HB0390
Handbook
CorePCS v3.5

Microsemi
Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

50200390. 7.0 11/17

Contents

Figures

Tables

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 7.0

Updated changes related to CorePCS v3.5.

1.2 Revision 6.0

Updated changes related to CorePCS v3.4.

1.3 Revision 5.0

Updated changes related to CorePCS v3.3.

1.4 Revision 4.0

Updated changes related to CorePCS v3.2.

1.5 Revision 3.0

Updated changes related to CorePCS v3.1.

1.6 Revision 2.0

Updated changes related to CorePCS v3.0.

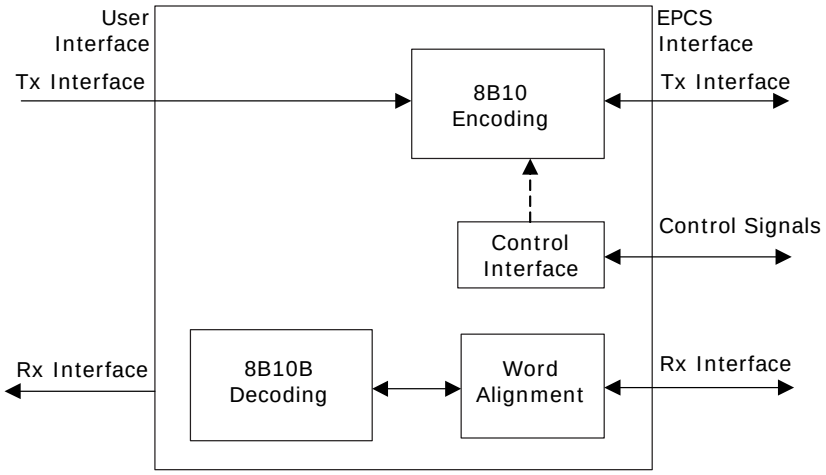
1.7 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CorePCS v2.0.

2 Introduction

CorePCS provides the 8B10B function for the physical coding sublayer for Gigabit Ethernet as defined in the IEEE 802.3z specification. The 8B10B is a marriage of two sub-blocks, the 5b6b and the 3b4b encoder/decoders. The purpose of the encoder/decoders is to convert 8-bit data into a 10-bit code that contains an equal number of 0's and 1's. In addition, the code is built so that no more than five consecutive 0's or 1's is ever transmitted. CorePCS is designed to work directly with a variety of standard transceiver devices. A set of generic signals provides a data and command interface to system logic. CorePCS provides a user interface and a transceiver interface. The user interface consists of transmit data, receive data, and several control and status signals used to qualify the data. The transceiver is responsible for serializing transmit data and deserializing receive data. In addition, the transceiver is designed to resynchronize the serial stream whenever it detects illegal coding errors. 8B10B is commonly used in some protocols not supported by the Microsemi system-on-chip (SoC) high speed serial/deserializer (SERDES) interface, so this core is ideal to use with unsupported protocols. The core can be configured as a transmitter only, receiver only, or both transmitter and receiver. Word alignment support is included in the receiver. The core can be configured to support 10-bit, 20-bit, 40-bit, or 80-bit external physical coding sublayer (EPCS) data. The CorePCS block diagram is as shown in Figure 1, page 2.

Figure 1 • **CorePCS Block Diagram**



2.1 CorePCS Blocks

CorePCS consists of three major blocks, as described below. All signals on the Tx Interface are clocked using EPCS_TxCLK and all signals on the Rx interface are clocked using EPCS_RxCLK.

If the generic/parameter LANE_MODE is set to 0 or 2, the transmitter lane does the following:

2.1.1 8B10B Encoding

When generic/parameter EPCS_DWIDTH is set to 10-bit, the 8b10b transmitter converts 8-bit command or data information into one 10-bit encoded values. Command and data information are qualified by the TX_K_CHAR bus. The data on the TX_DATA bus is continuously registered into the transmitter. Because of the pipe-lined nature of the transmitter, the first encoded data will be driven on the TX_DATA bus several cycles after it is registered into the transmitter. All data input information is valid; however, command possibilities are limited. If the transmitter detects a bad command, then it will assert the INVALID_K signal. The core of the transmitter consists of a data encoder, a command encoder, and a disparity calculator. Each encoder calculates a 4B and 6B code for the input data. The correct code, command or data, is then selected based on the original input value of TX_K_CHAR. The disparity calculator determines whether the encoded value needs to be inverted to maintain the correct running disparity. The input FORCE_DISP can be used to force the data being registered into the transmitter to a selected running disparity. The input DISP_SEL is used to select the running disparity. Finally, the code is registered and sent to the transceiver on the TX_DATA bus.

When generic/parameter EPCS_DWIDTH is set to 20-bit or greater, the 8b10b transmitter is a pipe-lined structure that converts parallel command or data information into parallel encoded values. Command and data information are qualified by the TX_K_CHAR[IO_SIZE-1:0] bus. TX_K_CHAR[IO_SIZE-1] corresponds to the upper data byte on TX_DATA[ENDEC_DWIDTH-1:0] and TX_K_CHAR[0] is for the lower byte. The data on the TX_DATA bus is continuously registered into the transmitter. The transmitter will encode and send the upper byte first followed by the lower bytes. Because of the pipe-lined nature of the transmitter, the first encoded data will be driven on the TX_DATA bus several cycles after it is registered into the transmitter. All data input information is valid; however, command possibilities are limited. If the transmitter detects a bad command, then it will assert the INVALID_K signal. The core of the transmitter consists of a data encoder, a command encoder, and a disparity calculator. Each encoder calculates a 4B and 6B code for the input data. The correct code, command or data, is then selected based on the original input value of TX_K_CHAR. The disparity calculator determines whether the encoded value needs to be inverted to maintain the correct running disparity. The input FORCE_DISP[IO_SIZE-1:0] can be used to forced the data being registered into the transmitter to a selected running disparity. The input DISP_SEL[IO_SIZE-1:0] is used to select the running disparity. FORCE_DISP [IO_SIZE-1] and DISP_SEL [IO_SIZE-1] corresponds to the upper data byte on TX_DATA [ENDEC_DWIDTH-1:0] while FORCE_DISP [0] and DISP_SEL [0] is for the lower byte. Finally, the code is registered and sent to the transceiver on the TX_DATA bus.

If generic/parameter LANE_MODE is set to 1 or 2 the receiver lane does the following:

2.1.2 8B10B Decoding and Word Alignment

When generic/parameter EPCS_DWIDTH is set to 10-bit, the 8b10b receiver converts one 10-bit encoded values and converts it to 8-bit command or data information. Command information is indicated by the RX_K_CHAR signal asserted high.

When generic/parameter EPCS_DWIDTH is set to 20-bit or greater, the 8b10b receiver is a pipe-lined structure that converts parallel 10-bit encoded values and converts them to parallel command or data information. Command information is indicated by the RX_K_CHAR [IO_SIZE-1:0] bus signals asserted high. The data on the upper byte of the RX_DATA bus is the first decoded value in the sequence. Receive data is first registered into parallel registers. The codes are decoded in parallel moving from stage to stage.

Several signals qualify the validity of the information on RX_DATA. RX_DATA contains good information whenever both the CODE_ERR_N is inactive (high) and ALIGNED is active (high). If ALIGNED is low or CODE_ERR_N is low, then some problem exists in the transmission. Whenever the receiver loses sync (ALIGNED is low) the transceiver will resynchronize the data on subsequent COMMA commands (K28.1 and/or K28.5 and/or K28.7). When sync is re-established, the ALIGNED will again be driven high after the pipeline has been flushed of potentially bad data. The error check block monitors the incoming codes and checks for illegal codes and bad running disparity. Whenever an error in the 8b10b code is detected, the CODE_ERR_N is asserted. If several codes in a row are received with errors, then the 8b10b will assume that synchronization with the transceiver has been lost and will deactivate ALIGNED and assert the COMMA_DET_EN signal. The number of consecutive errors required to force a resynchronization is

fixed to 4. The transceiver will then resynchronize the data using COMMA codes. The 8b10b responds by asserting ALIGNED indicating that the transceiver has reacquired sync.

The CorePCS supports word alignment for COMMA characters K28.1, K28.5, and K28.7. Word alignment must be performed in the receiver before the data can make it through the core. Word alignment is achieved by transmitting a burst of consecutive COMMA characters before transmitting the data, however the number of consecutive COMMA characters is configurable when PROG_COMMA_EN is enabled.

Note:

1. If parameter/generic PROG_COMMA_EN is disabled, an initial stream of 10 or more consecutive COMMA characters will guarantee word alignment. If the receiver loses word alignment another stream of 10 or more consecutive COMMA characters will re-align the receiver channel.
2. If parameter/generic PROG_COMMA_EN is enabled, the number of COMMAs required for word alignment can be configured to a value between 1 and 2048. However, the word alignment is not guaranteed when parameter/generic NO_OF_COMMAS is set to 1, so it is required that the user has their own detection outside this core. This is due to the fact that bit boundary mismatch can occur in multiple scenarios. Lower detection was added to allow support for protocols do not have a stream of consecutive COMMA characters (For example, one COMMA character in a four symbol idle pattern).

2.2 Features

Following are the key features:

- 8B10B encoding in the transmitter lane.
- Word alignment and 8B10B decoding in the receiver lane.
- EPCS data width of 10, 20, 40, or 80 bits can be selected, depending on the EPCS_DWIDTH parameter/generic.
- The core can be configured as a transmitter only, receiver only, or both transmitter and receiver, depending on the LANE_MODE parameter/generic.
- Supports 8, 16, 32, or 64 bit word alignment, depending on the SHIFT_EN parameter/generic.
- Fixed or configurable COMMA character detection options for word alignment.

2.3 Core Version

This handbook is for CorePCS version 3.5.

2.4 Supported Families

This version of CorePCS supports the following FPGA families:

- PolarFire™
- RTG4™
- IGLOO®2
- SmartFusion®2

2.5 Supported Interface

CorePCS includes the following interface:

- External PCS interface

2.6 Device Utilization and Performance

CorePCS has been implemented in several Microsemi device families. A summary of the implementation data for CorePCS is listed in Table 1, page 5.

Table 1 • **CorePCS Device Utilization and Performance for 16-bit mode**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total %**	**Performance MHz**
PolarFire	795	936	1731	MPF300	0.58	300 MHz
RTG4	949	964	1,913	RTG4150	1.26	200 MHz
SmartFusion2	616	894	1,510	M2S050	2.7	250 MHz
IGLOO2	869	1,035	1,904	M2GL150T	1.3	250 MHz

Note:

1. Data in this table were achieved using synthesis and layout settings optimized for speed. Top-level parameters/generics were left at their default values.
2. Performance results were achieved with speed grade -1, speed grade STD will achieve a lower performance frequency.

Table 2 • **CorePCS Device Utilization and Performance for 32-bit mode**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total %**	**Performance MHz**
PolarFire	1620	1680	3300	MPF300	1.1	250 MHz

Table 3 • **CorePCS Device Utilization and Performance for 64-bit mode**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total %**	**Performance MHz**
PolarFire	2736	3893	6629	MPF300	2.21	210 MHz

3 Functional Description

CorePCS, shown in Figure 1, page 2 consists of 8B10B encoding for the EPCS transmitter interface and word alignment/8B10B decoding for the EPCS receiver interface.

3.1 Word Alignment Shift Option

Word alignment shift is controlled by the parameter/generic SHIFT_EN. It is required that word alignment shift is enabled when CorePCS is configured for a 20-bit or greater EPCS received channel and it is handling data from protocols with a continuous stream of COMMA characters for word alignment followed by a stream of data which required the 16-bit data received to be in the upper and lower bytes after the last COMMA characters is received. For protocols that do not have this requirement word alignment shift should remain disabled.

For example, a 16-bit ADC transmits a continuous stream of COMMA characters for word alignment followed by 16-bit data. There is no control over when the last COMMA character is received. There are two scenarios for the position of the last COMMA character received will be.

Figure 2, page 6 shows the first scenario. In this case the last COMMA received in the lower position followed by data in the upper position on the next clock tick. This is good for protocols that required 16-bit data received to be in the upper and lower 8-bits after the last COMMA characters because on the next clock tick D0u and D0l are in upper and lower position. This means the 16-bit data will be valid data.

Figure 2 • **Last COMMA Characters Scenario 1**

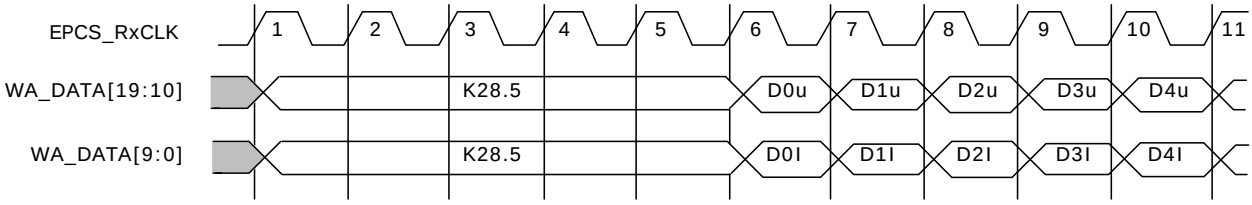
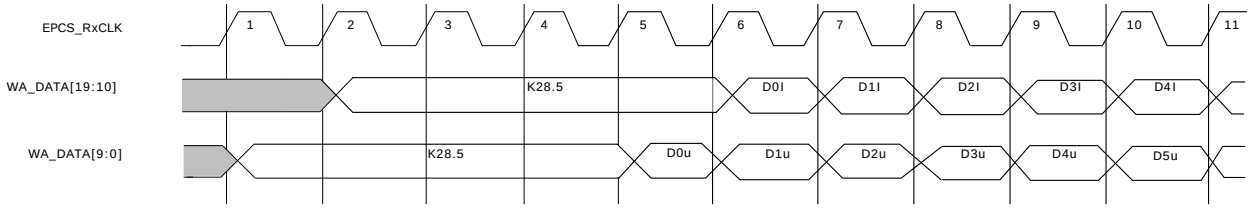


Figure 3, page 6 shows the second scenario. In this case the last COMMA received in the upper position followed by data in the lower position on the same clock tick. This is bad for protocols that required 16-bit data received to be in the upper and lower 8-bits after the last COMMA characters because on the next clock tick D0l and D1u are in upper and lower position. This means the 16-bit data will always shifted by a byte, making it invalid data.

Note: WA_DATA is an internal signal that is the word aligned data from EPCS_RxDATA

Figure 3 • **Last COMMA Characters Scenario 2**

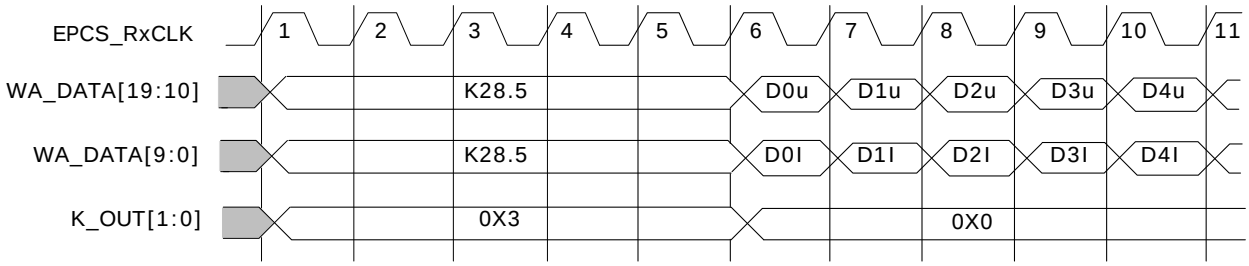


The problem is that we have no control over when the last COMMA character is received so scenario 1 or 2 can happen anytime. This is where word alignment shift control solves this issue, if where have a system that required 16-bit data received to be in the upper and lower 8-bits after the last COMMA characters from word alignment then we would enabled word alignment shift.

With word alignment shift enabled no matter what position the last COMMA character received during word alignment is the core will handle both scenarios and output the correct 16-bit data in the upper and lower position. Figure 4, page 7 shows the position of the data on the output of the 8B10B decoder for both scenarios when word alignment shift enabled.

Note: DATA_16B_OUT is an internal signal that is the decoded value WA_DATA

Figure 4 • **8B10B Decoder Result**



4 Interface

4.1 Verilog/VHDL Parameters

CorePCS has parameters (Verilog) or generics (VHDL) for configuring the register transfer level (RTL) code, described in Table 4, page 8. All parameters and generics are integer types.

Table 4 • **CorePCS Parameters/Generics Descriptions**

Parameter Name	Valid Range	Default	Description
FAMILY	19 to 26	19	Must be set to the required FPGA family: 19: SmartFusion2 24: IGLOO2 25: RTG4 26: PolarFire
LANE_MODE	0 to 2	2	Option to use the core select 8B10B encoding and/or 8B10B decoding: 0: Transmitter only 1: Receiver only 2: Transmitter and Receiver
EPCS_DWIDTH	10, 20, 40 or 80	20	The transmit/receive data bus on EPCS side. This generic/parameter should be set to match the width of the EPCS data width connected to CorePCS from the SERDES.
ENDEC_DWIDTH	8, 16, 32, or 64	16	This generic/parameter sets the width of the data on the user interface side. The user cannot configure its value because it depends on setting of EPCS_DWIDTH: ENDEC_DWIDTH = (EPCS_DWIDTH/10) *8
IO_SIZE	1,2,4 or 8	2	This generic/parameter sets the width of the error and control signals on the user interface side. Its value cannot be configure because it depends on setting of EPCS_DWIDTH: IO_SIZE = EPCS_DWIDTH/10
SHIFT_EN	0	0	This generic/parameter is only used when EPCS_DWIDTH is 20/40/80 bit. If enabled, it will shift the user data to the upper and lower byte afterword alignment; this is useful if 16/32/64 bit data is used. 0: Disabled 1: Enabled **Note:** Connected to a 16-bit ADC is an example of when this functionality should be enabled.
PROG_COMMA_EN	0 to 1		Enable for programmable COMMA pattern match: 0: Disabled 1: Enabled
NO_OF_COMMAS	1 to 2,048		When PROG_COMMA_EN is enabled, the value of this parameter/generic determines the number of consecutive COMMA character patterns is required follow word alignment. **Note:** If set to 1, word alignment must be monitored by some extern block from this core.

Table 4 • **CorePCS Parameters/Generics Descriptions**

COMMA_DETECT_SEL	0 to 2	0	The COMMA character(s) used for word alignment: 0: K28.5 1: K28.1 2: K28.1 or K28.5 or K28.7 3: K28.7

4.2 I/O Signals

The port signals for the CorePCS macro are as shown in Figure 5, page 9 and defined in Table 5, page 9.

Figure 5 • **CorePCS I/O Signal Diagram**

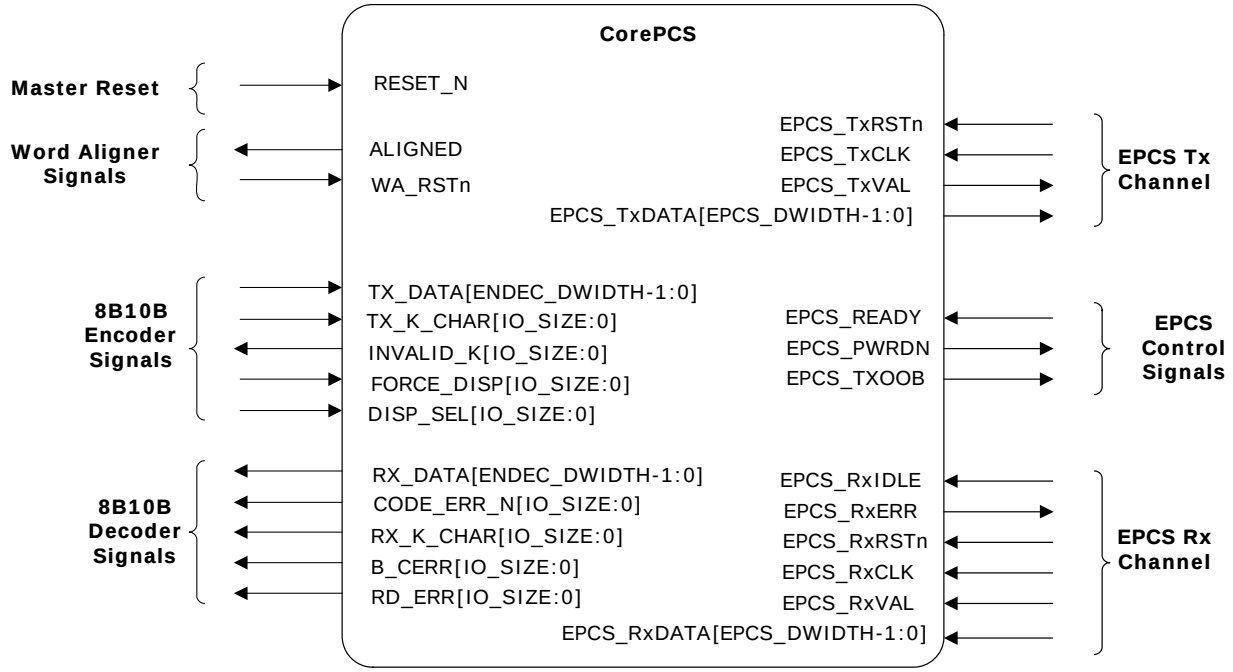


Table 5 • **CorePCS I/O Signal Descriptions**

Port Name	Type	Description
Master Reset		
RESET_N	In	Active low asynchronous reset. This is used to reset the core to its initial state.
8B10B Encoder Ports		
TX_DATA	In	User data which is encoded before being transmitted through the EPCS interface.
TX_K_CHAR	In	Active High signal indicating that the TX_DATA contains command information. Bit 0 corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of TX_DATA.
FORCE_DISP	In	When asserted High the data on TX_DATA is encoded to the disparity encoding of DISP_SEL Bit[0] corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of TX_DATA. **Note:** Only lower byte available when EPCS_DWIDTH = 10

Table 5 • **CorePCS I/O Signal Descriptions**

DISP_SEL	In	Only used when FORCE_DISP is asserted High. 0: Negative running disparity 1: Positive running disparity Bit [0] corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of TX_DATA. **Note:** Only lower byte available when EPCS_DWIDTH = 10
INVALID_K	Out	Active High signal indicating that the transmitter has detected an invalid K character.
8B10B Decoder Ports		
RX_DATA	Out	Decoded data received from the EPCS interface.
RX_K_CHAR	Out	Active high output from the decoder to the receiver indicating that the received data is a command code. Bit 0 corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of RX_DATA.
B_CERR	Out	This active high signal asserts when the received code group is not found in the 8B/10B decoding table for either disparity. Bit 0 corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of RX_DATA.
RD_ERR	Out	This active high signal asserts when the received code group exists in the 8B/10B decoding table, but is not found in the proper column according to the current running disparity. Bit 0 corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of RX_DATA.
CODE_ERR_N	Out	Active low signal indicating that the decoder has detected an error in the received. Bit 0 corresponds to the lower byte and IO_SIZE bit corresponds to the upper byte of RX_DATA.
External PCS Control Interface		
EPCS_READY	In	PHY ready signal. This signal is asserted when the PHY has completed the calibration sequence for each specific lane. This signal can be used in order to release the reset for the external PCS and controller, start transmitting data to the PMA or any other purpose.
EPCS_PWRDN	Out	PHY power-down signal. This signal is used to put the PMA in power-down state, where Rx CDR PLL is bypassed and other low power features are applied to the PMA. When exiting power down, no calibration is required and the link can be operational much faster than using the EPCS_TxOOB or EPCS_RSTn signals.
External PCS Transmit Interface		
EPCS_TxOOB	Out	PHY transmit out-of-band (OOB) signal. This signal is used to load Electrical Idle III in the Tx driver of the PMA macro. It can be used for SATA as part of the sequencing for transmitting very short OOB signaling.
EPCS_TxRSTn	In	PHY clean active low synchronous reset on Tx clock. This signal is a clean version of the EPCS_RSTn signal, which has clean deassertion timing compared to EPCS_TxCLK.
EPCS_TxCLK	In	PHY transmit clock signal. This signal is the aTxClk signal generated by the PMA macro and must be used by the external PCS logic to provide data on EPCS_TxDATA.
EPCS_TxDATA	Out	PHY transmit data signal. This output should match with the width of the EPCS receiver channel on the SERDES.
EPCS_TxVAL	Out	PHY transmit valid signal. This signal is used to transmit valid data. If deasserted, the PMA macro is put in Electrical Idle I. It can be used for protocols requiring Electrical Idle (SATA) and must also be deasserted as long as EPCS_READY is not asserted. This must be generated one clock cycle earlier than corresponding EPCS_TxDATA[] signals.

Table 5 • **CorePCS I/O Signal Descriptions**

External PCS Receive Interface		
EPCS_RxRSTn	In	PHY clean active low synchronous reset on Rx clock. This signal is a clean version of the EPCS_RSTn signal, which has clean deassertion timing compared to EPCS_RxCLK.
EPCS_RxCLK	In	PHY receive clock signal This signal is the aRxClk signal generated by the PMA macro and must be used by the external PCS logic to provide data on EPCS_RxDATA.
EPCS_RxDATA	In	PHY receive data signal. This input should match with the width of the EPCS receiver channel on the SERDES.
EPCS_RxVAL	In	PHY receive valid data signal. This signal is used to signal receive valid data. It corresponds to the two conditions completed by the PMA control logic: - Receiver detect incoming data (not in Electrical Idle) - CDR PLL is locked to input bitstream in fine grain state **Note:** If PMA driven mode is used by the selected protocol (see the CDRPLL_DIS register in reg00), this signal cannot be monitored and the EPCS_RxIDLE signal must be used instead.
EPCS_RxIDLE	In	PHY Receive Idle signal. This signal is used to signal an Electrical Idle condition detected by the PMA control logic. Note that this signal is generated on EPCS_TxCLK of the selected lane.
EPCS_RxERR	Out	PHY Receive Error signal. This signal is used to report to PMA control logic that error data has been detected by the external PCS logic. This signal is used in PCS-driven mode of the CDR PLL to switch back to frequency lock acquisition if too many errors are detected by the PMA control logic. This signal is unused in PMA-driven mode and can also be hardwired to zero if the PCS should rely only on Electrical Idle detection circuitry to switch the CDR PLL back into frequency lock state.
Word Aligner Signals		
WA_RSTn	In	Active low synchronous reset for the word aligner on Rx clock, when asserted the word aligner assumes a loss of alignment on the next clock tick and will start the re-alignment process once de-assertion of this signal is detected.
ALIGNED	Out	When high indicates that the word aligner has achieved alignment. Alignment is achieved by transmitting simultaneous K28.1, K28.5, or K28.7 character.

Note:

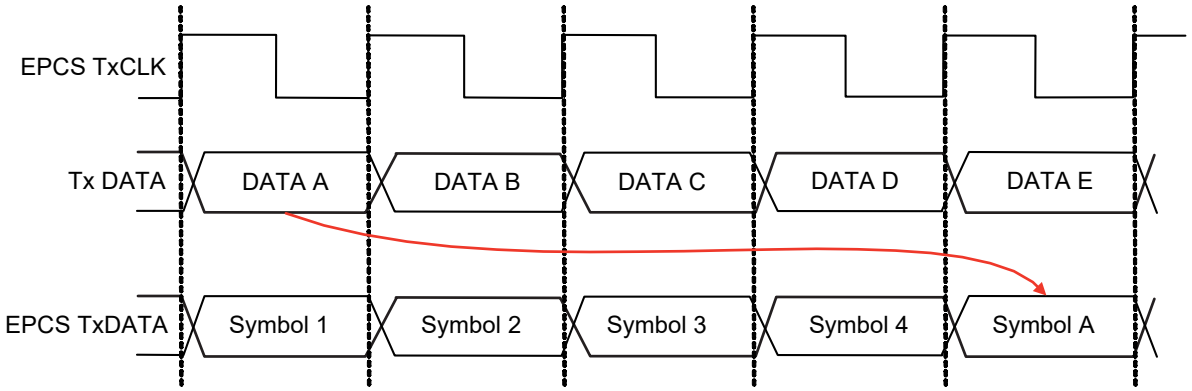
1. All signals are active High (logic 1) unless otherwise noted.
2. EPCS_RxIDLE, EPCS_RxERR, and EPCS_RxVAL are not used by the core but are added inputs to complete the EPCS interface.

5 Timing Diagrams

5.1 Transmitter Lane Timing

CorePCS implements 8B10B encoding across the transmit lane to the external PCS interface.

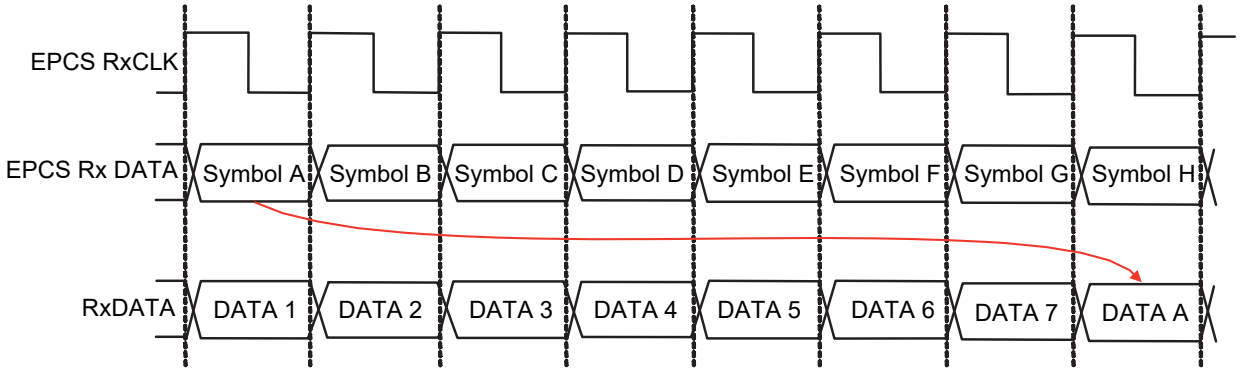
Figure 6 • **Transmitter Lane Timing**



5.2 Receiver Lane Timing

CorePCS implements word alignment and 8B10B decoding across the receive lane from the external PCS interface.

Figure 7 • **Receiver Lane Timing**



6 Tool Flow

6.1 License

CorePCS is license free.

6.1.1 RTL

Complete RTL source code is provided for the core and testbenches.

6.2 SmartDesign

CorePCS is preinstalled in the SmartDesign IP Deployment design environment. The core should be configured using the configuration GUI within SmartDesign, as shown in Figure 8, page 13. For information on using the SmartDesign to instantiate and generate cores, refer to the Using DirectCore in Libero SoC User Guide.

Figure 8 • **CorePCS Full I/O View**

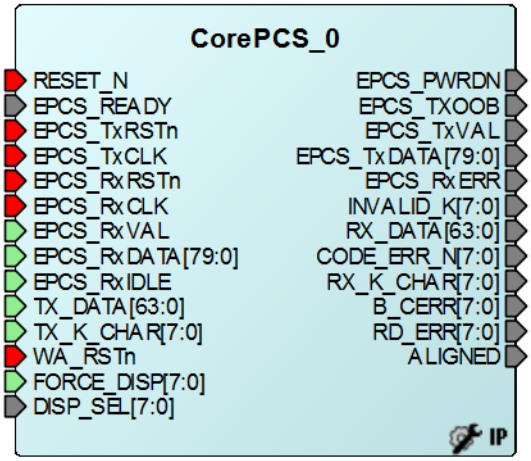
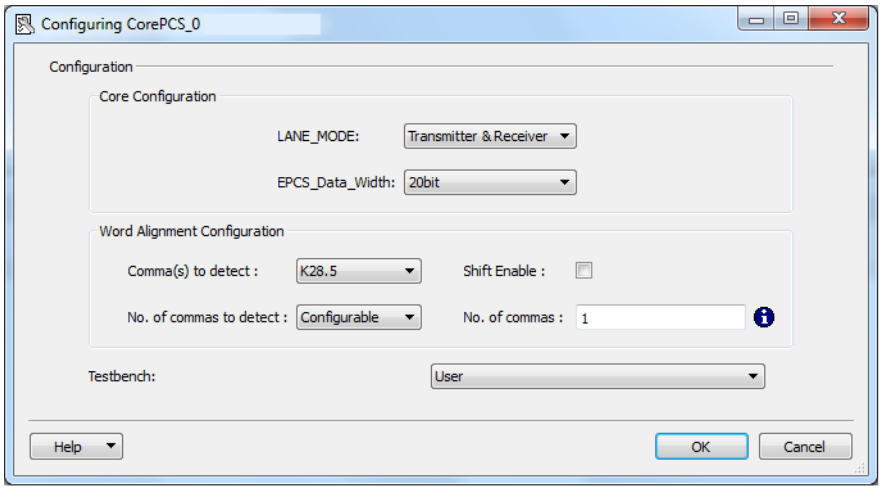


Figure 9 • **CorePCS SmartDesign Configuration Window**



6.3 Simulation Flows

The User Testbench for CorePCS is included in all releases.

To run simulations, select the **User Testbench** flow within **SmartDesign CorePCS** configuration GUI, right-click the canvas, and select **Generate Design**.

When SmartDesign generates the design files, it will install the user testbench files.

To run the user testbench, Set the design root to the CorePCS instantiation in the Libero® System-on-Chip (SoC) design hierarchy pane and click the Simulation icon in the Libero SoC Design Flow window. This will invoke ModelSim® and automatically run the simulation.

6.4 Synthesis in Libero

After setting the design root appropriately for your design, click **Synthesis** in the Libero SoC. The Synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To run Synthesis, click **Run**.

6.5 Place-and-Route in Libero

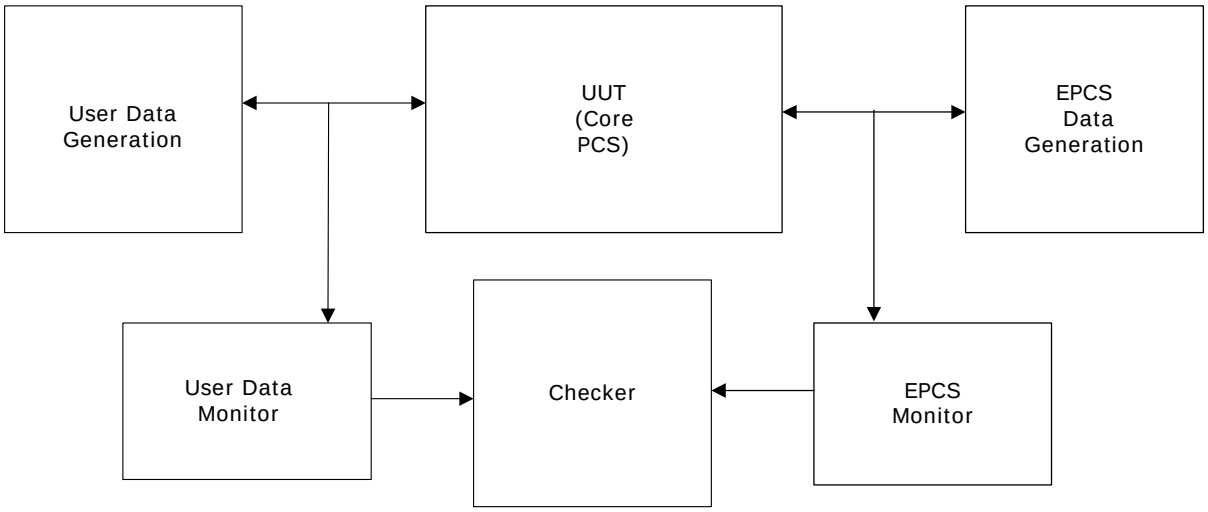
After setting the design root appropriately for the design, and after running Synthesis, click **Layout** in the Libero SoC software to invoke Designer. CorePCS requires no special place-and-route settings.

7 Testbench

7.1 User Testbench

The CorePCS user testbench gives an example of how to use the core with an external PCS. As shown in Figure 10, page 15, the testbench instantiates a User Data Generation block and EPCS Data Generation block to emulate using an external PCS.

Figure 10 • **CorePCS User Testbench**



The simulation testbench is as shown in Figure 10, page 15 includes an instantiation of the CorePCS macro, user data generation, EPCS data generation, user data/EPCS monitors, and data checker. The testbench transmits from user data to EPCS and/or receives from EPCS to user data.