55: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
22	redund_54	RW	1'b0	redundancy enable for AIB IO pad 54: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
21	redund_53	RW	1'b0	redundancy enable for AIB IO pad 53: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
20	redund_52	RW	1'b0	redundancy enable for AIB IO pad 52: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
19	reserved	RO	1'b0	Reserved
18	reserved	RO	1'b0	Reserved
17	redund_49	RW	1'b0	redundancy enable for AIB IO pad 49: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
16	redund_48	RW	1'b0	redundancy enable for AIB IO pad 48: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
15	redund_47	RW	1'b0	redundancy enable for AIB IO pad 47: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
14	redund_46	RW	1'b0	redundancy enable for AIB IO pad 46: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
13	redund_45	RW	1'b0	redundancy enable for AIB IO pad 45: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
12	redund_44	RW	1'b0	redundancy enable for AIB IO pad 44: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
11	redund_43	RW	1'b0	redundancy enable for AIB IO pad 43: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
10	redund_42	RW	1'b0	redundancy enable for AIB IO pad 42: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
9	redund_41	RW	1'b0	redundancy enable for AIB IO pad 41: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
8	redund_40	RW	1'b0	redundancy enable for AIB IO pad 40: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
7	redund_39	RW	1'b0	redundancy enable for AIB IO pad 39: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
6	redund_38	RW	1'b0	redundancy enable for AIB IO pad 38:

				0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
5	redund_37	RW	1'b0	redundancy enable for AIB IO pad 37: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
4	redund_36	RW	1'b0	redundancy enable for AIB IO pad 36: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
3	redund_35	RW	1'b0	redundancy enable for AIB IO pad 35: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
2	redund_34	RW	1'b0	redundancy enable for AIB IO pad 34: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
1	redund_33	RW	1'b0	redundancy enable for AIB IO pad 33: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
0	redund_32	RW	1'b0	redundancy enable for AIB IO pad 32: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

Table 53: AIB Redundancy 1

3.4.15 **AIB Redundancy 2 (redund_2)**

More details about redundancy can be found in **Redundancy** section.

AIB Redundancy 2 (redund_2) – 0x328 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	redund_95	RW	1'b0	redundancy enable for AIB IO pad 95: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
30	redund_94	RW	1'b0	redundancy enable for AIB IO pad 94: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
29	redund_93	RW	1'b0	redundancy enable for AIB IO pad 93: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
28	redund_92	RW	1'b0	redundancy enable for AIB IO pad 92: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
27	redund_91	RW	1'b0	redundancy enable for AIB IO pad 91: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
26	redund_90	RW	1'b0	redundancy enable for AIB IO pad 90: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
25	redund_89	RW	1'b0	redundancy enable for AIB IO pad 89: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
24	redund_88	RW	1'b0	redundancy enable for AIB IO pad 88: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
23	redund_87	RW	1'b0	redundancy enable for AIB IO pad 87: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
22	redund_86	RW	1'b0	redundancy enable for AIB IO pad 86: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
21	redund_85	RW	1'b0	redundancy enable for AIB IO pad 85: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
20	redund_84	RW	1'b0	redundancy enable for AIB IO pad 84 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
19	redund_83	RW	1'b0	redundancy enable for AIB IO pad 83: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
18	redund_82	RW	1'b0	redundancy enable for AIB IO pad 82: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
17	redund_81	RW	1'b0	redundancy enable for AIB IO pad 81: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
16	redund_80	RW	1'b0	redundancy enable for AIB IO pad 80: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
15	redund_79	RW	1'b0	redundancy enable for AIB IO pad 79: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
14	redund_78	RW	1'b0	redundancy enable for AIB IO pad 78: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
13	redund_77	RW	1'b0	redundancy enable for AIB IO pad 77: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
12	redund_76	RW	1'b0	redundancy enable for AIB IO pad 76: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
11	redund_75	RW	1'b0	redundancy enable for AIB IO pad 75: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
10	redund_74	RW	1'b0	redundancy enable for AIB IO pad 74: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
9	redund_73	RW	1'b0	redundancy enable for AIB IO pad 73: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
8	redund_72	RW	1'b0	redundancy enable for AIB IO pad 72: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
7	redund_71	RW	1'b0	redundancy enable for AIB IO pad 71: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
6	redund_70	RW	1'b0	redundancy enable for AIB IO pad 70: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
5	redund_69	RW	1'b0	redundancy enable for AIB IO pad 69: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
4	redund_68	RW	1'b0	redundancy enable for AIB IO pad 68: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
3	redund_67	RW	1'b0	redundancy enable for AIB IO pad 67: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
2	redund_66	RW	1'b0	redundancy enable for AIB IO pad 66: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
1	redund_65	RW	1'b0	redundancy enable for AIB IO pad 65: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
0	redund_64	RW	1'b0	redundancy enable for AIB IO pad 64: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

Table 54: AIB Redundancy 2

3.4.16 AIB Redundancy 3 (redund_3)

More details about redundancy can be found in **Redundancy** section.

AIB Redundancy 3 (redund_3) – 0x32C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:6	reserved	RO	26'b0	Reserved.
5	redund_101	RW	1'b0	redundancy enable for AIB IO pad 101: 0: redundancy shift disabled. 1: redundancy shift enabled.

				NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).
4	redund_100	RW	1'b0	redundancy enable for AIB IO pad 100: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
3	redund_99	RW	1'b0	redundancy enable for AIB IO pad 99: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
2	redund_98	RW	1'b0	redundancy enable for AIB IO pad 98: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
1	redund_97	RW	1'b0	redundancy enable for AIB IO pad 97: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**
0	redund_96	RW	1'b0	redundancy enable for AIB IO pad 96: 0: redundancy shift disabled. 1: redundancy shift enabled. **NOTE: This field shall be written only during configuration phase (When i_conf_done is negated).**

Table 55: AIB Redundancy 3

3.4.17 AIB Analog Control 1 (anactrl1)

AIB Analog Control1 register (anactrl) – 0x330 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:16	digmon	RW	16'h0	Digital monitor code selection.
15:14	reserved	RO	2'h0	Reserved
13:12	dll_avg_sel	RW	2'h1	DLL average selection

<table>
<tr><td>11</td><td>io_anamon_en</td><td>RW</td><td>1′b0</td><td>Enables analog monitor.</td></tr>
<tr><td>10</td><td>digmon_en</td><td>RW</td><td>1′b0</td><td>Enables digital monitor.</td></tr>
<tr><td>9</td><td>rx_lpbk_en</td><td>RW</td><td>1′b0</td><td>Enable RX loopback in IO cells.
NOTE: The loopback_mode field of txadpcfg_1 register shall be configured as 2′b00, when rx_lpbk_en is set.</td></tr>
<tr><td>8</td><td>tx_lpbk_en</td><td>RW</td><td>1′b0</td><td>Enable TX loopback in IO cells.
NOTE: The loopback_mode field of txadpcfg_1 register shall be configured as 2′b00, when tx_lpbk_en is set.</td></tr>
<tr><td>7:3</td><td>reserved</td><td>RO</td><td>5′h0</td><td>Reserved</td></tr>
<tr><td>2:0</td><td>pll_freq</td><td>RW</td><td>3′h2</td><td>Configure PHY custom implementation to operate according to PLL clock frequency:
<table><tr><td>Frequency range</td><td>pll_frep[2:0]</td></tr><tr><td>reserved</td><td>3′b000- 3′b001</td></tr><tr><td>2.1GHz to 1.7GHz</td><td>3′b010</td></tr><tr><td>1.7GHz to 1.4GHz</td><td>3′b011</td></tr><tr><td>1.4GHz to 1.1GHz</td><td>3′b100</td></tr><tr><td>1.1GHz to 850MHz</td><td>3′b101</td></tr><tr><td>850MHz to 650MHz</td><td>3′b110</td></tr><tr><td>650MHz to 500MHz</td><td>3′b111</td></tr></table></td></tr>
</table>

Table 56: AIB Analog Control 1 register

3.4.18 AIB Analog Control 2 (anactrl2)

AIB Analog Control1 register (anactrl2) – 0x334 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:3	reserved	RO	29′h0	Reserved
2:0	anamon	RW	3′h0	Analog monitor code selection.

Table 57: AIB Analog Control 2 register.

3.4.19 AIB Voltage Reference Code register (vrefcode)

AIB Voltage Reference Code register (vrefcode) – 0x338 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	reserved	RO	1'h0	Reserved
30:24	vrefn_gen2	RW	7'h40	Voltage Reference N code in GEN2 mode
23	reserved	RO	1'h0	Reserved
22:16	vrefp_gen2	RW	7'h40	Voltage Reference P code in GEN2 mode
15	reserved	RO	1'h0	Reserved
14:8	vrefn_gen1	RW	7'h7F	Voltage Reference N code in GEN1 mode
7	reserved	RO	1'h0	Reserved
6:0	vrefp_gen1	RW	7'h7F	Voltage Reference P code in GEN1 mode

Table 58: AIB Voltage Reference Code register

3.4.20 AIB Calibration Voltage Reference register (calvref)

AIB Calibration Voltage Reference register (vrefcode) – 0x33C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	vcal_done	RO	1'b0	Indicates voltage offset calibration is done.
30	vcalcode_ovrd	RW	1'b0	When set, overrides internal voltage offset calibration code by *calvcode* field.
29	vcaldone_ovrd	RW	1'b0	When set, overrides internal voltage offset calibration done indication.
28:5	reserved	RO	24'h0	Reserved
4:0	calvcode	RW	5'h0	Calibration voltage reference code.

Table 59: AIB Calibration Voltage Reference register.

3.4.21 AIB Rx DLL1 (rxdll1)

AIB Rx DLL1 register (rxdll1) – 0x340 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description

31:24	reserved	RO	8'h0	Reserved.
23:22	rxdll_lockctrl_code	RW	2'h0	Decides number of cycles to be considered for locking.
21:20	rxdll_clk_sel[1:0]	RW	2'h0	Selects the DLL input clock. 00: Far side clock 01: DLL test clock 10: Calibration clock 11: JTAG clock
19:12	reserved	RO	8'h0	Reserved.
11:8	rxdll_lockthr_code	RW	4'h0	Decides pulse width to be considered for locking.
7:3	reserved	RO	5'h0	Reserved.
2:0	rxdll_cdrctrl_code	RW	3'h0	Controls CDR loop.

Table 60: AIB RX DLL1 register.

3.4.22 AIB Rx DLL2 (rxdll2)

Please note that withing the rxdll2 register the following fields are swapped within the RTL itself. The names as listed within the specification are correct but the names within the RTL have been swapped. The following table lists the fields that do not match.

rxpi_aclk_code_ovrd	**rxpi_sclk_code_ovrd**
rxadp_lock_ovrd	**rxsoc_lock_ovrd**
rx_soc_clk_lock	**rx_adp_clk_lock**
rx_adp_clkph_code	**rx_soc_clkph_code**

Table 61 Swapped fields in RTL

AIB Rx DLL2 register (rxdll2) – 0x344 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	rxpi_aclk_code_ovrd	RW	1'h0	Overrides PI adapter code with pi_adpclk_code
30	rxpi_sclk_code_ovrd	RW	1'h0	Overrides PI SoC code with pi_socclk_code
29	rxadp_lock_ovrd	RW	1'h0	Overrides internal Rx adapt clock lock.
28	rxsoc_lock_ovrd	RW	1'h0	Overrides internal Rx SoC clock lock.
27	rx_soc_clk_lock	RO	1'h0	Indicates when Rx SoC clock is locked:

				0: Rx SoC clock is unlocked 1: Rx SoC clock is locked
26	rx_adp_clk_lock	RO	1'h0	Indicates when Rx adapter clock is locked: 0: Rx adapter clock is unlocked 1: Rx adapter clock is locked
25:24	reserved	RO	4'h0	Reserved
23:20	rxpi_adpclk_code	RW	4'h0	PI adapter clock code
19:16	rxpi_socclk_code	RW	4'h0	PI SoC clock code
15:12	rx_adp_clkph_code	RO	4'h0	Rx adapter clock phase code
11:8	rx_soc_clkph_code	RO	4'h0	Rx SoC clock phase code
7:5	reserved	RO	3'h0	Reserved
4:0	rxdll_digview	RW	5'h0	RX DLL digital view.

Table 62: AIB RX DLL2 register.

3.4.23 **AIB CDR (cdr)**

AIB CDR register (cdr) – 0x348 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	picode_up	RW	1'h0	Active high signal for updating PI codes. One code update takes place for every transition from low to high. This needs to be pulled down and then pulled up for next code update.
30	cdr_ovrd_sel	RW	1'h0	Overrides internal even code and internal odd code selections.
29	cdr_lock_ovrd	RW	1'h0	Overrides internal calibration lock
28	cdr_lock	RO	1'h0	Indicates when CDR is locked: 0: CDR is unlocked. 1: CDR is locked.
27:18	reserved	RO	10'h0	Reserved
17:16	cdr_clk_sel	RW	2'h0	Calibration clock selection
15	reserved	RO	18'h0	Reserved
14:8	picode_odd	RW	7'h0	Code for odd clock calibration.
7	reserved	RO	1'h0	Reserved
6:0	picode_even	RW	7'h0	Code for even clock calibration.

Table 63: AIB RX DLL3 register.

3.4.24 AIB Tx DLL1 (txdll1)

AIB Tx DLL1 register (txdll1) – 0x34C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:30	reserved	RO	2′h0	Reserved
29:28	txlockctrl_code	RW	2′h0	Decides number of cycles to be considered for locking
27:26	reserved	RO	2′h0	Reserved
25:24	txclkin_sel	RW	2′h0	Select the DLL input clock
23:12	reserved	RO	12′h0	Reserved
11:8	txpi_aclk_code	RW	4′h0	PI adapter clock code
7:4	txlockthr_code	RW	4′h0	Decides pulse width to be considered for locking
3:0	reserved	RO	4′h0	Reserved

Table 64: AIB TX DLL1 register.

3.4.25 AIB Tx DLL2 (txdll2)

AIB Tx DLL2 register (txdll2) – 0x350 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	txpi_aclk_code_ovrd	RW	1′h0	Overrides PI adapter code with pi_adpclk_code
30:29	reserved	RO	2′h0	Reserved.
28	txpi_sclk_code_ovrd	RW	1′h0	Overrides PI SoC code with pi_adpclk_code
27	txadp_lock_ovrd	RW	1′h0	Overrides internal TX DLL lock for TX adapt clock.
26	txsoc_lock_ovrd	RW	1′h0	Overrides internal TX DLL lock for TX SoC clock.
25	tx_adp_clk_lock	RO	1′h0	Tx adapter clock phase lock indication: 0: Tx adapter clock phase is unlocked 1: Tx adapter clock phase is locked

24	tx_soc_clk_lock	RO	1'h0	Tx SoC clock phase lock indication: 0: Tx SoC clock phase is unlocked 1: Tx SoC clock phase is locked
23:20	tx_adp_clkph_code	RO	4'h0	Indicates Tx adapter clock phase code.
19:16	tx_soc_clkph_code	RO	4'h0	Indicates Tx Soc clock phase code.
15:13	reserved	RO	3'h0	Reserved.
12:8	txdll_digview_sel	RW	5'h0	Digital output selector for testing (active high).
7:4	reserved	RO	4'h0	Reserved.
3:0	txpi_socclk_code	RW	4'h0	PI SoC clock code

Table 65: AIB TX DLL2 register.

3.4.26 AIB Rx Clock (rxclk)

AIB Rx Clock register (rxclk) – 0x358 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	rx_dcc_lock	RO	1'b0	Indicates RX DC is locked: 0: RX DCC is unlocked. 1: RX DCC locked.
30:27	reserved	RO	4'h0	Reserved
26:24	dcc_bias_code	RW	3'h4	Controls the mirror ratio for bias
23:7	reserved	RO	17'h0	Reserved
6:4	bias_nored	RW	3'h4	Controls bias when redundancy is disabled.
3	reserved	RO	1'h0	Reserved
2:0	bias_red	RW	3'h4	Controls bias when redundancy is enabled.

Table 66: AIB RX Clock register

3.4.27 AIB DCS1 (dcs1)

AIB DCS1 register (dcs1) – 0x364 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description

31	dcssel_ovrd	RW	1'h0	DCS internal selections are overridden by the register fields in DCS2 register.
30	dcs_lock_ovrd	RW	1'h0	DCS internal lock is overridden.
29	dcs_chopen_ovrd	RW	1'h0	DCS internal chopen signal is overridden.
28	dcs_lock	RO	1'h0	Indicates when DCS is locked: 0: DCS is unlocked 1: DCS is locked
27:5	reserved	RO	23'h0	Reserved
4	dcs_single_ended	RW	1'h0	Select clock configuration single ended or differential
3:2	dcs_cbb_clkdiv	RW	2'h0	Selects the ADC input clock divider
1	dcs_sel_clkp	RW	1'h0	Selects clock configuration single ended p or differential
0	dcs_sel_clkn	RW	1'h0	Selects clock configuration single ended n or differential

Table 67: AIB DCS register.

3.4.28 **AIB DCS2 (dcs2)**

AIB DCS2 register (dcs2) – 0x368 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:29	reserved	RO	3'h0	Reserved
28:24	dcs2_npusel_code	RW	5'h0	N pull up code
23:21	reserved	RO	3'h0	Reserved
20:16	dcs2_npdsel_code	RW	5'h0	N pull down code
15:13	reserved	RO	3'h0	Reserved
12:8	dcs2_ppusel_code	RW	5'h0	P pull up code
7:5	reserved	RO	3'h0	Reserved
4:0	dcs2_ppdsel_code	RW	5'h0	P pull down code

Table 68: AIB DCS register.

3.4.29 AIB IO control1 (io_ctrl1)

AIB IO control1 register (io_ctrl1) – 0x37C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	tx_wkpu	RW	1'b0	Enables weak pull-up on TX IOs: 0: weak pull-up is disabled. 1: weak pull-up is enabled.
30	tx_wkpd	RW	1'b0	Enables weak pull-down on TX IOs: 0: weak pull-down is disabled. 1: weak pull-down is enabled.
29	rx_wkpu	RW	1'b0	Enables weak pull-up on RX IOs: 0: weak pull-up is disabled. 1: weak pull-up is enabled.
28	rx_wkpd	RW	1'b0	Enables weak pull-down on RX IOs: 0: weak pull-down is disabled. 1: weak pull-down is enabled.
27:24	reserved	RO	8'b0	Reserved.
23:16	txppu_code	RW	8'h0	TX P-pullup code
15:8	txnpu_code	RW	8'h0	TX N-pullup code
7:0	txnpd_code	RW	8'h0	TX N-pulldown code

Table 69: AIB IO control1 register.

3.4.30 AIB IO Control2 register (io_ctrl2)

AIB IO control2 register (io_ctrl2) – 0x380 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	rx_deskew_en	RW	1'b1	Enables receive deskew
30	rx_deskew_step	RW	1'b1	Receive deskew step
29	rx_deskew_ovrd	RW	1'b0	Overrides internal Rx deskew value with *rx_deskew* field.

28	reserved	RO	1'b0	Reserved.
27:24	rx_deskew	RW	4'b0	RX deskew
23	tx_deskew_en	RW	1'b1	Enables transmit deskew
22	tx_deskew_step	RW	1'b1	Transmit deskew step
21	tx_deskew_ovrd	RW	1'b0	Overrides internal Tx deskew value with *tx_deskew* field.
20	reserved	RO	1'b0	Reserved.
19:16	tx_deskew	RW	4'b0	TX deskew
15:0	reserved	RO	16'h0	Reserved

Table 70: AIB IO Control2 register.

3.4.31 AIB FSM clock divider register (fsm_div)

AIB FSM clock divider register (fsm_div) – 0x384 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	reserved	RO	1'h0	Reserved.
30:27	dcs_fsm_div	RW	4'h7	DCS state machine clock divider: 0000: clock gated 0001: clock divided by 1 0010: clock divided by 2 0011: clock divided by 4 0100: clock divided by 8 0101: clock divided by 16 0110: clock divided by 32 0111: clock divided by 64 1000: clock divided by 128 1001: clock divided by 256 1010-111: reserved.
26:24	rcompg2_div	RW	3'h1	Gen2 RCOMP clock divider: 000: clock gated 001: clock divided by 1 010: clock divided by 2 011: clock divided by 4 100: clock divided by 8 101: clock divided by 16 110: clock divided by 32

				111: reserved.
23:19	reserved	RO	5'h0	Reserved.
18:16	rcompg1_div	RW	3'h1	Gen1 RCOMP clock divider: 000: clock gated 001: clock divided by 1 010: clock divided by 2 011: clock divided by 4 100: clock divided by 8 101: clock divided by 16 110: clock divided by 32 111: reserved.
15:11	reserved	RO	5'h0	Reserved.
10:8	sys_div	RW	3'h2	System clock divider: 000: clock gated 001: clock divided by 1 010: clock divided by 2 011: clock divided by 4 100: clock divided by 8 101: clock divided by 16 11x: reserved.
7:3	reserved	RO	5'h0	Reserved.
2:0	rxoff_div	RW	3'h1	Rx offset clock divider: 000: clock gated 001: clock divided by 1 010: clock divided by 2 011: clock divided by 4 100: clock divided by 8 101: clock divided by 16 110: clock divided by 32 111: clock divided by 64

Table 71: AIB FSM Clock Divider register.

3.4.32 **AIB RCOMP1 register (rcomp1)**

AIB RCOMP register (rcomp) – 0x388 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	rcompg2_done	RO	1'h0	Gen2 calibration done indication.

30	rcompg2_calsel_ovrd	RW	1'h0	Gen2 calibration selection override.
29	rcompg2_calcode_ovrd	RW	1'h0	Gen2 RCOMP calibration code override.
28:23	reserved	RO	6'h0	Reserved.
22:16	rcomp_calcode_g2	RW	7'h0	Gen2 RCOMP calibration code
15	rcompg1_done	RO	1'h0	Gen1 calibration done indication.
14	rcompg1_calsel_ovrd	RW	1'h0	Gen1 calibration selection override.
13	rcompg1_calcode_ovrd	RW	1'h0	Gen1 RCOMP calibration code override.
12:7	reserved	RO	6'h0	Reserved.
6:0	rcomp_calcode_g1	RW	7'h0	Gen1 RCOMP calibration code.

Table 72: AIB RCOMP1 register.

3.4.33 AIB RCOMP2 register (rcomp2)

AIB RCOMP register (rcomp) – 0x38C + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:15	reserved	RO	17'h0	Reserved.
14:8	calcode_g2_sta	RO	7'h0	Rx compensation calibration code status for gen 2 mode.
7	reserved	RO	1'h0	Reserved.
6:0	calcode_g1_sta	RO	7'h0	Rx compensation calibration code status for gen 1 mode.

Table 73: AIB RCOMP1 register.

3.4.34 AIB Non-Test Logic 1 register (aib_ntl1)

AIB Non-Test Logic 1 register (aib_ntl1) – 0x390 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	ntl_done	RO	1'h0	Non test logic done
30:24	reserved	RO	7'h0	Reserved.
23	rxen_async_ovrd	RW	1'h0	RX asynchronous enable override

22	txen_async_ovrd	RW	1'h0	TX asynchronous enable override
21	ntl_done_ovrd	RW	1'h0	Non test logic done override
20	tx_dodd_ovrd	RW	1'h0	TX odd data override
19	tx_deven_ovrd	RW	1'h0	TX even data override
18	rxen_ovrd	RW	1'h0	RX enable override
17	txen_ovrd	RW	1'h0	TX enable override
16	ntl_en_ovrd	RW	1'h0	Non test logic enable override.
15	ntl_en	RW	1'h0	Non test logic enable.
14:7	reserved	RO	8'h0	Reserved.
6:0	pad_num	RW	7'h0	PAD number selection.

Table 74: AIB Non-Test Logic 1 register.

3.4.35 Non-Test Logic 2 register (aib_ntl2)

AIB Non-Test Logic 2 register (aib_ntl2) – 0x394 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31:16	ntl_cnt_val	RO	16'h0	Non test logic counter.
15:0	ntl_count	RW	16'h0	Override code for non test logic counter.

Table 75: Non-Test Logic 2 register.

3.4.36 AIB ADC0 (adc0)

AIB ADC0 (adc0) – 0xC000				
Bits	Field	Type	Reset	Description
31	adc0_start	RW	1'h0	Starts ADC0 conversion. Goes to 0 after adc_done changes from 0->1
30:24	reserved	RO	7'h0	Reserved
23	adc0_dfx_en	RW	1'h0	Enables ADC0 DFX.
22:20	adc0_in_sel	RW	3'h0	Selects ADC0 input channel
19:18	adc0_ckdiv_sel	RW	2'h0	Selects the ADC0 input clock divider
17	adc0_en	RW	1'h0	Enables ADC0.

16	adc0_chopen	RW	1'h0	Swaps input and ref to ADC0
15	adc0_done	RO	1'h0	Set to 1 after ADC0 conversion is done. Set to 0 after start_adc goes from 0->1
14:10	reserved	RO	5'h0	Reserved
9:0	adc0_value	RO	10'h0	ADC0 converted data

Table 76: AIB ADC0 register.

3.4.37 AIB ADC1 (adc1)

AIB ADC1 (adc1) – 0xC004				
Bits	Field	Type	Reset	Description
31	adc1_start	RW	1'h0	Starts ADC1 conversion. Goes to 0 after adc_done changes from 0->1
30:24	reserved	RO	7'h0	Reserved
23	adc1_dfx_en	RW	1'h0	Enables ADC1 DFX.
22:20	adc1_in_sel	RW	3'h0	Selects ADC1 input channel
19:18	adc1_ckdiv_sel	RW	2'h0	Selects the ADC1 input clock divider
17	adc1_en	RW	1'h0	Enables ADC1.
16	adc1_chopen	RW	1'h0	Swaps input and ref to ADC1
15	adc1_ done	RO	1'h0	Set to 1 after ADC1 conversion is done. Set to 0 after start_adc goes from 0->1
14:10	reserved	RO	5'h0	Reserved
9:0	adc1_value	RO	10'h0	ADC1 converted data

Table 77: AIB ADC1 register.

3.4.38 AIB ADC2 (adc2)

AIB ADC2 (adc2) – 0xC008				
Bits	Field	Type	Reset	Description
31	adc2_start	RW	1'h0	Starts ADC2 conversion. Goes to 0 after adc_done changes from 0->1
30:24	reserved	RO	7'h0	Reserved
23	adc2_dfx_en	RW	1'h0	Enables ADC2 DFX.

22:20	adc2_in_sel	RW	3'h0	Selects ADC2 input channel
19:18	adc2_ckdiv_sel	RW	2'h0	Selects the ADC2 input clock divider
17	adc2_en	RW	1'h0	Enables ADC2.
16	adc2_chopen	RW	1'h0	Swaps input and ref to ADC2
15	adc2_ done	RO	1'h0	Set to 1 after ADC2 conversion is done. Set to 0 after start_adc goes from 0->1
14:10	reserved	RO	5'h0	Reserved
9:0	adc2_value	RO	10'h0	ADC2 converted data

Table 78: AIB ADC2 register.

3.4.39 **AIB ADC3 (adc3)**

AIB ADC3 (adc3) – 0xC00C				
Bits	Field	Type	Reset	Description
31	adc3_start	RW	1'h0	Starts ADC3 conversion. Goes to 0 after adc_done changes from 0->1
30:24	reserved	RO	7'h0	Reserved
23	adc3_dfx_en	RW	1'h0	Enables ADC3 DFX.
22:20	adc3_in_sel	RW	3'h0	Selects ADC3 input channel
19:18	adc3_ckdiv_sel	RW	2'h0	Selects the ADC3 input clock divider
17	adc3_en	RW	1'h0	Enables ADC3.
16	adc3_chopen	RW	1'h0	Swaps input and ref to ADC3
15	adc3_ done	RO	1'h0	Set to 1 after ADC3 conversion is done. Set to 0 after start_adc goes from 0->1
14:10	reserved	RO	5'h0	Reserved
9:0	adc3_value	RO	10'h0	ADC3 converted data

Table 79: AIB ADC3 register.

3.4.40 **AIB ADC4 (adc4)**

AIB ADC4 (adc4) – 0xC010				
Bits	Field	Type	Reset	Description
31	adc4_start	RW	1'h0	Starts ADC4 conversion. Goes to 0 after adc_done changes from 0->1
30:24	reserved	RO	7'h0	Reserved
23	adc4_dfx_en	RW	1'h0	Enables ADC4 DFX.
22:20	adc4_in_sel	RW	3'h0	Selects ADC4 input channel
19:18	adc4_ckdiv_sel	RW	2'h0	Selects the ADC4 input clock divider
17	adc4_en	RW	1'h0	Enables ADC4.
16	adc4_chopen	RW	1'h0	Swaps input and ref to ADC4
15	adc4_ done	RO	1'h0	Set to 1 after ADC4 conversion is done. Set to 0 after start_adc goes from 0->1
14:10	reserved	RO	5'h0	Reserved
9:0	adc4_value	RO	10'h0	ADC4 converted data

Table 80: AIB ADC4 register.

3.4.41 AIB Auxiliary Channel (auxch)

AIB Auxiliary Channel register (auxch) – 0xC018				
Bits	Field	Type	Reset	Description
31:3	reserved	RO	29'h0	Reserved
2:0	auxch_rbuf_code	RW	3'h0	Configures RX buffers.

Table 81: AIB Auxiliary Channel register.

3.4.42 AIB PVTA0 (pvta0)

AIB PVTA0 register (pvta0) – 0xC024 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mon_en	RW	1'h0	Enables PVT monitor
30	mon_dfx_en	RW	1'h0	Enables PVT monitor DFX

29:27	mon_digview_sel	RW	3'h0	PVT monitor digital view selection.
26:24	mref_clkdiv	RW	3'h0	Reference clock divider
23	reserved	RO	1'b0	Reserved.
22:20	mosc_ckdiv	RW	3'h0	Oscillator clock divider
19	reserved	RO	1'b0	Reserved.
18:16	mosc_sel	RW	3'h0	Selects oscillator for measurement
15:0	reserved	RO	16'b0	Reserved.

Table 82: AIB PVTA0 register.

3.4.43 AIB PVTB0 (pvtb0)

AIB PVTB0 register (pvtb0) – 0xC028 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mcnt_done	RO	1'b0	Goes for 0->1 when oscillator counter is done.
30:17	reserved	RO	14'b0	Reserved.
16	mcnt_start	RW	1'b0	Starts oscillator counter.
15:10	reserved	RO	6'b0	
9:0	mosc_ck_cnt	RO	10'b0	Output of oscillator clock count.

Table 83: AIB PVTB0 register.

3.4.44 AIB PVTA1 (pvta1)

AIB PVTA1 register (pvta1) – 0xC034 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mon_en	RW	1'h0	Enables PVT monitor
30	mon_dfx_en	RW	1'h0	Enables PVT monitor DFX

29:27	mon_digview_sel	RW	3'h0	PVT monitor digital view selection.
26:24	mref_clkdiv	RW	3'h0	Reference clock divider
23	reserved	RO	1'b0	Reserved.
22:20	mosc_ckdiv	RW	3'h0	Oscillator clock divider
19	reserved	RO	1'b0	Reserved.
18:16	mosc_sel	RW	3'h0	Selects oscillator for measurement
15:0	reserved	RO	16'b0	Reserved.

Table 84: AIB PVTA1 register.

3.4.45 AIB PVTB1 (pvtb1)

AIB PVTB1 register (pvtb1) – 0xC038 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mcnt_done	RO	1'b0	Goes for 0->1 when oscillator counter is done.
30:17	reserved	RO	14'b0	Reserved.
16	mcnt_start	RW	1'b0	Starts oscillator counter.
15:10	reserved	RO	6'b0	Reserved.
9:0	mosc_ck_cnt	RO	10'b0	Output of oscillator clock count.

Table 85: AIB PVTB1 register.

3.4.46 AIB PVTA2 (pvta2)

AIB PVTA2 register (pvta2) – 0xC044 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mon_en	RW	1'h0	Enables PVT monitor
30	mon_dfx_en	RW	1'h0	Enables PVT monitor DFX

29:27	mon_digview_sel	RW	3'h0	PVT monitor digital view selection.
26:24	mref_clkdiv	RW	3'h0	Reference clock divider
23	reserved	RO	1'b0	Reserved.
22:20	mosc_ckdiv	RW	3'h0	Oscillator clock divider
19	reserved	RO	1'b0	Reserved.
18:16	mosc_sel	RW	3'h0	Selects oscillator for measurement
15:0	reserved	RO	16'b0	Reserved.

Table 86: AIB PVTA2 register.

3.4.47 AIB PVTB2 (pvtb2)

AIB PVTB2 register (pvtb2) – 0xC048 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mcnt_done	RO	1'b0	Goes for 0->1 when oscillator counter is done.
30:17	reserved	RO	14'b0	Reserved.
16	mcnt_start	RW	1'b0	Starts oscillator counter.
15:10	reserved	RO	6'b0	Reserved.
9:0	mosc_ck_cnt	RO	10'b0	Output of oscillator clock count.

Table 87: AIB PVTB2 register.

3.4.48 AIB PVTA3 (pvta3)

AIB PVTA3 register (pvta3) – 0xC054 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mon_en	RW	1'h0	Enables PVT monitor
30	mon_dfx_en	RW	1'h0	Enables PVT monitor DFX

29:27	mon_digview_sel	RW	3'h0	PVT monitor digital view selection.
26:24	mref_clkdiv	RW	3'h0	Reference clock divider
23	reserved	RO	1'b0	Reserved.
22:20	mosc_ckdiv	RW	3'h0	Oscillator clock divider
19	reserved	RO	1'b0	Reserved.
18:16	mosc_sel	RW	3'h0	Selects oscillator for measurement
15:0	reserved	RO	16'b0	Reserved.

Table 88: AIB PVTA3 register.

3.4.49 AIB PVTB3 (pvtb3)

AIB PVTB3 register (pvtb3) – 0xC058 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mcnt_done	RO	1'b0	Goes for 0->1 when oscillator counter is done.
30:17	reserved	RO	14'b0	Reserved.
16	mcnt_start	RW	1'b0	Starts oscillator counter.
15:10	reserved	RO	6'b0	Reserved.
9:0	mosc_ck_cnt	RO	10'b0	Output of oscillator clock count.

Table 89: AIB PVTB3 register.

3.4.50 AIB PVTA4 (pvta4)

AIB PVTA4 register (pvta4) – 0xC064 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mon_en	RW	1'h0	Enables PVT monitor
30	mon_dfx_en	RW	1'h0	Enables PVT monitor DFX

29:27	mon_digview_sel	RW	3'h0	PVT monitor digital view selection.
26:24	mref_clkdiv	RW	3'h0	Reference clock divider
23	reserved	RO	1'b0	Reserved.
22:20	mosc_ckdiv	RW	3'h0	Oscillator clock divider
19	reserved	RO	1'b0	Reserved.
18:16	mosc_sel	RW	3'h0	Selects oscillator for measurement
15:0	reserved	RO	16'b0	Reserved.

Table 90: AIB PVTA4 register.

3.4.51 AIB PVTB4 (pvtb4)

AIB PVTB4 register (pvtb4) – 0xC068 + (channel_nb*0x800)				
Bits	Field	Type	Reset	Description
31	mcnt_done	RO	1'b0	Goes for 0->1 when oscillator counter is done.
30:17	reserved	RO	14'b0	Reserved.
16	mcnt_start	RW	1'b0	Starts oscillator counter.
15:10	reserved	RO	6'b0	Reserved.
9:0	mosc_ck_cnt	RO	10'b0	Output of oscillator clock count.

Table 91: AIB PVTB4 register.

3.4.52 BERT Domain Registers

TX BERT DOMAIN registers				
SEEDP0_0 (acc_addr[4:0] = 5'h0)				
Bits	Field	Type	Reset	Description
31:0	seedp0_0	WO	32'hFFFF_FFFF	Register to configure the bits from 31 to 0 of seed register for TX BERT0 generator. When 127-bit pattern is

				selected, the least significant bit of the register is not used by the hardware.
SEEDP1_0 (acc_addr[4:0] = 5'h1)				
Bits	Field	Type	Reset	Description
31:0	seedp1_0	WO	32'hFFFF_FFFF	Register to configure the bits from 63 to 32 of seed register for TX BERT0 generator.
SEEDP2_0 (acc_addr[4:0] = 5'h2)				
Bits	Field	Type	Reset	Description
31:0	seedp2_0	WO	32'hFFFF_FFFF	Register to configure the bits from 95 to 64 of seed register for TX BERT0 generator.
SEEDP3_0 (acc_addr[4:0] = 5'h3)				
Bits	Field	Type	Reset	Description
31:0	seedp3_0	WO	32'hFFFF_FFFF	Register to configure the bits from 127 to 96 of seed register for TX BERT0 generator.
SEEDP0_1 (acc_addr[4:0] = 5'h4)				
Bits	Field	Type	Reset	Description
31:0	seedp0_1	WO	32'hFFFF_FFFF	Register to configure the bits from 31 to 0 of seed register for TX BERT1 generator. When 127-bit pattern is selected, the least significant bit of the register is not used by the hardware.
SEEDP1_1 (acc_addr[4:0] = 5'h5)				
Bits	Field	Type	Reset	Description
31:0	seedp1_1	WO	32'hFFFF_FFFF	Register to configure the bits from 95 to 64 of seed register for TX BERT1 generator.
SEEDP2_1 (acc_addr[4:0] = 5'h6)				
Bits	Field	Type	Reset	Description
31:0	seedp2_1	WO	32'hFFFF_FFFF	Register to configure the bits from 127 to 96 of seed register for TX BERT1 generator.
SEEDP3_1 (acc_addr[4:0] = 5'h7)				
Bits	Field	Type	Reset	Description
31:0	seedp3_1	WO	32'hFFFF_FFFF	Register to configure the bits from 127 to 96 of seed register for TX BERT1 generator.
SEEDP0_2 (acc_addr[4:0] = 5'h8)				
Bits	Field	Type	Reset	Description

31:0	seedp0_2	WO	32'hFFFF_FFFF	Register to configure the bits from 31 to 0 of seed register for TX BERT2 generator. When 127-bit pattern is selected, the least significant bit of the register is not used by the hardware.
SEEDP1_2 (acc_addr [4:0] = 5'h9)				
Bits	Field	Type	Reset	Description
31:0	seedp1_2	WO	32'hFFFF_FFFF	Register to configure the bits from 63 to 32 of seed register for TX BERT2 generator.
SEEDP2_2 (acc_addr[4:0] = 5'hA)				
Bits	Field	Type	Reset	Description
31:0	Seedp2_2	WO	32'hFFFF_FFFF	Register to configure the bits from 95 to 64 of seed register for TX BERT2 generator.
SEEDP3_2 (acc_addr[4:0] = 5'hB)				
Bits	Field	Type	Reset	Description
31:0	seedp3_2	WO	32'hFFFF_FFFF	Register to configure the bits from 127 to 96 of seed register for TX BERT2 generator.
SEEDP0_3 (acc_addr[4:0] = 5'hC)				
Bits	Field	Type	Reset	Description
31:0	seedp0_3	WO	32'hFFFF_FFFF	Register to configure the bits from 31 to 0 of seed register for TX BERT3 generator. When 127-bit pattern is selected, the least significant bit of the register is not used by the hardware.
SEEDP1_3 (acc_addr[4:0] = 5'hD)				
Bits	Field	Type	Reset	Description
31:0	seedp1_3	WO	32'hFFFF_FFFF	Register to configure the bits from 63 to 32 of seed register for TX BERT3 generator.
SEEDP2_3 (acc_addr[4:0] = 5'hE)				
Bits	Field	Type	Reset	Description
31:0	seedp2_3	WO	32'hFFFF_FFFF	Register to configure the bits from 95 to 64 of seed register for TX BERT3 generator.
SEEDP3_3 (acc_addr[4:0] = 5'hF)				
Bits	Field	Type	Reset	Description

31:0	seedp3_3	WO	32'hFFFF_FFFF	Register to configure the bits from 127 to 96 of seed register for TX BERT3 generator.
GEN_SEL0 (acc_addr[4:0] = 5'h10)				
Bits	Field	Type	Reset	Description
31:30	lane15_sel	RW	2'h0	Transmit generator option for lane 15: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
29:28	lane14_sel	RW	2'h0	Transmit generator option for lane 14: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
27:26	lane13_sel	RW	2'h0	Transmit generator option for lane 13: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
25:24	lane12_sel	RW	2'h0	Transmit generator option for lane 12: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
23:22	lane11_sel	RW	2'h0	Transmit generator option for lane 11: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
21:20	lane10_sel	RW	2'h0	Transmit generator option for lane 10: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
19:18	lane9_sel	RW	2'h0	Transmit generator option for lane 9: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.

17:16	lane8_sel	RW	2'h0	Transmit generator option for lane 8: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
15:14	lane7_sel	RW	2'h0	Transmit generator option for lane 7: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
13:12	lane6_sel	RW	2'h0	Transmit generator option for lane 6: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
11:10	lane5_sel	RW	2'h0	Transmit generator option for lane 5: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
9:8	lane4_sel	RW	2'h0	Transmit generator option for lane 4: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
7:6	lane3_sel	RW	2'h0	Transmit generator option for lane 3: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
5:4	lane2_sel	RW	2'h0	Transmit generator option for lane 2: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
3:2	lane1_sel	RW	2'h0	Transmit generator option for lane 1: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
1:0	lane0_sel	RW	2'h0	Transmit generator option for lane 0: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
GEN_SEL1 (acc_addr[4:0] = 5'h11)				

Bits	Field	Type	Reset	Description
31:30	lane31_sel	RW	2'h0	Transmit generator option for lane 31: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
29:28	lane30_sel	RW	2'h0	Transmit generator option for lane 30: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
27:26	lane29_sel	RW	2'h0	Transmit generator option for lane 29: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
25:24	lane28_sel	RW	2'h0	Transmit generator option for lane 28: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
23:22	lane27_sel	RW	2'h0	Transmit generator option for lane 27: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
21:20	lane26_sel	RW	2'h0	Transmit generator option for lane 26: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
19:18	lane25_sel	RW	2'h0	Transmit generator option for lane 25: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
17:16	lane24_sel	RW	2'h0	Transmit generator option for lane 24: 00: TX BERT0 generator.

				01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
15:14	lane23_sel	RW	2'h0	Transmit generator option for lane 23: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
13:12	lane22_sel	RW	2'h0	Transmit generator option for lane 22: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
11:10	lane21_sel	RW	2'h0	Transmit generator option for lane 21: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
9:8	lane20_sel	RW	2'h0	Transmit generator option for lane 20: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
7:6	lane19_sel	RW	2'h0	Transmit generator option for lane 19: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
5:4	lane18_sel	RW	2'h0	Transmit generator option for lane 18: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
3:2	lane17_sel	RW	2'h0	Transmit generator option for lane 17: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.

1:0	lane16_sel	RW	2′h0	Transmit generator option for lane 16: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
GEN_SEL2 (acc_addr[4:0] = 5′h12)				
Bits	Field	Type	Reset	Description
31:16	reserved	RO	16′h0	Reserved
15:14	lane39_sel	RW	2′h0	Transmit generator option for lane 39: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
13:12	lane38_sel	RW	2′h0	Transmit generator option for lane 38: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
11:10	lane37_sel	RW	2′h0	Transmit generator option for lane 37: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
9:8	lane36_sel	RW	2′h0	Transmit generator option for lane 36: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
7:6	lane35_sel	RW	2′h0	Transmit generator option for lane 35: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
5:4	lane34_sel	RW	2′h0	Transmit generator option for lane 34: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.

3:2	lane33_sel	RW	2′h0	Transmit generator option for lane 33: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
1:0	lane32_sel	RW	2′h0	Transmit generator option for lane 32: 00: TX BERT0 generator. 01: TX BERT1 generator. 10: TX BERT2 generator. 11: TX BERT3 generator.
GEN_PTRN_SEL (acc_addr[4:0] = 5′h13)				
Bits	Field	Type	Reset	Description
31:27	reserved	RO	5′h0	Reserved
26:24	gen3_ptrn_sel	RW	3′h0	Transmit pattern option for TX BERT3 generator: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.
23:19	reserved	RO	5′h0	Reserved
18:16	gen2_ptrn_sel	RW	3′h0	Transmit pattern option for TX BERT2 generator: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.
15:11	reserved	RO	5′h0	Reserved
10:8	gen1_ptrn_sel	RW	3′h0	Transmit pattern option for TX BERT1 generator: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.
7:3	reserved	RO	5′h0	Reserved
2:0	gen0_ptrn_sel	RW	3′h0	Transmit pattern option for TX BERT0 generator:

				000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.

Table 92: TX BERT Domain registers

RX BERT DOMAIN registers				
RXLANE_SEL (acc_addr[4:0] = 5'h14)				
Bits	Field	Type	Reset	Description
31:30	reserved	RO	2'h0	Reserved
29:24	chk3_lane_sel	RW	6'h0	Selects the lane number to be checked at receive side RX BERT3 checker: 0-39
23:22	reserved	RO	2'h0	Reserved
21:16	chk2_lane_sel	RW	6'h0	Selects the lane number to be checked at receive side RX BERT2 checker: 0-39
15:14	reserved	RO	2'h0	Reserved
13:8	chk1_lane_sel	RW	6'h0	Selects the lane number to be checked at receive side RX BERT1 checker: 0-39
7:6	reserved	RO	2'h0	Reserved
5:0	chk0_lane_sel	RW	6'h0	Selects the lane number to be checked at receive side RX BERT0 checker: 0-39
CHK_PTRN_SEL (acc_addr[4:0] = 5'h15)				
Bits	Field	Type	Reset	Description
31:27	reserved	RO	5'h0	Reserved
26:24	chk3_ptrn_sel	RW	3'h0	Pattern option for RX BERT3 checker: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.

23:19	reserved	RO	5′h0	Reserved
18:16	chk2_ptrn_sel	RW	3′h0	Pattern option for RX BERT2 checker: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.
15:11	reserved	RO	5′h0	Reserved
10:8	chk1_ptrn_sel	RW	3′h0	Pattern option for RX BERT1 checker: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.
7:3	reserved	RO	5′h0	Reserved
2:0	chk0_ptrn_sel	RW	3′h0	Pattern option for RX BERT0 checker: 000: PRBS-7 generator. 001: PRBS-15 generator. 010: PRBS-23 generator. 011: PRBS-31 generator. 100: 127-bit pattern buffer. 101: 128-bit pattern buffer.
RXBERT_ BITCNT_LO (acc_addr[4:0] = 5′h16)				
Bits	Field	Type	Reset	Description
31:0	rxbert0_bitcnt_lo	RO	32′h0	32 LSBs of receive bit count.
RXBERT_ BITCNT_HI (acc_addr[4:0] = 5′h17)				
Bits	Field	Type	Reset	Description
31:17	reserved	RO	15′h0	Reserved
16:0	rxbert0_bitcnt_hi	RO	17′h0	17 MSBs of receive bit count.
RXBERT0_ ERRCNT (acc_addr[4:0] = 5′h18)				
Bits	Field	Type	Reset	Description
31:16	reserved	RO	16′h0	Reserved
15:0	rxbert0_errcnt	RO	16′h0	Total number of bit errors detected by RX BERT checker 0.
RXBERT1_ ERRCNT (acc_addr[4:0] = 5′h19)				
Bits	Field	Type	Reset	Description

31:16	reserved	RO	16'h0	Reserved
15:0	rxbert1_errcnt	RO	16'h0	Total number of bit errors detected by RX BERT checker 1.
RXBERT2_ ERRCNT (acc_addr[4:0] = 5'h1A)				
Bits	Field	Type	Reset	Description
31:16	reserved	RO	16'h0	Reserved
15:0	rxbert2_errcnt	RO	16'h0	Total number of bit errors detected by RX BERT checker 2.
RXBERT3_ ERRCNT (acc_addr[4:0] = 5'h1B)				
Bits	Field	Type	Reset	Description
31:16	reserved	RO	16'h0	Reserved
15:0	rxbert3_errcnt	RO	16'h0	Total number of bit errors detected by RX BERT checker 3.
RXBERT0_ DATA0 (acc_addr[4:0] = 5'h1C)				
Bits	Field	Type	Reset	Description
31:0	rx_bert0_data0	RO	32'h0	Stores bits 31-0 from the last 128 bits received in RX BERT0 checker.
RXBERT0_ DATA1 (acc_addr[4:0] = 5'h1D)				
Bits	Field	Type	Reset	Description
31:0	rx_bert0_data1	RO	32'h0	Stores bits 63-32 from the last 128 bits received in RX BERT0 checker.
RXBERT0_ DATA2 (acc_addr[4:0] = 5'h1E)				
Bits	Field	Type	Reset	Description
31:0	rx_bert0_data2	RO	32'h0	Stores bits 95-64 from the last 128 bits received in RX BERT0 checker.
RXBERT0_ DATA3 (acc_addr[4:0] = 5'h1F)				
Bits	Field	Type	Reset	Description
31:0	rx_bert0_data3	RO	32'h0	Stores bits 127-96 from the last 128 bits received in RX BERT0 checker.
RXBERT1_ DATA0 (acc_addr[4:0] = 5'h20)				
Bits	Field	Type	Reset	Description
31:0	rx_bert1_data0	RO	32'h0	Stores bits 31-0 from the last 128 bits received in RX BERT1 checker.
RXBERT1_ DATA1 (acc_addr[4:0] = 5'h21)				
Bits	Field	Type	Reset	Description
31:0	rx_bert1_data1	RO	32'h0	Stores bits 63-32 from the last 128 bits received in RX BERT1 checker.
RXBERT1_ DATA2 (acc_addr[4:0] = 5'h22)				
Bits	Field	Type	Reset	Description
31:0	rx_bert1_data2	RO	32'h0	Stores bits 95-64 from the last 128 bits received in RX BERT1 checker.

RXBERT1_ DATA3 (acc_addr[4:0] = 5'h23)				
Bits	Field	Type	Reset	Description
31:0	rx_bert1_data3	RO	32'h0	Stores bits 127-96 from the last 128 bits received in RX BERT1 checker.
RXBERT2_ DATA0 (acc_addr[4:0] = 5'h24)				
Bits	Field	Type	Reset	Description
31:0	rx_bert2_data0	RO	32'h0	Stores bits 31-0 from the last 128 bits received in RX BERT2 checker.
RXBERT2_ DATA1 (acc_addr[4:0] = 5'h25)				
Bits	Field	Type	Reset	Description
31:0	rx_bert2_data1	RO	32'h0	Stores bits 63-32 from the last 128 bits received in RX BERT2 checker.
RXBERT2_ DATA2 (acc_addr[4:0] = 5'h26)				
Bits	Field	Type	Reset	Description
31:0	rx_bert2_data2	RO	32'h0	Stores bits 95-64 from the last 128 bits received in RX BERT2 checker.
RXBERT2_ DATA3 (acc_addr[4:0] = 5'h27)				
Bits	Field	Type	Reset	Description
31:0	rx_bert2_data3	RO	32'h0	Stores bits 127-96 from the last 128 bits received in RX BERT2 checker.
RXBERT3_ DATA0 (acc_addr[4:0] = 5'h28)				
Bits	Field	Type	Reset	Description
31:0	rx_bert3_data0	RO	32'h0	Stores bits 31-0 from the last 128 bits received in RX BERT3 checker.
RXBERT3_ DATA1 (acc_addr[4:0] = 5'h29)				
Bits	Field	Type	Reset	Description
31:0	rx_bert3_data1	RO	32'h0	Stores bits 63-32 from the last 128 bits received in RX BERT3 checker.
RXBERT3_ DATA2 (acc_addr[4:0] = 5'h2A)				
Bits	Field	Type	Reset	Description
31:0	rx_bert3_data2	RO	32'h0	Stores bits 95-64 from the last 128 bits received in RX BERT3 checker.
RXBERT3_ DATA3 (acc_addr[4:0] = 5'h2B)				
Bits	Field	Type	Reset	Description
31:0	rx_bert3_data3	RO	32'h0	Stores bits 127-96 from the last 128 bits received in RX BERT3 checker.

Table 93: RX BERT Domain registers

4 MAC Interface Signal List

Parameters	Value
NBR_CHNLS	24
NBR_PHASES	4
NBR_LANES	40
MS_SSR_LEN	81
SL_SSR_LEN	73

Table 94: RTL parameters.

Port Name	Width	Direction
data_in	[(NBR_CHNLS*2*NBR_LNS)-1:0]	*Input*
data_in_f	[(NBR_CHNLS*NBR_PHASES*2*NBR_LNS)-1:0]	*Input*
data_out	[(NBR_CHNLS*2*NBR_LNS)-1:0]	*Output*
data_out_f	[(NBR_CHNLS*NBR_PHASES*2*NBR_LNS)-1:0]	*Output*
m_ns_fwd_clk	[NBR_CHNLS-1:0]	*Input*
m_fs_fwd_clk	[NBR_CHNLS-1:0]	*Output*
m_wr_clk	[NBR_CHNLS-1:0]	*Input*
m_rd_clk	[NBR_CHNLS-1:0]	*Input*
ns_fwd_clk	[NBR_CHNLS-1:0]	*Output*
ns_fwd_clk_div	[NBR_CHNLS-1:0]	*Output*
fs_fwd_clk	[NBR_CHNLS-1:0]	*Output*
fs_fwd_clk_div	[NBR_CHNLS-1:0]	*Output*
m_ns_rcv_clk	[NBR_CHNLS-1:0]	*Input*
m_fs_rcv_clk	[NBR_CHNLS-1:0]	*Output*
ns_adapter_rstn	[NBR_CHNLS-1:0]	*Input*
ns_mac_rdy	[NBR_CHNLS-1:0]	*Input*
fs_mac_rdy	[NBR_CHNLS-1:0]	*Output*
i_conf_done	1	Input
i_osc_clk	1	Input
dual_mode_select	1	Input
m_gen2_mode	1	Input
ms_rx_dcc_dll_lock_req	[NBR_CHNLS-1:0]	Input

ms_tx_dcc_dll_lock_req	[NBR_CHNLS-1:0]	Input
sl_rx_dcc_dll_lock_req	[NBR_CHNLS-1:0]	Input
sl_tx_dcc_dll_lock_req	[NBR_CHNLS-1:0]	Input
ms_tx_transfer_en	[NBR_CHNLS-1:0]	*Output*
ms_rx_transfer_en	[NBR_CHNLS-1:0]	*Output*
sl_tx_transfer_en	[NBR_CHNLS-1:0]	*Output*
sl_rx_transfer_en	[NBR_CHNLS-1:0]	*Output*
m_rx_align_done	[NBR_CHNLS-1:0]	*Output*
sl_external_cntl_26_0	[(NBR_CHNLS*27)-1:0]	Input
sl_external_cntl_30_28	[(NBR_CHNLS*3)-1:0]	Input
sl_external_cntl_57_32	[(NBR_CHNLS*26)-1:0]	Input
ms_external_cntl_4_0	[(NBR_CHNLS*5)-1:0]	Input
ms_external_cntl_65_8	[(NBR_CHNLS*58)-1:0]	Input
sr_ms_tomac	[(NBR_CHNLS*MS_SSR_LEN)-1:0]	*Output*
sr_sl_tomac	[(NBR_CHNLS*SL_SSR_LEN)-1:0]	*Output*

Table 95: MAC Interface Signals

5 AIB IO Interface

5.3 AIB IO Signal List

Parameters	Value
NBR_BUMPS	102

Port Name	Width	Direction	Standby Driver	Power Domain
iopad_ch0_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch1_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch2_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx

iopad_ch3_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch4_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch5_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch6_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch7_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch8_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch9_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch10_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch11_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch12_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch13_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch14_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch15_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch16_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch17_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch18_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch19_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch20_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx

iopad_ch21_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch22_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_ch23_aib	[NBR_BUMPS-1:0]	Inout	Weak pull-down	vddc2/ vddtx
iopad_device_detect	1	Inout	-	vddc2
iopad_power_on_reset	1	Inout	-	vddc2

Table 96: AIB IO Signals

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[101]	Rx[38]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[100]	Rx[39]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[99]	Rx[36]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[98]	Rx[37]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[97]	Rx[34]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[96]	Rx[35]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[95]	Rx[32]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[94]	Rx[33]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[93]	Rx[30]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[92]	Rx[31]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[91]	Rx[28]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[90]	Rx[29]	Inout	0	Weak pull-down	vddtx

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[89]	Rx[26]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[88]	Rx[27]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[87]	Rx[24]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[86]	Rx[25]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[85]	Rx[22]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[84]	Rx[23]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[83]	Rx[20]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[82]	Rx[21]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[81]	Rx[18]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[80]	Rx[19]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[79]	Rx[16]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[78]	Rx[17]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[77]	Rx[14]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[76]	Rx[15]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[75]	Rx[12]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[74]	Rx[13]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[73]	Rx[10]	Inout	0	Weak pull-down	vddtx

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[72]	Rx[11]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[71]	fs_fwd_clk	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[70]	fs_fwd_clkb	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[69]	Rx[8]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[68]	Rx[9]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[67]	Rx[6]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[66]	Rx[7]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[65]	Rx[4]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[64]	Rx[5]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[63]	Rx[2]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[62]	Rx[3]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[61]	Rx[0]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[60]	Rx[1]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[59]	fs_rcv_clk	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[58]	fs_rcv_clkb	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[57]	fs_sr_clk	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[56]	fs_sr_clkb	Inout	0	Weak pull-down	vddc2

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[55]	fs_sr_data	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[54]	fs_sr_load	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[53]	fs_mac_rdy	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[52]	fs_adapter_rstn	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[51]	spare[1]	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[50]	spare[0]	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[49]	ns_adapter_rstn	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[48]	ns_mac_rdy	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[47]	ns_sr_load	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[46]	ns_sr_data	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[45]	ns_sr_clkb	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[44]	ns_sr_clk	Inout	0	Weak pull-down	vddc2
iopad_chxx_aib[43]	ns_rcv_clkb	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[42]	ns_rcv_clk	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[41]	Tx[1]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[40]	Tx[0]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[39]	Tx[3]	Inout	0	Weak pull-down	vddtx

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[38]	Tx[2]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[37]	Tx[5]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[36]	Tx[4]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[35]	Tx[7]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[34]	Tx[6]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[33]	Tx[9]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[32]	Tx[8]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[31]	ns_fwd_clkb	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[30]	ns_fwd_clk	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[29]	Tx[11]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[28]	Tx[10]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[27]	Tx[13]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[26]	Tx[12]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[25]	Tx[15]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[24]	Tx[14]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[23]	Tx[17]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[22]	Tx[16]	Inout	0	Weak pull-down	vddtx

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[21]	Tx[19]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[20]	Tx[18]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[19]	Tx[21]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[18]	Tx[20]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[17]	Tx[23]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[16]	Tx[22]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[15]	Tx[25]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[14]	Tx[24]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[13]	Tx[27]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[12]	Tx[26]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[11]	Tx[29]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[10]	Tx[28]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[9]	Tx[31]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[8]	Tx[30]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[7]	Tx[33]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[6]	Tx[32]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[5]	Tx[35]	Inout	0	Weak pull-down	vddtx

Port Name	Bump name	Direction	Reset value	Standby driver	Power Domain
iopad_chxx_aib[4]	Tx[34]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[3]	Tx[37]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[2]	Tx[36]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[1]	Tx[39]	Inout	0	Weak pull-down	vddtx
iopad_chxx_aib[0]	Tx[38]	Inout	0	Weak pull-down	vddtx

Table 97: IOPAD ports per channel.

5.4 Micro bump Signal Assignment

		With Redundancy and Fault on	
µBump	Original	AIB0	AIB101
AIB0	tx[38]	FAULT	tx[38]
AIB2	tx[36]	tx[38]	tx[36]
AIB4	tx[34]	tx[36]	tx[34]
AIB6	tx[32]	tx[34]	tx[32]
AIB8	tx[30]	tx[32]	tx[30]
AIB10	tx[28]	tx[30]	tx[28]
AIB12	tx[26]	tx[28]	tx[26]
AIB14	tx[24]	tx[26]	tx[24]
AIB16	tx[22]	tx[24]	tx[22]
AIB18	tx[20]	tx[22]	tx[20]
AIB20	tx[18]	tx[20]	tx[18]
AIB22	tx[16]	tx[18]	tx[16]
AIB24	tx[14]	tx[16]	tx[14]
AIB26	tx[12]	tx[14]	tx[12]
AIB28	tx[10]	tx[12]	tx[10]
AIB30	ns_fwd_clk	tx[10]	ns_fwd_clk
AIB32	tx[8]	ns_fwd_clk	tx[8]
AIB34	tx[6]	tx[8]	tx[6]
AIB36	tx[4]	tx[6]	tx[4]
AIB38	tx[2]	tx[4]	tx[2]
AIB40	tx[0]	tx[2]	tx[0]
AIB42	ns_rcv_clk	tx[0]	ns_rcv_clk
AIB44	ns_sr_clk	ns_rcv_clk	ns_sr_clk
AIB46	ns_sr_data	ns_sr_clk	ns_sr_data
AIB48	ns_mac_rdy	ns_sr_data	ns_mac_rdy
AIB50	spare[0]	ns_mac_rdy	fs_adapter_rstn
AIB52	fs_adapter_rstn	fs_adapter_rstn	fs_sr_load
AIB54	fs_sr_load	fs_sr_load	fs_sr_clkb
AIB56	fs_sr_clkb	fs_sr_clkb	fs_rcv_clkb
AIB58	fs_rcv_clkb	fs_rcv_clkb	rx[1]
AIB60	rx[1]	rx[1]	rx[3]
AIB62	rx[3]	rx[3]	rx[5]
AIB64	rx[5]	rx[5]	rx[7]
AIB66	rx[7]	rx[7]	rx[9]
AIB68	rx[9]	rx[9]	fs_fwd_clkb
AIB70	fs_fwd_clkb	fs_fwd_clkb	rx[11]
AIB72	rx[11]	rx[11]	rx[13]
AIB74	rx[13]	rx[13]	rx[15]
AIB76	rx[15]	rx[15]	rx[17]
AIB78	rx[17]	rx[17]	rx[19]
AIB80	rx[19]	rx[19]	rx[21]
AIB82	rx[21]	rx[21]	rx[23]
AIB84	rx[23]	rx[23]	rx[25]
AIB86	rx[25]	rx[25]	rx[27]
AIB88	rx[27]	rx[27]	rx[29]
AIB90	rx[29]	rx[29]	rx[31]
AIB92	rx[31]	rx[31]	rx[33]
AIB94	rx[33]	rx[33]	rx[35]
AIB96	rx[35]	rx[35]	rx[37]
AIB98	rx[37]	rx[37]	rx[39]
AIB100	rx[39]	rx[39]	UNUSED

		With Redundancy and Fault on	
µBump	Original	AIB0	AIB101
AIB1	tx[39]	UNUSED	tx[39]
AIB3	tx[37]	tx[39]	tx[37]
AIB5	tx[35]	tx[37]	tx[35]
AIB7	tx[33]	tx[35]	tx[33]
AIB9	tx[31]	tx[33]	tx[31]
AIB11	tx[29]	tx[31]	tx[29]
AIB13	tx[27]	tx[29]	tx[27]
AIB15	tx[25]	tx[27]	tx[25]
AIB17	tx[23]	tx[25]	tx[23]
AIB19	tx[21]	tx[23]	tx[21]
AIB21	tx[19]	tx[21]	tx[19]
AIB23	tx[17]	tx[19]	tx[17]
AIB25	tx[15]	tx[17]	tx[15]
AIB27	tx[13]	tx[15]	tx[13]
AIB29	tx[11]	tx[13]	tx[11]
AIB31	ns_fwd_clkb	tx[11]	ns_fwd_clkb
AIB33	tx[9]	ns_fwd_clkb	tx[9]
AIB35	tx[7]	tx[9]	tx[7]
AIB37	tx[5]	tx[7]	tx[5]
AIB39	tx[3]	tx[5]	tx[3]
AIB41	tx[1]	tx[3]	tx[1]
AIB43	ns_rcv_clkb	tx[1]	ns_rcv_clkb
AIB45	ns_sr_clkb	ns_rcv_clkb	ns_sr_clkb
AIB47	ns_sr_load	ns_sr_clkb	ns_sr_load
AIB49	ns_adapter_rstn	ns_sr_load	ns_adapter_rstn
AIB51	spare[1]	ns_adapter_rstn	fs_mac_rdy
AIB53	fs_mac_rdy	fs_mac_rdy	fs_sr_data
AIB55	fs_sr_data	fs_sr_data	fs_sr_clk
AIB57	fs_sr_clk	fs_sr_clk	fs_rcv_clk
AIB59	fs_rcv_clk	fs_rcv_clk	rx[0]
AIB61	rx[0]	rx[0]	rx[2]
AIB63	rx[2]	rx[2]	rx[4]
AIB65	rx[4]	rx[4]	rx[6]
AIB67	rx[6]	rx[6]	rx[8]
AIB69	rx[8]	rx[8]	fs_fwd_clk
AIB71	fs_fwd_clk	fs_fwd_clk	rx[10]
AIB73	rx[10]	rx[10]	rx[12]
AIB75	rx[12]	rx[12]	rx[14]
AIB77	rx[14]	rx[14]	rx[16]
AIB79	rx[16]	rx[16]	rx[18]
AIB81	rx[18]	rx[18]	rx[20]
AIB83	rx[20]	rx[20]	rx[22]
AIB85	rx[22]	rx[22]	rx[24]
AIB87	rx[24]	rx[24]	rx[26]
AIB89	rx[26]	rx[26]	rx[28]
AIB91	rx[28]	rx[28]	rx[30]
AIB93	rx[30]	rx[30]	rx[32]
AIB95	rx[32]	rx[32]	rx[34]
AIB97	rx[34]	rx[34]	rx[36]
AIB99	rx[36]	rx[36]	rx[38]
AIB101	rx[38]	rx[38]	FAULT

Table 98: Micro bump Signal Assignment

5.5 Micro bump Map

Document medium-density micro bump map

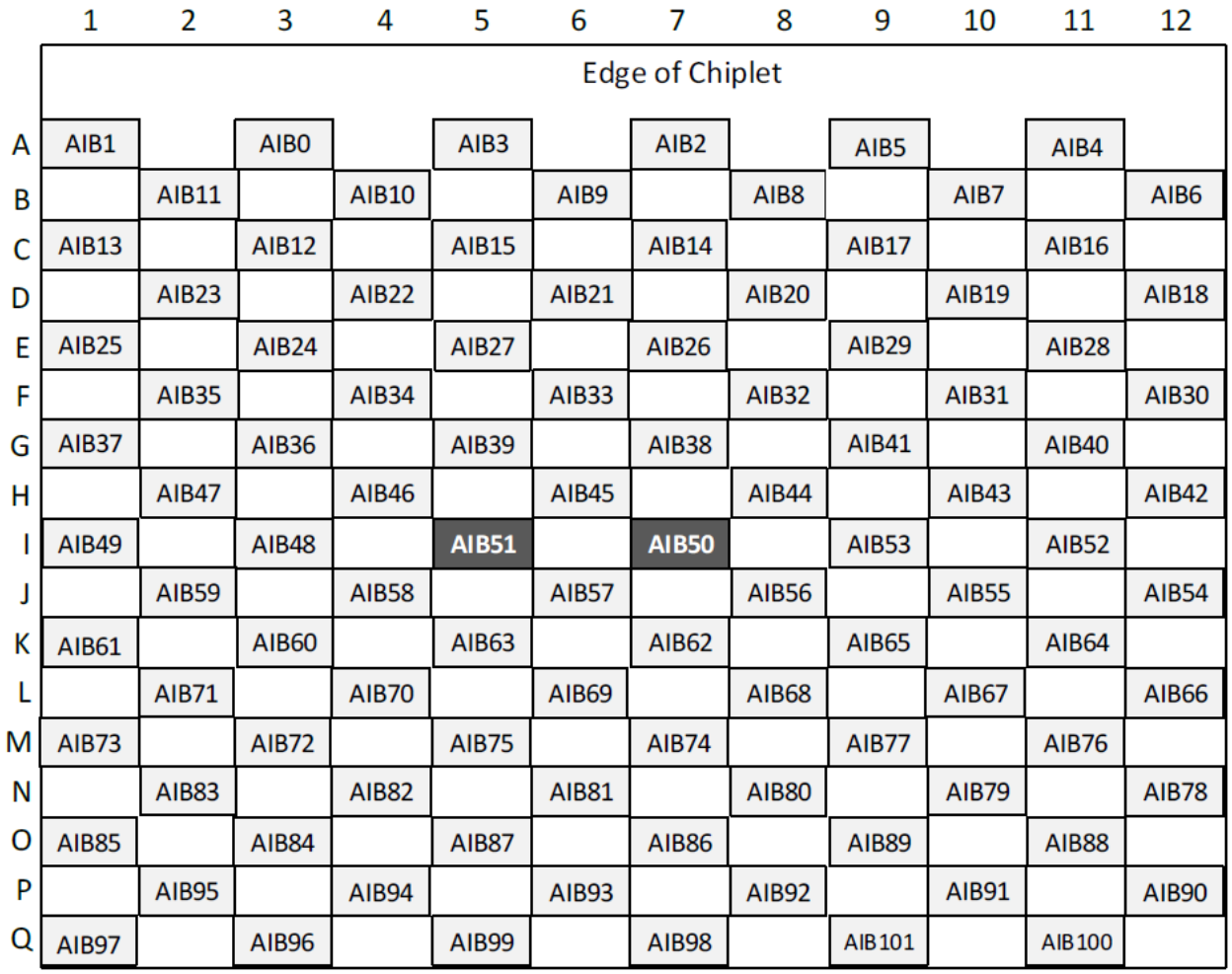

Figure 44: Medium-Density Micro bump Map

These 102 micro bump maps support a Gen2 AIB Plus interface with 80 balanced (40 TX and 40 RX) IOs.

5.6 Micro bump Configuration

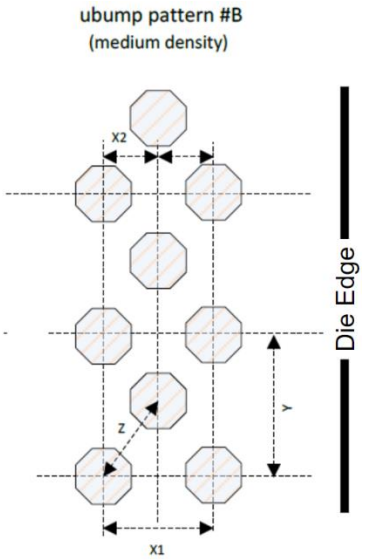

Figure 45: Micro bump Spacing

Symbol	Description	Medium-Density (µm)
X1	Aligned-row bump-to-bump pitch	36.70
X2	Staggered-row bump-to-bump pitch	73.40
Z	Diagonal bump-to-bump pitch	45.00
Y	Aligned-column bump-to-bump pitch	52.08

Table 99: Micro bump Spacing Specifications

The channel height is required to match the existing AIB Gen1 channel height, which is 312.48 µm. Thus, for medium-density micro bump spacings, Y = 312.48 / 6 = 52.08 µm. The minimum micro bump pitch allowed in manufacturing is 45 µm. This sets Z. X1 is derived from these two dimensions. $X1 = \sqrt{Z^2 - (\frac{Y}{2})^2}$ and X2 = 2(X1).

5.7 IO Control and States

Pin name	Direction	Active value	Description
ckrx_odd	input	N.A.	Deserializer clock used to launch receive odd data to adapter

ckrx_even	input	N.A.	Deserializer clock used to launch receive even data to adapter
rx_en	input	1	Configures block as synchronous receive pad
cktx_odd	input	N.A.	Serializer clock used to capture transmit odd data from adapter
cktx_even	input	N.A.	Serializer clock used to capture transmit even data from adapter
tx_en	input	1	Configures pad as synchronous transmit one
tx_datain_odd	input	N.A.	Synchrounous transmit odd data from adapter
tx_datain_even	input	N.A.	Synchrounous transmit even data from adapter
tx_async	input	N.A.	Asynchronous transmit data from adapter
tx_async_en	input	1	Configures block as asynchronous transmit pad
rx_async_en	input	1	Configures block as asynchronous receive pad
sdr_mode_en	input	1	Enables SDR mode
wkpu_en	input	1	Enables weak pull-up driver.
wkpd_en	input	1	Enables weak pull-down driver.
rx_out_even	output	N.A.	Receive synchronous even data to adapter
rx_out_odd	output	N.A.	Receive synchronous odd data to adapter
rx_async	output	N.A.	Receive asynchronous data to adapter
xx_pad	inout	N.A.	IO pad port

Table 100 Description of main IO pad macro pins

Pin name	Values					
rx_en	X	X	1	0	0	0
tx_en	X	X	0	1	0	0
tx_async_en	X	X	0	0	1	0
rx_async_en	X	X	0	0	0	1
wkpu_en	1	0	0	0	0	0
wkpd_en	X	1	0	0	0	0
rx_out_even	0	0	sampled even data	0	0	0
rx_out_odd	0	0	sampled odd data	0	0	0
rx_async	0	0	0	0	0	Asynchronos data
xx_pad	weak 1	weak0	Data from bump	odd/even data from adapter	Asynchronous data from adapter	Data from bump

Table 101 Simplified IO pad macro truth table

5.8 Electrical Specifications

The PHY is designed to meet the AIB 2.0 specifications. The achieved performance and comparison against AIB 2.0 requirements can be found in IO and Signal Integrity report.

5.8.1 Voltage Sequencing Requirements

VDDC1/2 should be power up before VDDTX is brought up. Power on reset should be settled to all of the IOs, in order to ensure this occurs there should be a minimum of 20ns between VDDC1/2 and VDDTX power up.

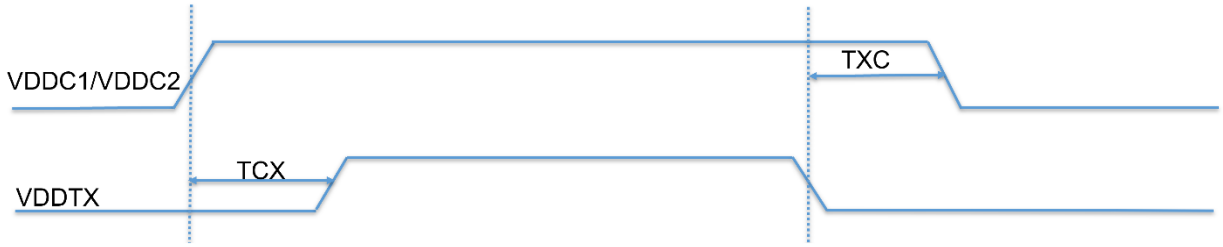

Figure 46 Power Ramp Sequence

5.8.2 Input CAP Values

	Input Cap (fF)	Vs. AIB 2.0 Standard
Gen1	438	Meets spec
Gen2	406	106fF above spec

Table 102 Input CAP Values

5.9 Timing Specifications

The PHY is designed to meet the AIB 2.0 specifications. The achieved performance and comparison against AIB 2.0 requirements can be found in IO and Signal Integrity report.

5.9.1 Latency

The specified latencies include only the sequential delay and does not include analog delays through IO buffers and physical connections. Besides, for DDR operation, it is considered the delay for data even data only and thus an extra value of 0.5 AIB IO clock period needs to be added to the table values to get the total delay of data odd data.

Figure 47 and **Figure 48** shows the minimum data latency and maximum data latency between two PHYs, respectively. The difference between maximum and minimum delays for FIFO mode are due to the uncertainty introduced by FIFO synchronizers. The phase compensation mechanism supports an extra delay or drift between read and write side of

FIFO which are not considered on the tables below. DBI feature introduces an extra clock period of delay which is not considered on the table either. **Figure 47: Minimum data latency between two PHYs considering a typical SoC implementation.**

Unit = AIB IO Clock

Min Latency

FIFO 1:1

			Rx					
			AIB2.0/Gen2		AIB2.0/Gen1		AIB1.0	
			REG	FIFO	REG	FIFO	REG	FIFO
Tx	AIB2.0/Gen 2	REG	4	7				
		FIFO	6	9				
	AIB2.0/Gen 1	REG			4	7	4.5	6.5
		FIFO			6	9	6.5	8.5
	AIB1.0	REG			4.5	7.5	5	7
		FIFO			6.5	9.5	7	9

FIFO 2:1

			Rx					
			AIB2.0/Gen2		AIB2.0/Gen1		AIB1.0	
			REG	FIFO	REG	FIFO	REG	FIFO
Tx	AIB2.0/Gen 2	REG	4	10				
		FIFO	8	14				
	AIB2.0/Gen 1	REG			4	10	4.5	8.5
		FIFO			8	14	8.5	12.5
	AIB1.0	REG			4.5	10.5	5	9
		FIFO			7.5	13.5	8	12

FIFO 4:1

			Rx					
			AIB2.0/Gen2		AIB2.0/Gen1		AIB1.0	
			REG	FIFO	REG	FIFO	REG	FIFO
Tx	AIB2.0/Gen 2	REG	4	16				
		FIFO	12	24				
	AIB2.0/Gen 1	REG						
		FIFO						
	AIB1.0	REG						
		FIFO						

Figure 47: Minimum data latency between two PHYs considering a typical SoC implementation.

Unit = AIB IO Clock

Max Latency

FIFO 1:1

			Rx					
			AIB2.0/Gen2		AIB2.0/Gen1		AIB1.0	
			REG	FIFO	REG	FIFO	REG	FIFO
Tx	AIB2.0/Gen 2	REG	4	8				
		FIFO	7	11				
	AIB2.0/Gen 1	REG			4	8	4.5	7.5
		FIFO			7	11	7.5	10.5
	AIB1.0	REG			4.5	7.5	5	8
		FIFO			7.5	10.5	8	11

FIFO 2:1

			Rx					
			AIB2.0/Gen2		AIB2.0/Gen1		AIB1.0	
			REG	FIFO	REG	FIFO	REG	FIFO
Tx	AIB2.0/Gen 2	REG	4	12				
		FIFO	9	17				
	AIB2.0/Gen 1	REG			4	12	4.5	10.5
		FIFO			9	17	9.5	15.5
	AIB1.0	REG			4.5	10.5	5	11
		FIFO			8.5	14.5	9	15

FIFO 4:1

			Rx					
			AIB2.0/Gen2		AIB2.0/Gen1		AIB1.0	
			REG	FIFO	REG	FIFO	REG	FIFO
Tx	AIB2.0/Gen 2	REG	4	20				
		FIFO	13	29				
	AIB2.0/Gen 1	REG						
		FIFO						
	AIB1.0	REG						
		FIFO						

Figure 48: Maximum data latency between two PHYs considering a typical SoC implementation.

The equations below show how data latency can be calculated for one PHY in register mode for transmit side and receive side, respectively, with respect to the configuration used.

$$Tx\ REG\ mode\ delay = tx\_dbi\_en + 2$$

$$Rx\ REG\ mode\ delay = rx\_dbi\_en + 2$$

The equations below show how maximum and minimum latencies can be calculated for one PHY, respectively, for TX FIFO mode with respect to the configuration used.

$$Tx\ Delay\ (max)\ = RATIO + \ \text{tx\_phcomp}\ + \text{tx\_dbi\_en}\ + 2$$

$$Tx\ Delay\ (min)\ = RATIO + \ \text{tx\_phcomp}\ + \text{tx\_dbi\_en}\ + 1$$

The equations below show how maximum and minimum latencies can be calculated for one PHY , respectively, for RX FIFO mode with respect to the configuration used.

$$Rx\ Delay\ (max) = [(rx\_phcomp + 2) \times RATIO] + rx\_dbi\_en + 2$$

$$Rx\ Delay\ (min) = [(rx\_phcomp + 1) \times RATIO] + rx\_dbi\_en + 2$$

5.9.2 Input Clock Jitter

The input clock on pin m_ns_fwd_clk must have jitter no higher than 19ps.

6 Initialization Procedure

6.3 RCOMP Setup

The following registers shall be programed during configuration phase to set up properly pull-up/down drivers in IO pads as part of RCOMP functionality.

Mode	**GEN1**		
Register name	**io_ctrl1**		
Field name	**txnpu_code[7:0]**	**txppu_code[7:0]**	**txnpd_code[7:0]**
Config. value	**0**	**213**	**26**
MODE	**GEN2**		
Register name	**io_ctrl1**		

Field name	txnpu_code[7:0]	txppu_code[7:0]	txnpd_code[7:0]
	72	0	38

Table 103 RCOMP Default Values

7 Integration Guideline

7.3 Unconnected Instance

An unconnected instance should adhere to following guideline to avoid any unexpected behavior of IP:

- All the pads must be unconnected.
- The VDDTX supply must be unconnected or connected to ground.
- The VDDC1 supply must be unconnected or connected to ground.
- After the VDDC2 is stable the input to IP must be defined state and IP must be held in reset state to avoid any unwarranted leakage.
- No activity is expected on any input signal. The behavior of output pins is not defined for unconnected instance during any activity on input signals.

8 Silicon Debug Registers

This functionally is deprecated.

8.3 Analog Monitor

8.3.1 Analog-to-Digital Converters

The PHY has 5 ADCs which can be used for test purpose to convert signals from analog blocks. Each ADC is capable of select one signal of seven inputs to be converted. Only signals from channels 0, 5, 11, 17 and 23 can be converted. The figure below shows how ADC inputs are connected to each channel.

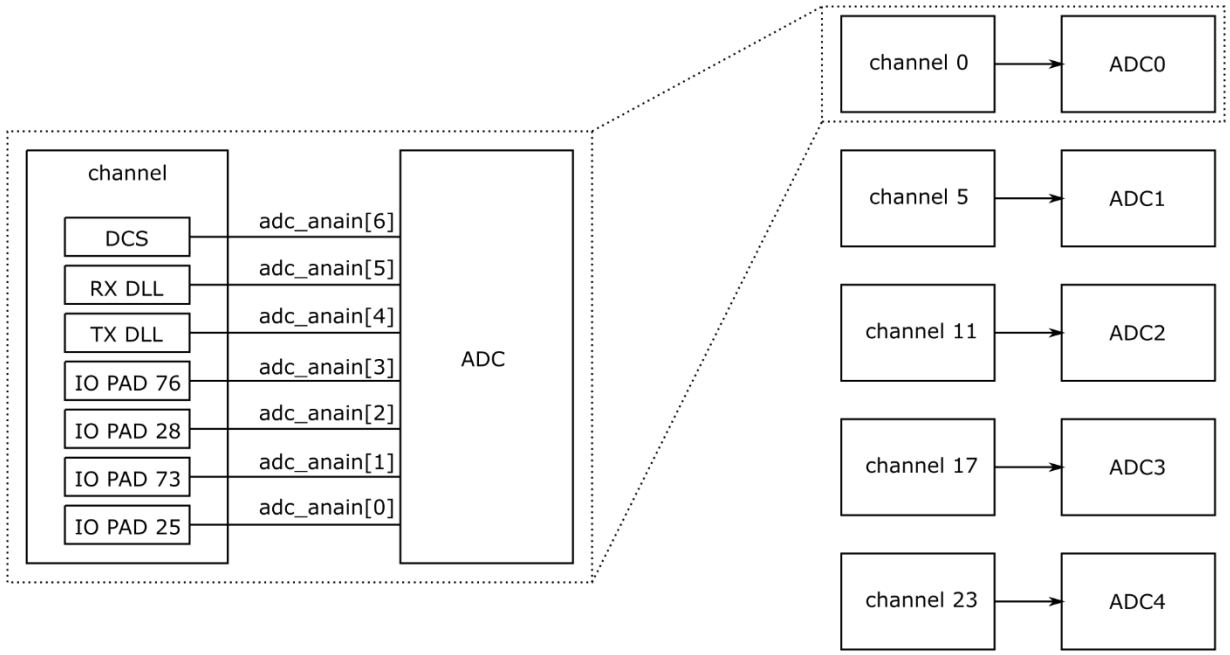

Figure 49: Channel connections with analog-to-digital converters.

ADC conversion can be triggered through CSR registers. See **Table 76**, **Table 77**, **Table 78**, **Table 79** and **Table 80** to find more details.

8.3.2 Process, Voltage and Temperature Monitors (PVT)

The hardware implements five PVT monitors to be used for test purpose. CSRs provides capability to the user of selecting different inputs and triggering a timer for measurement purpose. See **Table 82**, **Table 83**, **Table 84**, **Table 85**, **Table 86**, **Table 87**, **Table 88**, **Table 89**, **Table 90** and **Table 91** to find more details.

9 Glossary

Term	Definition
Intel 16	Intel's 22 nm FinFET Low Power process
AIB	Advanced Interconnect Bus
AUX	Auxiliary channel (or block)
AVMM	Avalon Memory-Mapped Interface
BCA	Blue Cheetah Analog Design
C4	Controlled collapse chip connection, otherwise known as a flip-chip solder bump (as opposed to an EMIB micro bump)
chiplet	
COTS	Commercial-off-the-shelf
CSR	Control and status register
DBI	Data bus inversion
DCC	Duty cycle correction (or corrector)
DCD	Duty cycle distortion
DDR	Double data rate, mode where data is transferred on both edges of the clock
deserializer	Structure in our implementation which replaces the RX FIFO to provide clock phase compensation in the receive direction
DLL	Delay-locked loop
doubleword (DW)	Unit of information exchanged between the MAC and the PHY when operating the FIFOs in 2:1 mode. Represents 160 bits of data or two 80-bit words.
EMIB	Embedded Multi-Die Interconnect Bridge, Intel's high-density silicon bridge technology used to connect two AIB PHY's together in a single module (package).
FIFO	First-in-first-out data structure. As used here and in previous versions of the ***AIB Spec***, it is one method for compensating for the phase difference between the mesochronous clocks used in the MAC and the PHY.
FSM	Finite state machine
GDSII	Graphic Design System 2, the database file format used to exchange circuit layout information.
GHz	Gigahertz ($1x10^9$ hertz)

IEEE	Institute of Electrical and Electronic Engineers
IO	Input-output
JTAG	Joint Test Action Group, standards body that created the IEEE 1149 series of boundary scan standards
MAC	Media access control layer (or controller)
mesochronous	Clocks which run at exactly the same frequency, but which have an unknown phase relationship.
micro bump	
MUX	Multiplexer
PDN	Power delivery (or distribution) network
PHY	Physical layer
PLL	Phase-locked loop
PnR	Place and route
quadword (QW)	Unit of information exchanged between the MAC and the PHY when operating the FIFOs in 4:1 mode. Represents 320 bits of data or four 80-bit words.
RX	Receive (or receiver)
SDR	Single data rate, mode where data is transferred on a single (usually rising) edge of the clock
serializer	Structure in our implementation which replaces the TX FIFO to provide clock phase compensation in the transmit direction
SoC	System-on-a-chip
SSN	Simultaneous switching noise
TX	Transmit (or transmitter)
UI	Unit interval
WAM	Word alignment marking (or marker)
word (W)	Unit of information exchanged between the MAC and the PHY when operating in register mode or the FIFOs in 1:1 mode. Represents 80 bits of data. This is because there are 40 TX (and 40 RX) data signals per channel in the AIB interface and it is designed to work at DDR rates (requiring two bits per signal per clock period) whereas the MAC works at SDR rates (providing one bit per signal per clock). Even when the AIB interface works at SDR rates and the bandwidth of the MAC and PHY are matched, the unit of information exchanged remains an 80-bit word with the odd bits being unused.

VDDC	AIB core (digital) voltage (0.85 V)
VDDTX	AIB low-voltage IO (analog) voltage (Gen2-0.4V, Gen1-0.85V).

Table 104: Glossary

10 References

1. **Intel Corporation.** Advanced Interface Bus (AIB) Specification. [Online] 02/03/21, Revision 2.0. https://github.com/chipsalliance/AIB-specification/blob/master/AIB_Specification%202_0.pdf.

2. **David Kehlet, Tim Hoang.** Advanced Interface Bus (AIB) Usage Note. [Online] Version 1.2.2, 09/2020. https://github.com/chipsalliance/aib-phy-hardware/blob/master/docs/AIB_Usage_Note_v1_2_3.pdf.

3. **Intel Corporation.** Avalon Interface Specifications (MNL-AVABUSREF). [Online] 12/21/20, Version 20.1. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf.

4. **IEEE.** Standard Test Access Port and Boundary-Scan Architecture. [Online] IEEE Std 1149.1-2001. https://standards.ieee.org/standard/1149_1-2001.html.

5.

11 Revision History

Version	Date	Comments
0.1	03/10/21	Initial release for phase 0 review.
0.2	03/26/21	Release to review PowerOnReset with Intel.
0.3	05/26/21	Release to review with contractors.
0.4	06/03/21	Release for phase 1 review.
0.5	06/08/21	Incorporated feedback from phase 1 review.
1	09/24/21	Initial version.
1.1	09/28/21	Filled TBDs regarding TX Adapter path and RX Adapt path.
1.2	10/01/21	Created initial description of CS register and WAM extraction FSM.
1.3	10/11/21	Included data path figure.
1.4	10/13/21	Updated signal names in FIFO figures and removed incorrect serializer/deserializer information.
1.5	10/20/21	Removed out of date content replacing by TBD.
1.6	11/02/21	Internal review to remove incorrect information or description of features which are not completely specified.
1.7	11/05/21	Several improvements based on BCA review of version 1.5 (Nov. 3rd, 2021).
1.8	12/07/21	Updated "*far side loopback implementation"* and "*near side loop back implementation"* figures. Updated "*Data Paths and Clock Trees"* and "*Digital portion of Rx Data Path"* figures. Reviewed specification from analog perspective.
1.9	12/20/21	Updated spec according to Intel feedback. Add information regarding to tx_phcomp field constraints for proper operation. Fixed far side loopback figure.
1.14	05/02/2022	Included FIFO clock drift information and reset table to indicate logic domain affected by each reset signal. Added reset logic figures. Added information about weak pull-down and weak pull-up configuration.
1.15	05/11/2022	Created chapter "Forwarded Clock Observability".

		Updated far side loopback figure. Updated redundancy table. Added two bits to redundancy registers. Removed some unused DLL trimmer CSR bits.
1.16	04/01/2022	Updated AIB IO interface chapter. Updated calibration chapter. Updated “weak pull-down and weak-up control registers” table. Created “Near side loopback mode” table.
1.17	09/26/2022	Added note to tx_rst field description of TX BERT Control register. Added note to rx_rst field description of RX BERT Control register. Updated “Forwarded Clock Observability” section to add more details to configuration steps. Added notes to reset signals section. Fixed field order in RX DLL 2 register Updated “Latency” chapter. Created chapters to provide more details about ADCs and PVT monitors. Added clock phase adjust FSM workaround to “Calibration” section. Added figure to provide more details about sideband architecture. Fixed “Reset logic of calibration state machine” figure.
1.18	10/19/2022	Added IO macro states and controls
1.19	10/25/2022	Updated simplified RCOMP section and removed the full FSM description Added input cap value
1.20	10/27/2022	Added table to call out mismatch in field names between RTL and spec Fixed page numbers Fixed section numbering
1.21	11/03/2022	Added power sequencing section Added input clock jitter constraint

1.22	11/04/2022	Update input cap to include AIB 2.0 standard
1.23	02/02/2024	Updated for open source release

Table 105: Revision History

12 Addendum A: Redundancy Map

In the following diagram, the first signal name is the signal on the bump during normal operation. The second signal name represents the signal on the bump when redundancy is enabled for this bump or one with a lower (for AIB0 to AIB49) or higher (for AIB52 to AIB101) index.

	1	2	3	4	5	6	7	8	9	10	11	12
A	TX[39] None		TX[38] None		TX[37] TX[39]		TX[36] TX[38]		TX[35] TX[37]		TX[34] TX[36]	
B		TX[29] TX[31]		TX[28] TX[30]		TX[31] TX[33]		TX[30] TX[32]		TX[33] TX[35]		TX[32] TX[34]
C	TX[27] TX[31]		TX[26] TX[28]		TX[25] TX[27]		TX[24] TX[26]		TX[23] TX[25]		TX[22] TX[24]	
D		TX[17] TX[19]		TX[16] TX[18]		TX[19] TX[21]		TX[18] TX[20]		TX[21] TX[23]		TX[20] TX[22]
E	TX[15] TX[17]		TX[14] TX[16]		TX[13] TX[15]		TX[12] TX[14]		TX[11] TX[13]		TX[10] TX[12]	
F		TX[7] TX[9]		TX[6] TX[8]		TX[9] ns_fwd_clkb		TX[8] ns_fwd_clk		ns_fwd_clkb TX[11]		ns_fwd_clk TX[10]
G	TX[5] TX[7]		TX[4] TX[6]		TX[3] TX[5]		TX[2] TX[4]		TX[1] TX[3]		TX[0] TX[2]	
H		ns_sr_load ns_sr_clkb		ns_sr_data ns_sr_clk		ns_sr_clkb ns_rcv_clkb		ns_sr_clk ns_rcv_clk		ns_rcv_clkb TX[1]		ns_rcv_clk TX[0]
I	ns_adapter_rstn ns_sr_load		ns_mac_rdy ns_sr_data		spare[1] fs_mac_rdy ns_adapter_rstn		spare[0] fs_adapter_rstn ns_mac_rdy		fs_mac_rdy fs_sr_data		fs_adapter_rstn fs_sr_load	
J		fs_rcv_clk RX[0]		fs_rcv_clkb RX[1]		fs_sr_clk fs_rcv_clk		fs_sr_clkb fs_rcv_clkb		fs_sr_data fs_sr_clk		fs_sr_load fs_sr_clkb
K	RX[0] RX[2]		RX[1] RX[3]		RX[2] RX[4]		RX[3] RX[5]		RX[4] RX[6]		RX[5] RX[7]	
L		fs_fwd_clk RX[10]		fs_fwd_clkb RX[11]		RX[8] fs_fwd_clk		RX[9] fs_fwd_clkb		RX[6] RX[8]		RX[7] RX[9]
M	RX[10] RX[12]		RX[11] RX[13]		RX[12] RX[14]		RX[13] RX[15]		RX[14] RX[16]		RX[15] RX[17]	
N		RX[20] RX[22]		RX[21] RX[23]		RX[18] RX[20]		RX[19] RX[21]		RX[16] RX[18]		RX[17] RS[19]
O	RX[22] RX[24]		RX[23] RX[25]		RX[24] RX[26]		RX[25] RX[27]		RX[26] RX[28]		RX[27] RX[29]	
P		RX[32] RX[34]		RX[33] RX[35]		RX[30] RX[32]		RX[31] RX[33]		RX[28] RX[30]		RX[29] RS[31]
Q	RX[34] RX[36]		RX[35] RX[37]		RX[36] RX[38]		RX[37] RX[39]		RX[38] None		RX[39] None	

Table 106: Redundancy Map