HB0861
Handbook
CoreRxIODBitAlign v2.2

50200861 3.0 2/21

Contents

1 Revision History . . . 1
- 1.1 Revision 3.0 . . . 1
- 1.2 Revision 2.0 . . . 1
- 1.3 Revision 1.0 . . . 1

2 Introduction . . . 2
- 2.1 Overview . . . 2
- 2.2 Features . . . 2
- 2.3 Core Version . . . 2
- 2.4 Supported Families . . . 2
- 2.5 Device Utilization and Performance . . . 2

3 Functional Description . . . 3
- 3.1 CoreRxIODBitAlign with Rx IOD Interface . . . 3
  - 3.1.1 Dynamic Re-training Mechanism . . . 4
  - 3.1.2 Hold Mechanism . . . 4
  - 3.1.3 Restart Mechanism . . . 4
  - 3.1.4 Skip Mechanism . . . 4
  - 3.1.5 MIPI based training Mechanism . . . 4
- 3.2 Configuration Parameters . . . 5
- 3.3 Ports . . . 5

4 Timing Diagrams . . . 7
- 4.1 CoreRxIODBitAlign Training Timing Diagram . . . 7

5 Tool Flow . . . 8
- 5.1 License . . . 8
  - 5.1.1 RTL . . . 8
- 5.2 SmartDesign . . . 8
- 5.3 Configuring CoreRxIODBitAlign in SmartDesign . . . 8
- 5.4 Simulation Flows . . . 9
- 5.5 Synthesis in Libero . . . 10
- 5.6 Place-and-Route in Libero . . . 10

6 Testbench . . . 11
- 6.1 User Test-bench . . . 11

7 System Integration . . . 12

8 Design Constraints . . . 13

Figures

Figure 1 CoreRxIODBitAlign Block Diagram 3
Figure 2 Re-training Mechanism Timing Diagram 4
Figure 3 CoreRxIODBitAlign Timing Diagram 7
Figure 4 SmartDesign CoreRxIODBitAlign Instance View 8
Figure 5 Configuring CoreRxIODBitAlign in SmartDesign 9
Figure 6 Example Simulation Subsystem 10
Figure 7 CoreRxIODBitAlign User Test-bench 11
Figure 8 CoreRxIODBitAlign System Integration 12
Figure 9 Constraints Manager - Derive Constraints TAB 13

Tables

Table 1 Device Utilization and Performance 2
Table 2 Parameter/Generic Descriptions 5
Table 3 Input/Output Signals 5
Table 4 Data and Clock Rate 12

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 3.0

Updated the handbook for left and right data eye signals in the top. For additional information, refer Block Diagram (Figure 1, page 3) and Ports (Table 3, page 5). Updated for CoreRxIODBitAlign v2.2.

1.2 Revision 2.0

Updated the Handbook with additional information for the sections: Functional Description and Timing Diagrams. Updated for CoreRxIODBitAlign v2.1.

1.3 Revision 1.0

This is the first publication of this document. Created for CoreRxIODBitAlign v2.0.

2 Introduction

2.1 Overview

This CoreRxIODBitAlign training IP is used in IO gearing blocks. This IP is geared specifically for bit alignment independent of the data or protocol being used. The CoreRxIODBitAlign provides controls to add or remove delay from the data path relative to the clock path.

2.2 Features

CoreRxIODBitAlign supports the following features:

- Supports Bit Alignment with different eye widths 1-7
- Supports different Fabric DDR modes 2/4/3p5/5
- Supports Skip, Restart / Hold mechanism
- Supports MIPI training through LP signaling start of frame
- Supports 256 Tap delays for bit alignment

2.3 Core Version

This handbook is for CoreRxIODBitAlign version 2.2.

2.4 Supported Families

- PolarFire® SoC
- PolarFire®

2.5 Device Utilization and Performance

The tables in this section are example only and might not be applicable to some cores or family of cores. More examples will be added in the next update.

Table 1 • **Device Utilization and Performance**

	FPGA Resources			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total (%)**	**Frequency (MHz)**
PolarFire	768	1056	1824	MPF500T	0.19	300
PolarFire SoC	768	1055	1823	MPFS250T	0.36	300

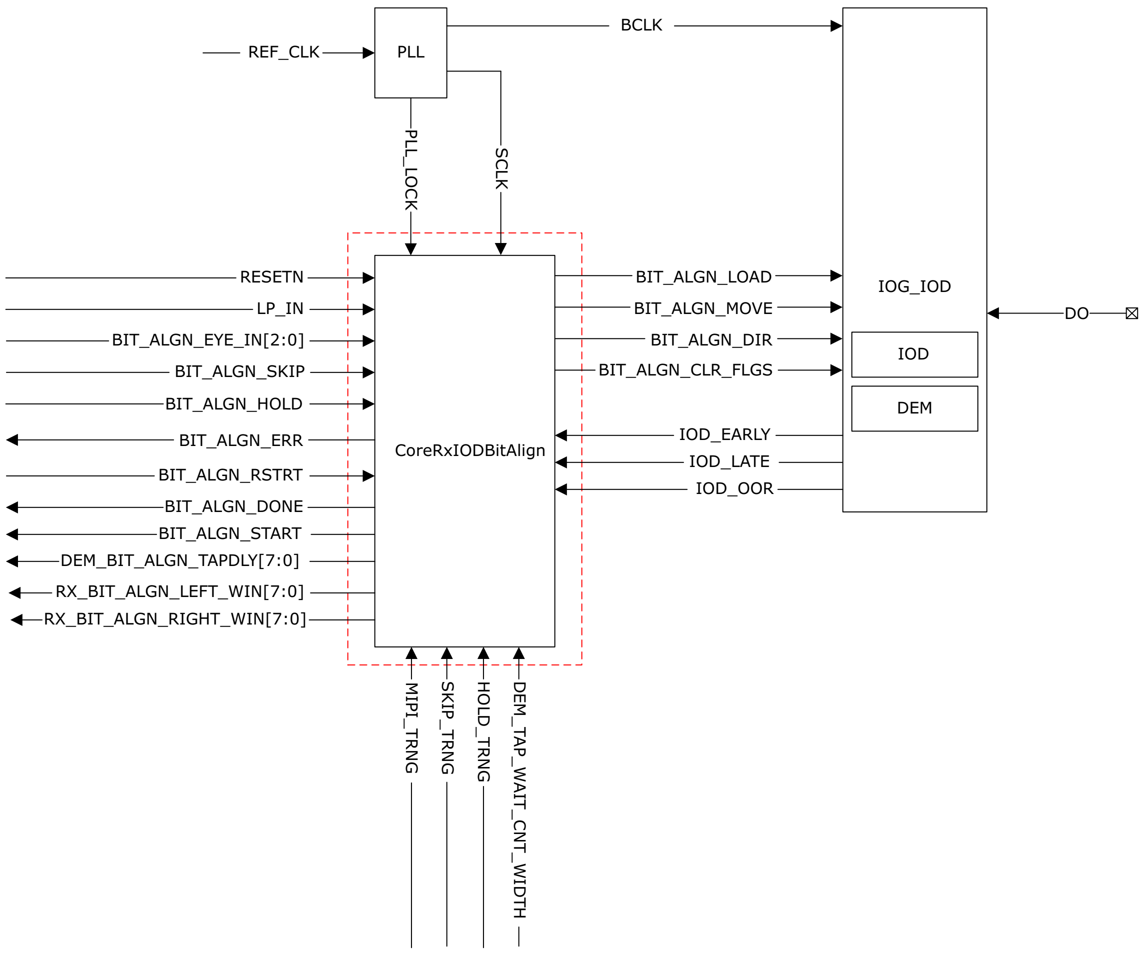
Note: Data in this table is gathered using typical synthesis and layout settings. Throughput is computed as follows: (Bit width / Number of cycles) × Clock Rate (Performance).

3 Functional Description

3.1 CoreRxIODBitAlign with Rx IOD Interface

The following figure shows a high-level block diagram of CoreRxIODBitAlign:

Figure 1 • **CoreRxIODBitAlign Block Diagram**



The description refers to the CoreRxIODBitAlign supporting PolarFire and PolarFire SoC devices. CoreRxIODBitAlign performs training and is also responsible for interfacing IOD devices and IOG to support as a dynamic source with adjusting delays to capture the data correctly. The complete training mechanism flow is explained in the Timing Diagrams, page 7 section.

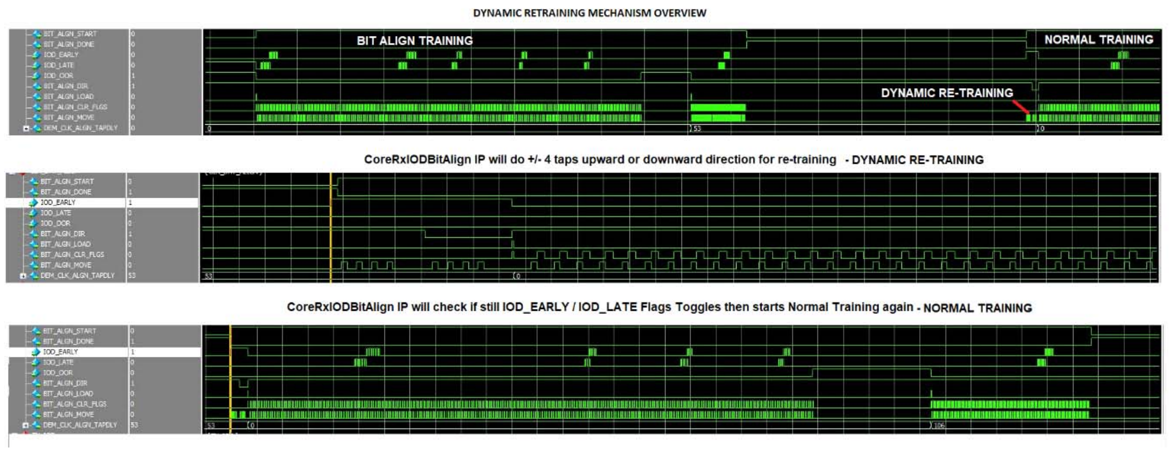
CoreRxIODBitAlign dynamically supports to add or remove delay from the data path relative to the clock path. Here RX_DDRX_DYN interface provides controls to the CoreRxIODBitAlign to perform the clock to data margin training by adding tap delays in upward direction. The CoreRxIODBitAlign in turn for later review (of each tap delay increment) will store the feedback status flags from RX_DDRX_DYN Interface. The CoreRxIODBitAlign will continue the training for every tap increment until the RX_DDRX_DYN Interface reaches the out of range condition Finally, The CoreRxIODBitAlign will sweep the complete feedback status flags. This step is mainly to optimize and calculate the bit alignment of the data to be 90 degrees centered from the clock edges. The final calculated tap delays are loaded in RX_DDRX_DYN Interface for completion of the bit alignment training.

The features supported by this CoreRxIODBitAlign are listed in detail as follows.

3.1.1 Dynamic Re-training Mechanism

CoreRxIODBitAlign continuously monitors the feedback status flags (IOD_EARLY/ IOD_LATE) and check if the flags are toggling. The IP firstly adjusts the previously calculated taps by +/- 4 taps in upward or downward direction. Even then if the flags toggle, the IP re-triggers the training again.

Figure 2 • **Re-training Mechanism Timing Diagram**



3.1.2 Hold Mechanism

This feature is used when the training needs to be on Hold state. The BIT_ALGN_HOLD is active high level based input and should be asserted to hold and de-asserted to continue the training. The HOLD_TRNG parameter should be set to 1 in the configurator to enable this feature. This parameter is set to 0 by default.

3.1.3 Restart Mechanism

This feature is used to restart the training. The BIT_ALGN_RSTRT input should be asserted for one clock pulse (SCLK) to restart the training. This will initiate the soft reset of the IP, which will reset BIT_ALGN_DONE to 0 and BIT_ALGN_START to 1.

3.1.4 Skip Mechanism

This feature is used when the training is not required and the complete training can be bypassed The BIT_ALGN_SKIP is active high level based input and should be asserted to skip the complete training. The SKIP_TRNG parameter should be set to 1 in the configurator to enable this feature. This parameter is set to 0 by default.

3.1.5 MIPI based training Mechanism

The MIPI_TRNG parameter should be set to 1 in the configurator to enable this feature. If set, then LP_IN input port is added to the CoreRxIODBitAlign. The IP detects the falling edge of LP_IN input port which indicates the valid start of frame to start the training.

3.2 Configuration Parameters

The following table shows the parameters and supported features of CoreRxIODBitAlign:

Table 2 • **Parameter/Generic Descriptions**

Parameter Name	Valid Range	Default	Description
SKIP_TRNG	0-1	0	0 – Skip All Training (Bypass Mode) 1 – Perform training as usual
HOLD_TRNG	0-1	0	0 – Hold Training 1 – Release Training to continue
MIPI_TRNG	0-1	0	0 – Normal Training 1 – MIPI based training, LP_IN input (Active low) added to the IP for frame start and end
DEM_TAP_CNT_WIDTH	3-7	3	3 – Width of the Tap delay counter (Default) **Note:** This delay count corresponds to the wait delay of the IP for each tap and so as to record the early late flags. Say the width is set as 3 then 7 clocks are used as a delay counter in the IP.

3.3 Ports

The following table shows the input and output signals used in the design of CoreRxIODBitAlign:

Table 3 • **Input/Output Signals**

Signal	Input/Output	Port width, bits	Description
Clocks and Reset			
SCLK	Input	1	Fabric clock.
PLL_LOCK	Input	1	PLL Lock.
RESETN	Input	1	Active low Asynchronous Reset.
Data Bus and Control			
IOD_EARLY	Input	1	Data eye monitor early flag.
IOD_LATE	Input	1	Data eye monitor late flag.
IOD_ OOR	Input	1	Data eye monitor out of range flag for delay line.
BIT_ALGN_EYE_IN	Input	3	User sets the data eye monitor width.
BIT_ALGN_RSTRT	Input	1	Bit Align Training restart (Pulse based Assertion). 1 – Restart Training 0 – No Restart Training
BIT_ALGN_CLR_FLGS	Output	1	Clear Early/Late flags.
BIT_ALGN_LOAD	Output	1	Load default.
BIT_ALGN_DIR	Output	1	Delay line up/down direction. 1 – Up (increment 1 tap) 0 – Down (decrement 1 tap)
BIT_ALGN_MOVE	Output	1	Increment the delay on move pulse.

Table 3 • **Input/Output Signals *(continued)***

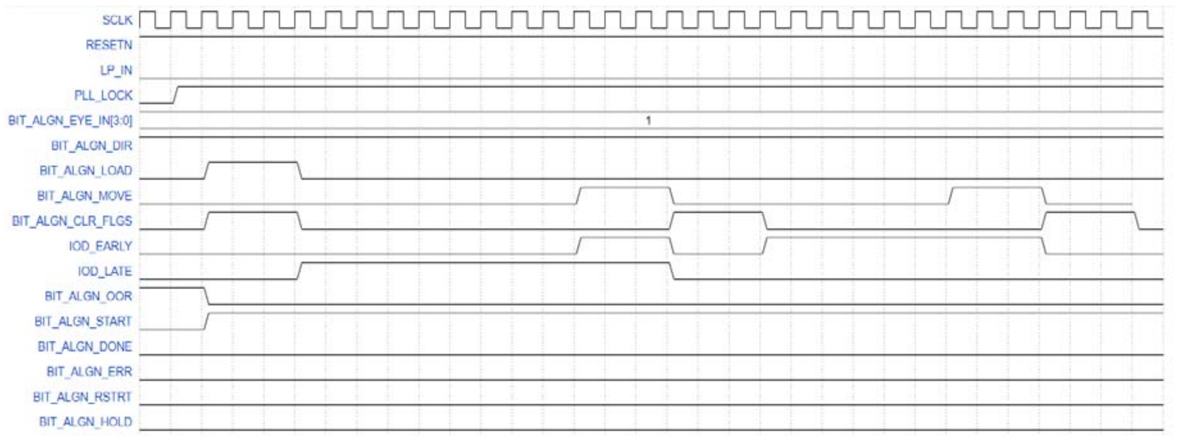
Signal	Input/Output	Port width, bits	Description
Clocks and Reset			
BIT_ALIGN_SKIP	Input	1	Bit Align Training skip (Level based Assertion) 1 – Skip the training and valid only when SKIP_TRNG parameter is set to 1 0 – Training should proceed as normal
BIT_ALIGN_HOLD	Input	1	Bit Align Training hold (Level based Assertion). 1 – Hold the training and valid only when HOLD_TRNG parameter is set to 1 0 – Training should proceed as normal
BIT_ALIGN_ERR	Output	1	Bit Align Training error (Level based Assertion). 1 – Error 0 – No Error
BIT_ALGN_START	Output	1	Bit Align Training start (Level based Assertion). 1 – Started 0 – Not Started
BIT_ALGN_DONE	Output	1	Bit Align Training done (Level based Assertion). 1 – Completed 0 – Not Completed
LP_IN	Input	1	MIPI based frame training (Level based Assertion). 1 – Active low signal should assert low to indicate the start of frame and should de-assert only at the end of the frame. 0 – Training should proceed as normal and this signal will be tied low internally.
DEM_BIT_ALGN_TAPDLY	Output	8	Calculated TAP delays and valid once BIT_ALGN_DONE is set high by the IP.
RX_BIT_ALIGN_LEFT_WIN	Output	8	Left Data Eye Monitor value **Note:** The values are valid only when the output BIT_ALGN_DONE is set to 1 and the output BIT_ALGN_START is set to 0. If the parameter SKIP_TRNG is set then it will return 0.
RX_BIT_ALIGN_RGHT_WIN	Output	8	Right Data Eye Monitor value **Note:** The values are valid only when the output BIT_ALGN_DONE is set to 1 and the output BIT_ALGN_START is set to 0. If the parameter SKIP_TRNG is set then it will return 0.

4 Timing Diagrams

4.1 CoreRxIODBitAlign Training Timing Diagram

The following timing diagram is an example of training sequence with the following parameters:

Figure 3 • CoreRxIODBitAlign Timing Diagram



CoreRxIODBitAlign works based on Fabric clock or SCLK or OUT2_FABCLK_* from CCC or PLL component and PF_IOD_GENERIC_RX IOD component used works based on OUT*_HS_IO_CLK_* or Bank clock or BCLK for bit alignment. Here the PF_IOD_GENERIC_RX IOD component will receive the serial data for bit alignment. For example, the required data rate is 1000 mbps at DDRx4 fabric mode then the OUT2_FABCLK_0 or SCLK should be driven from the PLL or CCC component as 125 MHz and OUT0_HS_IO_CLK_0 or BCLK to PF_IOD_GENERIC_RX will be 500 MHz.

CoreRxIODBitAlign starts the training once the PLL_LOCK is stable and driven high. Then the start of training by driving BIT_ALGN_START high and BIT_ALGN_DONE as low and then drives the output BIT_ALGN_LOAD to load the default settings in the PF_IOD_GENERIC_RX component. The BIT_ALGN_CLR_FLGS is used to clear the IOD_EARLY, IOD_LATE and BIT_ALGN_OOR flags.

CoreRxIODBitAlign proceeds with BIT_ALGN_MOVE followed with BIT_ALGN_CLR_FLGS for every TAP and records the IOD_EARLY, IOD_LATE flags. Once BIT_ALGN_OOR is set high by the PF_IOD_GENERIC_RX component then CoreRxIODBitAlign sweeps the recorded EARLY and LATE flags and finds the optimal EARLY and LATE flags to calculate the required TAP delays for clock and data bit alignment.

CoreRxIODBitAlign loads the calculated TAP delays and drives BIT_ALGN_START low and BIT_ALGN_DONE high to indicate the completion of the training.

CoreRxIODBitAlign continues the Re-training dynamically if it detects noisy IOD_EARLY or IOD_LATE feedback assertion from PF_IOD_GENERIC_RX component. Here the BIT_ALGN_DONE is reset and driven low and BIT_ALGN_START is driven high again by CoreRxIODBitAlign to indicate the restart of the training. The timeout counter once when reaches the timeout condition asserts the BIT_ALGN_ERR at the end of the training.

CoreRxIODBitAlign also provides restart mechanism for the end user to restart the training when ever required. The BIT_ALGN_RSTRT input is active high pulse should be driven high say for example 8 clocks. Here the BIT_ALGN_DONE is reset and driven low and BIT_ALGN_START is driven high again by CoreRxIODBitAlign to indicate the fresh start of the training.

CoreRxIODBitAlign also provides hold mechanism to hold the training in the middle. Here the HOLD_TRNG parameter should be set to 1 then CoreRxIODBitAlign uses the BIT_ALGN_HOLD input and actually should assert active high level based until it requires CoreRxIODBitAlign to hold the training and then continues the training once the input BIT_ALGN_HOLD is driven low.

5 Tool Flow

5.1 License

CoreRxIODBitAlign does not require a license.

5.1.1 RTL

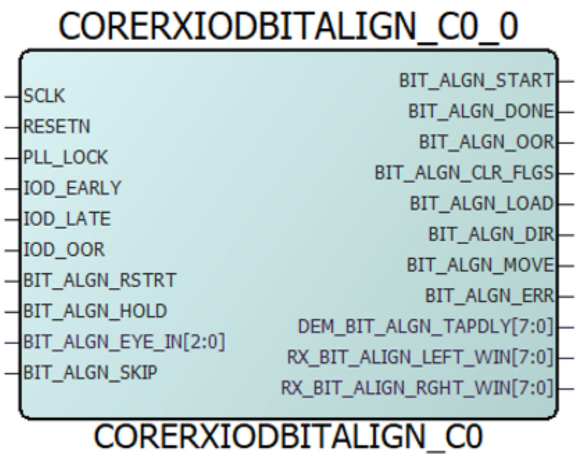
Complete Verilog source code is provided for the core. The core can be instantiated in Verilog or VHDL projects with SmartDesign. Simulation, Synthesis, and Layout performed within Libero SoC v12.0 or higher.

5.2 SmartDesign

CoreRxIODBitAlign is pre-installed in the SmartDesign IP Deployment design environment. Figure 4 shows an example of instantiated CoreRxIODBitAlign. The core can be configured using the configuration window in the SmartDesign, as shown in Figure 5.

For more information on using the SmartDesign to instantiate and generate cores, refer to the Using DirectCore in Libero® SoC User Guide.

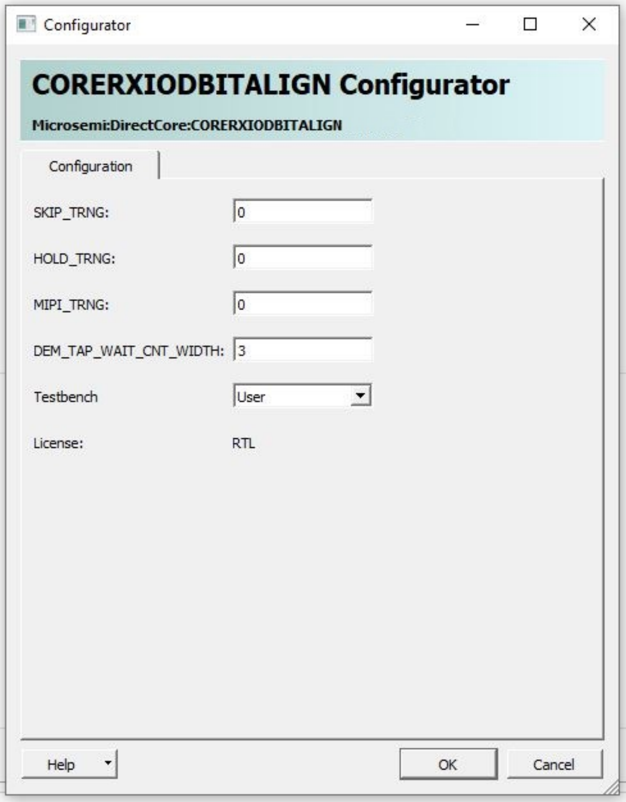
Figure 4 • **SmartDesign CoreRxIODBitAlign Instance View**



5.3 Configuring CoreRxIODBitAlign in SmartDesign

The core is configured using the configuration GUI within SmartDesign.

Figure 5 • **Configuring CoreRxIODBitAlign in SmartDesign**


Configurator
CORERXIODBITALIGN Configurator
Microsemi:DirectCore:CORERXIODBITALIGN
Configuration
Help
OK
Cancel

SKIP_TRNG: 0

HOLD_TRNG: 0

MIPI_TRNG: 0

DEM_TAP_WAIT_CNT_WIDTH: 3

Testbench User

License: RTL

Note: Click **Check Configuration** in the configuration window to verify for valid width x depth FIFO configuration.

5.4 Simulation Flows

The User Testbench for CoreRxIODBitAlign is included in all the releases.

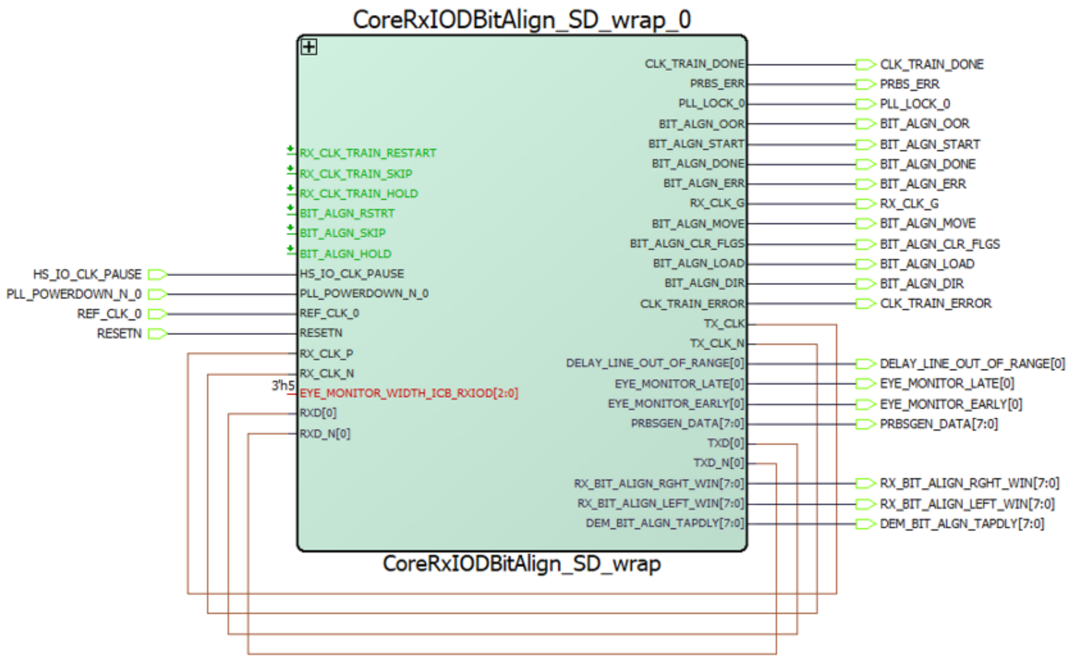
To run simulations, select the **User Testbench** flow in the SmartDesign and click **Save and Generate** on the Generate pane.

The User Testbench is selected through the Core Testbench Configuration GUI. When SmartDesign generates the Libero SoC project, it installs the user testbench files.

To run the user testbench, set the design root to the CoreRxIODBitAlign instantiation in the Libero SoC design hierarchy pane and click **Simulation** in the Libero SoC Design Flow window. This invokes ModelSim® and automatically runs the simulation.

The following simulation subsystem is an example using IOG_IOD component DDRX4 and DDTX4 in loop back mode with the CoreRxIODBitAlign for simulation. Here, the PRBS data generated is transmitted by DDTX4 serially to DDRX4 and finally the PRBS checker is used to check the data integrity after training is done.

Figure 6 • **Example Simulation Subsystem**



5.5 Synthesis in Libero

Click **Synthesis** in the Libero software. The Synthesis window displays the Synplify® project. Set Synplify to use the Verilog 2001 standard, if Verilog is used. To run the synthesis, click **Run**.

5.6 Place-and-Route in Libero

Click **Layout** in the Libero software to invoke the Designer. CoreRxIODBitAlign does not require any special place-and-route settings.

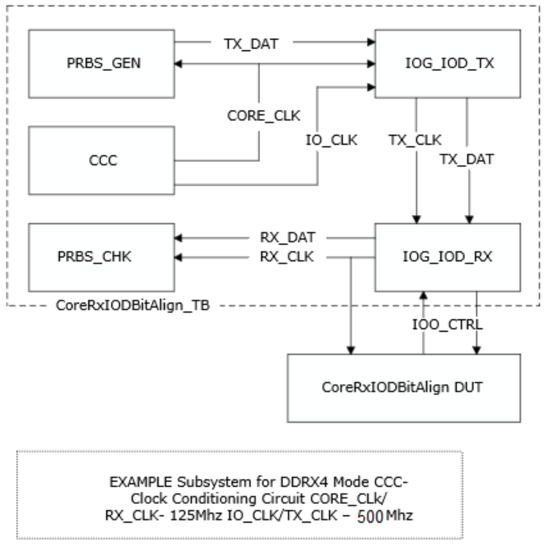
6 Testbench

A unified test-bench verifies and tests CoreRxIODBitAlign, which is called as user test-bench.

6.1 User Test-bench

The user test-bench is included with the releases of CoreRxIODBitAlign that verifies few features of the CoreRxIODBitAlign.

Figure 7 • **CoreRxIODBitAlign User Test-bench**



As shown in Figure 7, the user testbench consists of a Microsemi DirectCore CoreRxIODBitAlign DUT, PRBS_GEN, PRBS_CHK, CCC, IOG_IOD_TX, and IOG_IOD_RX to verify in loop back mode.

The CCC (clock conditioning circuit) drives the CORE_CLK and IO_CLK when clock is stable. PRBS_GEN drives the parallel data to IOG_IOD_TX and then IOG_ID_RX will receive the serial data in parallel the CoreRxIODBitAlign DUT will perform the training with IOD_CTRL signals. Once the training is done the PRBS_CHK block is enabled to check the data from the IOG_IOD_RX block for data integrity.

Note: The user testbench supports fixed configuration only.

7 System Integration

This section hints to ease the integration of CoreRxIODBitAlign.

The Rx/Tx IOG used supports numerous input and output modes. These data and clock rates may be slower and in some cases faster, based on final silicon characterization.

Table 4 • **Data and Clock Rate**

IOG Mode	Direction	Gear Ratio	Max IO Data Rate Expected	IO Clock Rate	Core Clock Rate	Data Type
DDRX4	Input	8:1	1600 Mbps	800 MHz	200 MHz	DDR

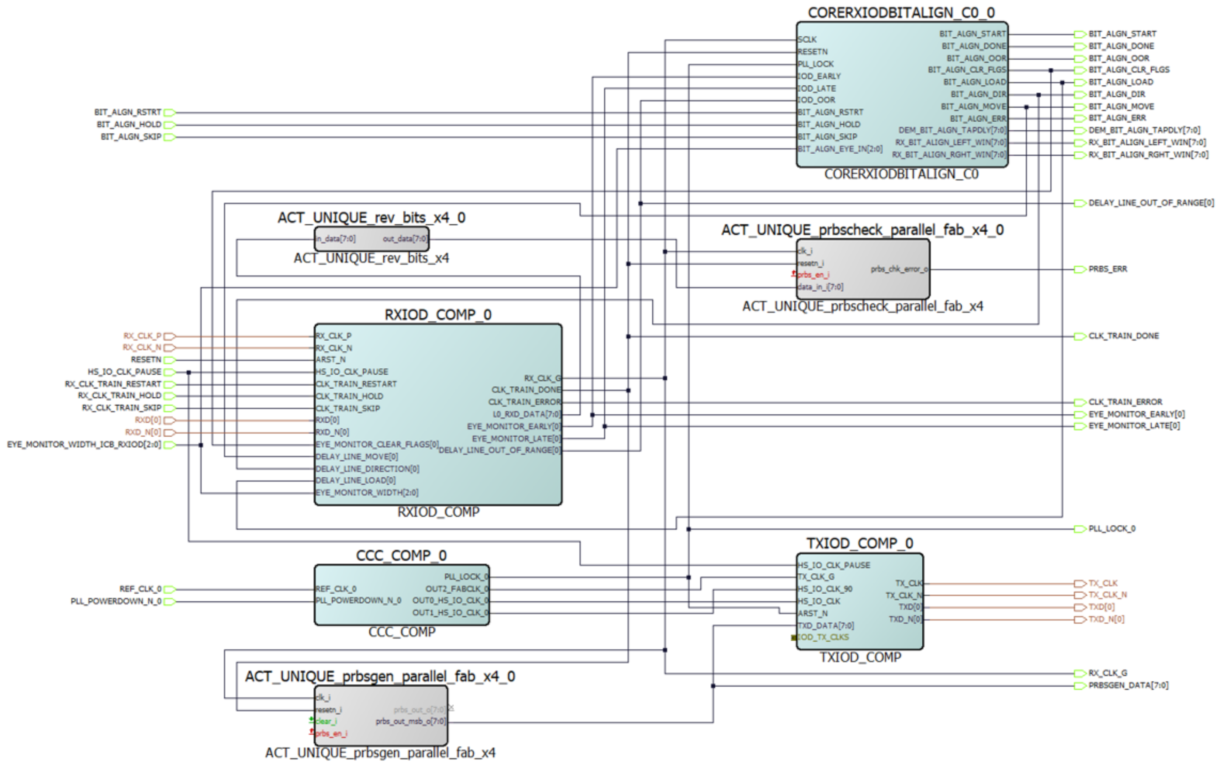


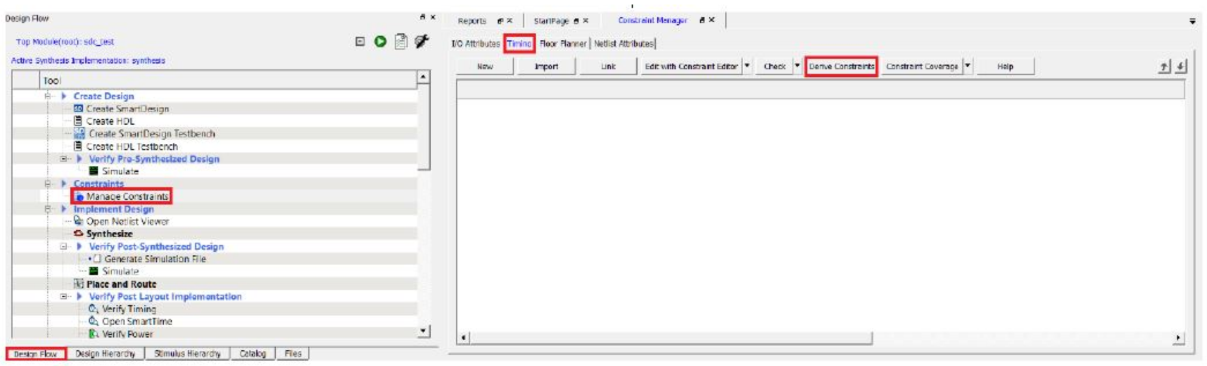
Figure 8 • **CoreRxIODBitAlign System Integration**

The preceding subsystem is an example using IOG_IOD component DDRX4 and DDTX4 in **LOOP BACK** mode with the CoreRxIODBitAlign for simulation. Here the PRBS data generated is transmitted by IOG_IOD_DDRTX4_0 serially to IOG_IOD_DDRX4_PF_0. The CoreRxIODBitAlign does the training (BIT_ALIGN_START set to 1, BIT_ALIGN_DONE set to 0) with the component IOG_IOD_DDRX4_PF_0 and finally once training is done (BIT_ALIGN_START set to 0, BIT_ALIGN_DONE set to 1) the PRBS checker is used to check the data integrity.

8 Design Constraints

To generate these timing constraints, select the **Timing** tab in **Constraint Manager**, and click **Derive Constraints**, as shown in the following figure.

Figure 9 • **Constraints Manager - Derive Constraints TAB**


Design Flow
Top Module(root): sdc_test
Active Synthesis Implementation: synthesis
Tool
Create Design
Create SmartDesign
Create HDL
Create SmartDesign Testbench
Create HDL Testbench
Verify Pre-Synthesized Design
Simulate
Constraints
Manage Constraints
Implement Design
Open Netlist Viewer
Synthesize
Verify Post-Synthesized Design
Generate Simulation File
Simulate
Place and Route
Verify Post Layout Implementation
Verify Timing
Open SmartTime
Verify Power
Design Flow
Design Hierarchy
Stimulus Hierarchy
Catalog
Files
Reports
StartPage
Constraint Manager
I/O Attributes
Timing
Floor Planner
Netlist Attributes
New
Import
Link
Edit with Constraint Editor
Check
Derive Constraints
Constraint Coverage
Help