
Appendix A

Sample SERDES Data -- RocketIO X Transceiver Overview

This information was extracted from the RocketIO™ X Transceiver User Guide. For up-to-date information, please go to:

<http://www.xilinx.com/bvdocs/userguides/ug035.pdf>

Basic Architecture and Capabilities

The definitions, descriptions, and recommendations in this user guide reflect Step 1 silicon. For Step 0 silicon, see the Errata for special considerations.

The RocketIO X block diagram is illustrated in Figure 1-1. Depending on the device, a Virtex™-II Pro X FPGA has between 8 and 20 transceiver modules, as shown in Table 1-1.

TABLE 1-1: Number of RocketIO X Cores per Device Type

| Device | RocketIO X Cores |
|----------|------------------|
| XC2VPX20 | 8 |
| XC2VPX70 | 20 |

Definitions:

- **Attribute** – An attribute is a control parameter to configure the RocketIO X transceiver. There are both primitive ports (traditional I/O ports for control and status) and transceiver attributes. Transceiver attributes are also controls to the transceiver that regulate data widths and encoding rules, but controls that are configured as a group in “soft” form through the invocation of a primitive.
- **GT10 Primitive** – A primitive is a predesigned collection of attribute values that accomplish a known data rate, encoding type, data width, and so on. A single primitive invocation, for

example, OC-192 mode which configures all the dozens of pertinent attributes to their correct values in a single step.

The transceiver module is designed to operate at any serial bit rate in the range of 2.488 Gb/s to 10.3125 Gb/s per channel, including the specific bit rates used by the communications standards listed in Table A-1-2, page 111. Data-rate specific attribute settings are set appropriately in the GT10 primitives.

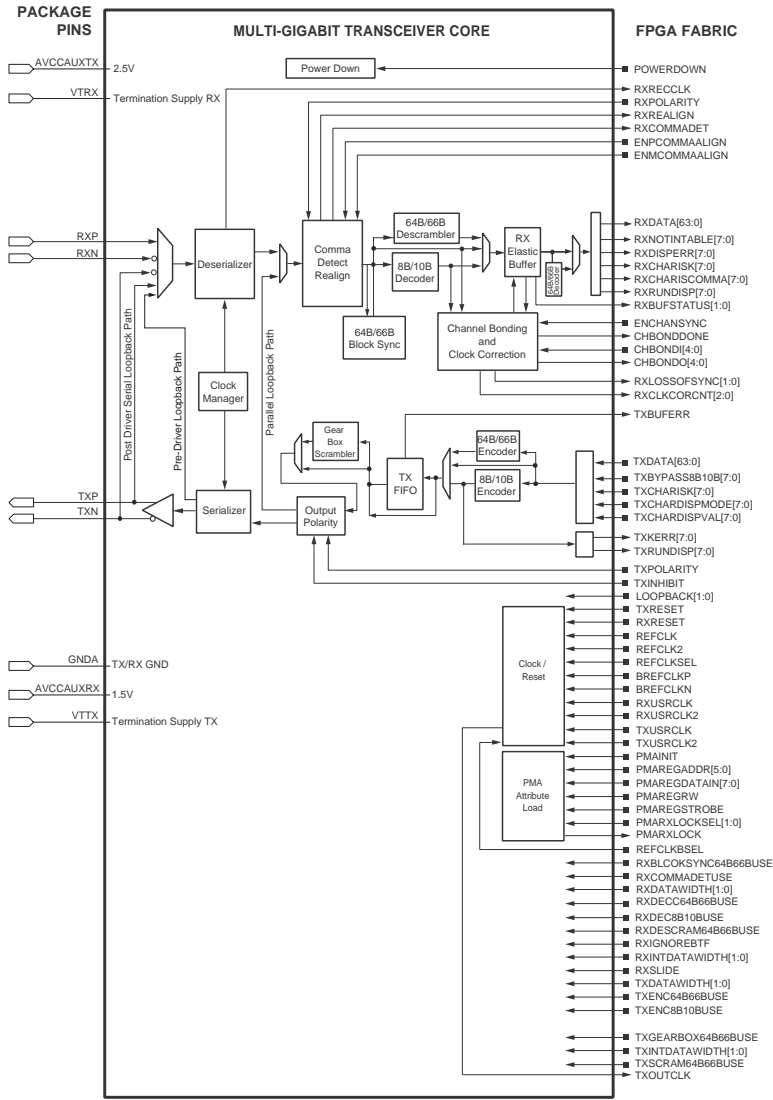


FIGURE 1-1: RocketIO X Transceiver Block Diagram

TABLE 1-2: Communications Standards Supported by RocketIO X Transceiver

| Mode | Channels ⁽¹⁾ (Lanes) | I/O Bit Rate (Gb/s) |
|---------------------------------|---------------------------------|---------------------|
| SONET OC-48 | 1 | 2.488 |
| PCI Express | 1, 2, 4, 8, 16 | 2.5 |
| Infiniband | 1, 4, 12 | 2.5 |
| XAUI (10-Gigabit Ethernet) | 4 | 3.125 |
| XAUI (10-Gigabit Fibre Channel) | 4 | 3.1875 |
| SONET OC-192 ⁽²⁾ | 1 | 9.95328 |
| Aurora (Xilinx protocol) | 1, 2, 3, 4,... | 2.488 – 10.3125 |
| Custom Mode | 1, 2, 3, 4,... | 2.488 – 10.3125 |

Notes:

1. One channel is considered to be one transceiver.
2. See Solution Record 19020 for implementation recommendations.

Table 1-3 lists the transceiver primitives provided. These primitives carry attributes set to default values for the communications protocols listed in Table 1-2. Data widths of one, two, and four bytes (lower speeds) or four and eight bytes (higher speeds) are selectable for the various protocols.

TABLE 1-3: Supported RocketIO X Transceiver Primitives

| Primitive | Description | Primitive | Description |
|--------------------|-------------------------------|---------------|------------------------------------|
| GT10_CUSTOM | Fully customizable by user | GT10_XAUI_4 | 10GE XAUI, 4-byte data path |
| GT10_OC48_1 | SONET OC-48, 1-byte data path | GT10_AURORA_1 | Xilinx protocol, 1-byte data path |
| GT10_OC48_2 | SONET OC-48, 2-byte data path | GT10_AURORA_2 | Xilinx protocol, 2-byte data path |
| GT10_OC48_4 | SONET OC-48, 4-byte data path | GT10_AURORA_4 | Xilinx protocol, 4-byte data path |
| GT10_PCI_EXPRESS_1 | PCI Express, 1-byte data path | GT10_OC192_4 | SONET OC-192, 4-byte data path |
| GT10_PCI_EXPRESS_2 | PCI Express, 2-byte data path | GT10_OC192_8 | SONET OC-192, 8-byte data path |
| GT10_PCI_EXPRESS_4 | PCI Express, 4-byte data path | GT10_10GE_4 | 10-Gbit Ethernet, 4-byte data path |

TABLE 1-3: Supported RocketIO X Transceiver Primitives (*Continued*)

| Primitive | Description | Primitive | Description |
|-------------------|------------------------------|----------------|---|
| GT10_INFINIBAND_1 | Infiniband, 1-byte data path | GT10_10GE_8 | 10-Gbit Ethernet, 8-byte data path |
| GT10_INFINIBAND_2 | Infiniband, 2-byte data path | GT10_10GFC_4 | 10-Gbit Fibre Channel, 4-byte data path |
| GT10_INFINIBAND_4 | Infiniband, 4-byte data path | GT10_10GFC_8 | 10-Gbit Fibre Channel, 8-byte data path |
| GT10_XAUI_1 | 10GE XAUI, 1-byte data path | GT10_AURORAX_4 | Xilinx 10G protocol, 4-byte data path |
| GT10_XAUI_2 | 10GE XAUI, 2-byte data path | GT10_AURORAX_8 | Xilinx 10G protocol, 8-byte data path |

There are three ways to configure the RocketIO X transceiver:

- Static properties can be set through attributes in the HDL code. Use of attributes are covered in detail in “Primitive Attributes,” page 120.
- Dynamic changes can be made to the attributes via the attribute programming bus.
- Dynamic changes can be made through the ports of the primitives.

The RocketIO X transceiver consists of the Physical Media Attachment (PMA) and Physical Coding Sublayer (PCS). The PMA contains the serializer/deserializer (SERDES), TX and RX buffers, clock generator, and clock recovery circuitry. The PCS contains the 8b/10b encoder/decoder, 64b/66b encoder/decoder/scrambler/descrambler, and the elastic buffer supporting channel bonding and clock correction. Refer again to [Table 1-1, page 110](#), showing the RocketIO X transceiver top-level block diagram and FPGA interface signals.

RocketIO X Transceiver Instantiations

For the different clocking schemes, several things must change, including the clock frequency for USRCLK and USRCLK2. The data and control ports for GT10_CUSTOM always use maximum bus widths. To implement the designs that do not take full advantage of the bus width, concatenate zeros onto inputs and the wires for outputs for Verilog designs, and set outputs to open and concatenate zeros on unused input bits for VHDL designs.

HDL Code Examples

The Architecture Wizard can be used to create instantiation templates. This wizard creates code and instantiation templates that define the attributes for a specific application.

Available Ports

Table 1-4 contains the port descriptions of all primitives. The RocketIO X transceiver primitives contain 72 ports. The differential serial data ports (RXN, RXP, TXN, and TXP) are connected directly to external pads; the remaining 68 ports are all accessible from the FPGA logic.

TABLE 1-4: Primitive Ports

| Port | I/O | Port Size | Definition |
|-----------------|-----|-----------|--|
| BREFCLKNIN | I | 1 | Differential BREFCLK negative input from the BREFCLK pad. |
| BREFCLKPIN | I | 1 | Differential BREFCLK positive input from the BREFCLK pad. |
| CHBONDDONE | O | 1 | Indicates a receiver has successfully completed channel bonding when asserted High. |
| CHBONDI[4:0] | I | 5 | The channel bonding control that is used only by “slaves” which is driven by a transceiver's CHBONDO port. |
| CHBONDO[4:0] | O | 5 | Channel bonding control that passes channel bonding and clock correction control to other transceivers. |
| ENCHANSYNC | I | 1 | Control from the fabric to the transceiver enables the transceiver to perform channel bonding. |
| ENMCOMMAALIGN | I | 1 | Selects realignment of incoming serial bitstream on minus-comma. When asserted realigns serial bitstream byte boundary to where minus-comma is detected. |
| ENPCOMMAALIGN | I | 1 | Selects realignment of incoming serial bitstream on plus-comma. When reasserted realigns serial bitstream byte boundary to where plus-comma is detected. |
| LOOPBACK[1:0] | I | 2 | Selects the three loopback test modes. These modes are internal parallel, pre-driver serial, and post-driver serial. |
| PMAINIT | I | 1 | When asserted High and then deasserted Low, reloads the PMA coefficients into the PMA from the attribute PMA_SPEED and then resets the PCS. |
| PMAREGADDR[5:0] | I | 6 | PMA attribute bus address. This input is asynchronous. |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|--------------------------|-----|---------------------------|--|
| PMAREGDATAIN[7:0] | I | 8 | PMA attribute bus data input. This input is asynchronous. |
| PMAREGRW | I | 1 | PMA attribute bus read/write control. This input is asynchronous. |
| PMAREGSTROBE | I | 1 | PMA attribute bus strobe. Note: This input is asynchronous. |
| PMARXLOCK | O | 1 | Indicates that the receive PLL has locked in the fine loop. When RX PLL is set to “Lock to Data,” this signal is always a logic 1. |
| PMARXLOCKSEL[1:0] | I | 2 | Selects determination of lock in the receive PLL. |
| POWERDOWN | I | 1 | Shuts down both the receiver and transmitter sides of the transceiver when asserted High. Note: This input is asynchronous. |
| REFCLK | I | 1 | The reference clock net that is embedded within the fabric. |
| REFCLK2 | I | 1 | An alternative to REFCLK. Can be selected by the REFCLKSEL. |
| REFCLKBSEL | I | 1 | Selects between BREFCLK and REFCLK/REFCLK2 as reference clock. Asserted selects BREFCLK. Deasserted selects REFCLK or REFCLK2, depending on REFCLKSEL. |
| REFCLKSEL | I | 1 | Selects between REFCLK or REFCLK2 as reference clock. Deasserted selects REFCLK. Asserted selects REFCLK2. |
| RXBUFSTATUS[1:0] | O | 2 | Receiver elastic buffer status. Indicates the status of the receive FIFO pointers, channel bonding skew, and clock correction events. |
| RXBLOCKSYNC 64B66BUSE | I | 1 | If asserted, the block sync is used. If deasserted, the block sync logic is bypassed. |
| RXCHARISCOMMA [7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | Indicates the reception of K28.0, K28.5, K28.7, and some out of band commas (depending on the setting of DEC_VALID_COMMA_ONLY by the 8b/10b decoder. |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|------------------------|-----|------------------------------|--|
| RXCHARISK[7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | If 8b/10b decoding is enabled, it indicates that the received data is a “K” character when asserted. Included in Byte-mapping. If 8b/10b decoding is bypassed, it remains as the first bit received (Bit “a”) of the 10-bit encoded data |
| RXCLKCORCNT[2:0] | O | 3 | Status that denotes occurrence of clock correction, channel bonding, and receive FIFO pointer status. This status is synchronized on the incoming RXDATA. |
| RXCOMMADET | O | 1 | Indicates that the symbol defined by PCOMMA_10B_VALUE (if ENPCOMMAALIGN is asserted) and/or MCOMMA_10B_VALUE (if ENMCOMMAALIGN is asserted) has been received. |
| RXCOMMADETUSE | I | 1 | If asserted High, the comma detect is used. If deasserted, the comma detect is bypassed. |
| RXDATA[63:0] | O | 8, 16, 32, 64 ⁽²⁾ | Up to eight bytes of decoded (8b/10b encoding) or encoded (8b/10b bypassed) received data at the user fabric. |
| RXDATAWIDTH[1:0] | I | 2 | (00, 01, 10, 11) Indicates width of FPGA parallel bus. |
| RXDEC64B66BUSE | I | 1 | If asserted High, the 64b/66b decoder is used. If deasserted, the 64b/66b decoder is bypassed. |
| RXDEC8B10BUSE | I | 1 | If asserted High, the 8b/10b decoder is used. If deasserted, the 8b/10b decoder is bypassed. CLK_COR_8B10B_DE = RXDEC8B10BUSE |
| RXDESCRAM 64B66BUSE | I | 1 | If asserted High, the scrambler is used. If deasserted, the scrambler is bypassed. |
| RXDISPERR[7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | If 8b/10b encoding is enabled it indicates whether a disparity error has occurred on the serial line. Included in Byte-mapping scheme. |
| RXIGNOREBTF | I | 1 | If asserted High, the block type field (BTF) is ignored in the 64b/66b decoder. Instead of reporting an error, the block is passed on as is. If deasserted, unrecognized BTFs are marked as error blocks. |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|---------------------|-----|---------------------------|--|
| RXINTDATAWIDTH[1:0] | I | 2 | (00, 01, 10, 11) Sets the internal mode of the receive PCS, either 16-, 20-, 32-, or 40-bit. |
| RXLOSSOFSYNC[1:0] | O | 2 | Bit 0 is always zero. Bit 1 indicates there is a 64b/66b Block Lock when deasserted to logic Low. |
| RXN | I | 1 | Serial differential port (FPGA external) |
| RXNOTINTABLE[7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | Status of encoded data when the data is not a valid character when asserted High. Applies to the byte-mapping scheme. |
| RXP | I | 1 | Serial differential port (FPGA external) |
| RXPOLARITY | I | 1 | Similar to TXPOLARITY, but for RXN and RXP. When deasserted, assumes regular polarity. When asserted, reverses polarity. |
| RXREALIGN | O | 1 | Signal from the PMA denoting that the byte alignment with the serial data stream changed due to a comma detection. Asserted High when alignment occurs. |
| RXRECCLK | O | 1 | Clock recovered from the data stream and divided. Divide ratio depends on PMA_SPEED setting and/or PMA attributes. |
| RXRESET | I | 1 | Synchronous RX system reset that “recenters” the receive elastic buffer. It also resets 8b/10b decoder, comma detect, channel bonding, clock correction logic, and other internal receive registers. It does not reset the receiver PLL. |
| RXRUNDISP[7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | Signals the running disparity (0 = negative, 1 = positive) in the received serial data. If 8b/10b encoding is bypassed, it remains as the second bit received (Bit “b”) of the 10-bit encoded data. |
| RXSLIDE | I | 1 | Enables the “slip” of the detection block by 1-bit. To enable a slide of 1-bit, it increments from a lower bit to a higher bit. This signal must be asserted and then deasserted synchronous to RXUSRCKLK2. RXSLIDE must be held Low for at least two clock cycles before being asserted High again. |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|--------------------|-----|-----------|--|
| RXUSRCLK | I | 1 | <p>Clock from a DCM or a BUFG that is used for reading the RX elastic buffer. It also clocks CHBONDI and CHBONDO in and out of the transceiver. Typically, the same as TXUSRCLK.</p> <p>RXUSRCLK and RXUSRCLK2 should be 180° out of phase from each other.</p> |
| RXUSRCLK2 | I | 1 | <p>Clock output from a DCM that clocks the receiver data and status between the transceiver and the FPGA fabric. Typically, the same as TXUSRCLK2.</p> <p>RXUSRCLK and RXUSRCLK2 should be 180° out of phase from each other.</p> |
| TXBUFERR | O | 1 | <p>Provides status of the transmission FIFO. If asserted High, an overflow/underflow has occurred. When this bit becomes set, it can only be reset by asserting TXRESET.</p> |
| TXBYPASS8B10B[7:0] | I | 8 | <p>If TXENC8B10BUSE = 1 and TXENC64B66BUSE = 0 (8b/10b encoder enabled and 64b/66b encoder disabled), each bit of TXBYPASS8B10B[7:0] controls the bypass of the corresponding TXDATA byte; an asserted bit bypasses encoding for the data in the corresponding byte lane.</p> <p>If TXENC8B10BUSE = 0 and TXENC64B66BUSE = 1 (8b/10b encoder disabled and 64b/66b encoder enabled), TXBYPASS8B10B[2:0] bits are used for additional 64B / 66B encoder block bypass control. TXBYPASS8B10B[7:3] bits are not relevant in this particular configuration. Bits [2:1] carry the substitute sync header (SH[1:0]) for the block bypass operation; bit [0] is asserted for each block that the user wants to bypass.</p> |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|-------------------------|-----|---------------------------------|---|
| TXCHARDISPMODE [7:0] | I | 1, 2, 4, 8 ⁽¹⁾ | If 8b/10b encoding is enabled, this bus determines what mode of disparity is to be sent. When 8b/10b is bypassed, this becomes the first bit transmitted (Bit “a”) of the 10-bit encoded TXDATA bus section for each byte specified by the byte-mapping. The bits have no meaning if TXENC8B10BUSE is deasserted. |
| TXCHARDISPVAL [7:0] | I | 1, 2, 4, 8 ⁽¹⁾ | If 8b/10b encoding is enabled, this bus determines what type of disparity is to be sent. When 8b/10b is bypassed, this becomes the second bit transmitted (Bit “b”) of the 10-bit encoded TXDATA bus section for each byte specified by the byte-mapping section. The bits have no meaning if TXENC8B10BUSE is deasserted. |
| TXCHARISK[7:0] | I | 1, 2, 4, 8 ⁽¹⁾ | If TXENC8B10BUSE = 1 (8b/10b encoder enable), then TXCHARISK[7:0] signals the K-definition of the TXDATA byte in the corresponding byte lane. (1 indicates that the byte is a K character; 0 indicates that the byte is a data character) If TXENC64B66BUSE = 1 (64b/66b encoder enable), then TXCHARISK[3:0] signals the block-formatting definitions of TXDATA (1 indicates that the byte is a control character; 0 indicates that the byte is a data character). TXCHARISK[7:4] bits are not relevant in this particular configuration. |
| TXDATA[63:0] | I | 8, 16, 32, 64 ⁽²⁾ | Transmit data from the FPGA user fabric that can be 1, 2, 4, or 8 bytes wide, depending on the primitive used. TXDATA[7:0] is always the first byte transmitted. The position of the first byte depends on selected TX data path width. |
| TXDATAWIDTH[1:0] | I | 2 | (00, 01, 10, 11) Indicates width of FPGA parallel bus. |
| TXENC64B66BUSE | I | 1 | If asserted High, the 64b/66b encoder is used. If deasserted, the 64b/66b encoder is bypassed. |
| TXENC8B10BUSE | I | 1 | If asserted High, the 8b/10b encoder is used. If deasserted, the 8b/10bencoder is bypassed. |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|----------------------|-----|---------------------------|--|
| TXGEARBOX64B66BUSE | I | 1 | If asserted High, the 64b/66b gearbox is used. If deasserted, the 64b/66b gearbox is bypassed. TXSCRAM64B66BUSE = TXGEARBOX64B66BUSE |
| TXINHIBIT | I | 1 | If asserted High, the TX differential pairs are forced to be a constant 1/0. TXN = 1, TXP = 0 |
| TXINTDATAWIDTH [1:0] | I | 2 | (00, 01, 10, 11) Indicates internal data width |
| TXKERR[7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | Indicates even boundary for bypassing in 64b/66b mode. |
| TXN | O | 1 | Transmit differential port (FPGA external) |
| TXOUTCLK | O | 1 | Synthesized Clock from RocketIO X transmitter. This clock can be scaled (e.g., for 64b/66b) relative to BREFCLK, depending upon the specific operating mode of the transmitter. |
| TXP | O | 1 | Transmit differential port (FPGA external) |
| TXPOLARITY | I | 1 | Specifies whether or not to invert the final transmitter output. Able to reverse the polarity on the TXN and TXP lines. Deasserted sets regular polarity. Asserted reverses polarity. |
| TXRESET | I | 1 | Synchronous TX system reset that “recenters” the transmit elastic buffer. It also resets 8b/10b encoder and other internal transmission registers. It does not reset the transmission PLL. |
| TXRUNDISP[7:0] | O | 1, 2, 4, 8 ⁽¹⁾ | Signals the running disparity for its corresponding byte, after that byte is encoded. Zero equals negative disparity and positive disparity for a one. This is also overloaded to be the data output bus of the PMA attribute bus. |
| TXSCRAM64B66BUSE | I | 1 | If asserted High, the 64b/66b scrambler is used. If deasserted, the 64b/66b scrambler is bypassed. TXSCRAM64B66BUSE = TXGEARBOX64B66BUSE |

TABLE 1-4: Primitive Ports (*Continued*)

| Port | I/O | Port Size | Definition |
|-----------|-----|-----------|---|
| TXUSRCLK | I | 1 | <p>Clock output from a DCM that is clocked with the REFCLK (or other reference clock). This clock is used for writing the TX buffer and is frequency-locked to the REFCLK.</p> <p>TXUSRCLK and TXUSRCLK2 should be 180° out of phase from each other.</p> |
| TXUSRCLK2 | I | 1 | <p>Clock output from a DCM that clocks transmission data and status and reconfiguration data between the transceiver and the FPGA fabric. The ratio between the TXUSRCLK and TXUSRCLK2 depends on the width of the TXDATA.</p> <p>TXUSRCLK and TXUSRCLK2 should be 180° out of phase from each other.</p> |

Notes:

1. Port size depends on which primitive is used (1, 2, 4, 8 byte).
2. Port size depends on which primitive is used (8, 16, 32, 64 byte).

Primitive Attributes

The primitives also contain attributes set by default to specific values controlling each specific primitive's protocol parameters. Included are channel-bonding settings (for primitives supporting channel bonding), and clock correction sequences. Table 1-5 shows a brief description of each attribute.

TABLE 1-5: RocketIO X Transceiver Attributes

| Attribute | Type | Description |
|---------------------|---------|---|
| ALIGN_COMMA_WORD | Integer | Integer (1, 2, 4) controls the alignment of detected commas within the transceiver's 4-byte wide data path. |
| CHAN_BOND_64B66B_SV | Boolean | TRUE/FALSE. This signal is reserved for future use and must be held to FALSE. |
| CHAN_BOND_LIMIT | Integer | <p>Integer 1-63 that defines maximum number of bytes a slave receiver can read following a channel bonding sequence and still successfully align to that sequence.</p> <p>This attribute must be set to 16.</p> |

TABLE 1-5: RocketIO X Transceiver Attributes (*Continued*)

| Attribute | Type | Description |
|--------------------|---------|---|
| CHAN_BOND_MODE | String | <p>STRING OFF, MASTER, SLAVE_1_HOP, SLAVE_2_HOPS</p> <p>OFF: No channel bonding involving this transceiver.</p> <p>MASTER: This transceiver is master for channel bonding. Its CHBONDO port directly drives CHBONDI ports on one or more SLAVE_1_HOP transceivers.</p> <p>SLAVE_1_HOP: This transceiver is a slave for channel bonding. SLAVE_1_HOP's CHBONDI is directly driven by a MASTER transceiver CHBONDO port. SLAVE_1_HOP's CHBONDO port can directly drive CHBONDI ports on one or more SLAVE_2_HOPS transceivers.</p> <p>SLAVE_2_HOPS: This transceiver is a slave for channel bonding. SLAVE_2_HOPS CHBONDI is directly driven by a SLAVE_1_HOP CHBONDO port.</p> |
| CHAN_BOND_ONE_SHOT | Boolean | <p>FALSE/TRUE that controls repeated execution of channel bonding.</p> <p>FALSE: Master transceiver initiates channel bonding whenever possible (whenever channel-bonding sequence is detected in the input) as long as input ENCHANSYNC is High and RXRESET is Low.</p> <p>TRUE: Master transceiver initiates channel bonding only the first time it is possible (channel bonding sequence is detected in input) following negated RXRESET and asserted ENCHANSYNC. After channel-bonding alignment is done, it does not occur again until RXRESET is asserted and negated, or until ENCHANSYNC is negated and reasserted.</p> <p>Slave transceivers should always have CHAN_BOND_ONE_SHOT set to FALSE.</p> |

TABLE 1-5: RocketIO X Transceiver Attributes (*Continued*)

| Attribute | Type | Description |
|-----------------------------|------------------|--|
| CHAN_BOND_SEQ_1_* [10:0] | 11-bit vector | These define the channel bonding sequence. The usage of these vectors also depends on CHAN_BOND_SEQ_LEN and CHAN_BOND_SEQ_2_USE. |
| CHAN_BOND_SEQ_1_MASK[3:0] | 4-bit vector | Each bit of the mask determines if that particular sequence is detected regardless of its value. If bit 0 is High, then CHAN_BOND_SEQ_1_1 is matched regardless of its value. |
| CHAN_BOND_SEQ_2_* [10:0] | 11-bit vector | These define the channel bonding sequence: The usage of these vectors also depends on CHAN_BOND_SEQ_LEN and CHAN_BOND_SEQ_2_USE. |
| CHAN_BOND_SEQ_2_MASK[3:0] | 4-bit vector | Each bit of the mask determines if that particular sequence is detected regardless of its value. If bit 0 is High, then CHAN_BOND_SEQ_2_1 is matched regardless of its value. |
| CHAN_BOND_SEQ_2_USE | Boolean | FALSE/TRUE that controls use of second channel bonding sequence. FALSE: Channel bonding uses only one channel bonding sequence defined by CHAN_BOND_SEQ_1_1 ... 4, or one 8-byte sequence defined by CHAN_BOND_SEQ_1_X and CHAN_BOND_SEQ_2_X in combination. TRUE: Channel bonding uses two channel bonding sequences defined by CHAN_BOND_SEQ_1_1 ... 4 and CHAN_BOND_SEQ_2_1 ... 4, as further constrained by CHAN_BOND_SEQ_LEN. |
| CHAN_BOND_SEQ_LEN | Integer | Integer (1, 2, 3, 4, 8) defines length in bytes of channel bonding sequence. This defines the length of the sequence the transceiver matches to detect opportunities for channel bonding. |
| CLK_COR_8B10B_DE | Boolean | This signal selects if clock correction occurs relative to the encoded or decoded version of the 8b/10b stream. If set to TRUE, the decoded version is used. If set to FALSE, the encoded version is used. Must be set in conjunction with RXDEC8B10USE. CLK_COR_8B10B_DE = RXDEC8B10BUSE |

TABLE 1-5: RocketIO X Transceiver Attributes (*Continued*)

| Attribute | Type | Description |
|--------------------------|---------------|---|
| CLK_COR_MAX_LAT | Integer | (0-63) Integer defines the upper bound of the receive FIFO. This attribute is recommended to be set to 48. |
| CLK_COR_MIN_LAT | Integer | (0-63) Integer defines the lower bound of the receive FIFO. This attribute is recommended to be set to 32. |
| CLK_COR_SEQ_1_*[10:0] | 11-bit vector | These define the sequence for clock correction. The attribute used depends on the CLK_COR_SEQ_LEN and CLK_COR_SEQ_2_USE. |
| CLK_COR_SEQ_1_MASK [3:0] | 4-bit vector | Each bit of the mask determines if that particular sequence is detected regardless of its value. If bit 0 is High, then CLK_COR_SEQ_1_1 is matched regardless of its value. |
| CLK_COR_SEQ_2_*[10:0] | 11-bit vector | These define the sequence for clock correction. The attribute used depends on the CLK_COR_SEQ_LEN and CLK_COR_SEQ_2_USE. |
| CLK_COR_SEQ_2_MASK [3:0] | 4-bit vector | Each bit of the mask determines if that particular sequence is detected regardless of its value. If bit 0 is High, then CLK_COR_SEQ_2_1 is matched regardless of its value. |
| CLK_COR_SEQ_2_USE | Boolean | FALSE/TRUE Control use of second clock correction sequence. FALSE: Clock correction uses only one clock correction sequence defined by CLK_COR_SEQ_1_1 ... 4, or one 8-byte sequence defined by CLK_COR_SEQ_1_X and CLK_COR_SEQ_2_X in combination. TRUE: Clock correction uses two clock correction sequences defined by CLK_COR_SEQ_1_1 ... 4 and CLK_COR_SEQ_2_1 ... 4, as further constrained by CLK_COR_SEQ_LEN. |

TABLE 1-5: RocketIO X Transceiver Attributes (*Continued*)

| Attribute | Type | Description |
|---------------------|---------------|---|
| CLK_COR_SEQ_DROP | Boolean | TRUE/FALSE. When asserted TRUE, the clock correction mode is via idle removal. When FALSE, the clock correction mode is via idle removal or insertion. This attribute must be set to FALSE. |
| CLK_COR_SEQ_LEN | Integer | Integer (1, 2, 3, 4, 8) that defines the length of the sequence the transceiver matches to detect opportunities for clock correction. It also defines the size of the correction, since the transceiver executes clock correction by repeating or skipping entire clock correction sequences. |
| CLK_CORRECT_USE | Boolean | TRUE/FALSE controls the use of clock correction logic. FALSE: Permanently disable execution of clock correction (rate matching). Clock RXUSRCLK must be frequency-locked with RXRECCLK in this case. TRUE: Enable clock correction (normal mode). |
| COMMA_10B_MASK[9:0] | 10-bit vector | These define the mask that is ANDed with the incoming serial-bit stream before comparison against PCOMMA_10B_VALUE and MCOMMA_10B_VALUE. |
| DEC_MCOMMA_DETECT | Boolean | TRUE/FALSE controls the raising of per-byte flag RXCHARISCOMMA on minus-comma. |
| DEC_PCOMMA_DETECT | Boolean | TRUE/FALSE controls the raising of per-byte flag RXCHARISCOMMA on plus-comma. |

TABLE 1-5: RocketIO X Transceiver Attributes (*Continued*)

| Attribute | Type | Description |
|------------------------|---------------|--|
| DEC_VALID_COMMA_ONLY | Boolean | TRUE/FALSE controls the raising of RXCHARISCOMMA on an invalid comma. FALSE: Raise RXCHARISCOMMA on: xxx1111100 (if DEC_PCOMMA_DETECT is TRUE) and/or on: xxx0000011 (if DEC_MCOMMA_DETECT is TRUE) or on 8b/10b translation commas regardless of the settings of the xxx bits. TRUE: Raise RXCHARISCOMMA only on valid characters that are in the 8b/10b translation. |
| MCOMMA_10B_VALUE [9:0] | 10-bit vector | These define minus-comma for the purpose of raising RXCOMMADET and realigning the serial bit stream byte boundary. This definition does not affect 8b/10b encoding or decoding. Also see COMMA_10B_MASK. |
| MCOMMA_DETECT | Boolean | TRUE/FALSE indicates whether to raise or not raise the RXCOMMADET when minus-comma is detected. |
| PCOMMA_10B_VALUE[9:0] | 10-bit vector | These define plus-comma for the purpose of raising RXCOMMADET and realigning the serial bit stream byte boundary. This definition does not affect 8b/10b encoding or decoding. Also see COMMA_10B_MASK. |
| PCOMMA_DETECT | Boolean | TRUE/FALSE indicates whether to raise or not raise the RXCOMMADET when plus-comma is detected. |
| PMA_PWR_CNTRL | Integer | This masks the startup sequence of the PMA and must always be set to all ones. |
| PMA_SPEED | String | (13_40) Selects the mode of the PMA. Refer to PMA section for the proper mode selection. |
| RX_BUFFER_USE | Boolean | TRUE/FALSE. Recommended to always be set to TRUE. Enables the use of the receive side buffer. When set to TRUE, the buffer is enabled. |

TABLE 1-5: RocketIO X Transceiver Attributes (*Continued*)

| Attribute | Type | Description |
|--------------------------|--------------|--|
| RX_LOS_INVALID_INCR[7:0] | Integer | Power of two in a range of 1 to 128 that denotes the number of valid characters required to "cancel out" appearance of one invalid character for loss of sync determination. |
| RX_LOS_THRESHOLD | Integer | Power of two in a range of 4 to 512. When divided by RX_LOS_INVALID_INCR, denotes the number of invalid characters required to cause FSM transition to "sync lost" state. |
| RX_LOSS_OF_SYNC_FSM | Boolean | Undefined. |
| SH_CNT_MAX[7:0] | 8-bit vector | 8-bit binary; controls when the 64b/66b synchronization state machine enters synchronization. (max sync header count) |
| SH_INVALID_CNT_MAX [7:0] | 8-bit vector | 8-bit binary; controls when the 64b/66b synchronization state machine leaves synchronization. (max invalid sync header count) |
| TX_BUFFER_USE | Boolean | When set to TRUE, this enables the use of the transmit buffer. |

Modifiable Attributes

As shown in Appendix F, "Modifiable Attributes" of the RocketIO X User Guide only certain attributes are modifiable for any primitive. These attributes help to define the protocol used by the primitive. Only the GT10_CUSTOM primitive allows the user to modify all of the attributes to a protocol not supported by another transceiver primitive. This allows for complete flexibility. The other primitives allow modification of the analog attributes of the serial data lines and several channel-bonding values.

Byte Mapping

Most of the 8-bit wide status and control buses correlate to a specific byte of the TXDATA or RXDATA. This scheme is shown in Table 1-6. This creates a way to tie all the signals together regardless of the data path width needed for the GT10_CUSTOM. All other primitives with specific data width paths and all byte-mapped ports are affected by this situation. For example, a 1-byte wide data

path has only 1-bit control and status bits (TXCHARISK[0]) correlating to the data bits TXDATA[7:0].

TABLE 1-6: Control/Status Bus Association to Data Bus Byte Paths

| Control/Status Bit | Data Bits |
|--------------------|-----------|
| [0] | [7:0] |
| [1] | [15:8] |
| [2] | [23:16] |
| [3] | [31:24] |
| [4] | [39:32] |
| [5] | [47:40] |
| [6] | [55:48] |
| [7] | [64:56] |

Digital Design Considerations

The Physical Coding Sublayer (PCS) portion of the RocketIO X transceiver has been significantly updated relative to the RocketIO. The RocketIO X PCS supports 8b/10b and 64b/66b encode/decode, SONET compatibility, and generic data modes. The RocketIO X transceiver operates in four basic internal modes: 16-bit, 20-bit, 32-bit, and 40-bit. When accompanied by the predefined modes of the Physical Media Attachment (PMA), the user has a large combination of protocols and data rates from which to choose. With the custom RocketIO X transceiver, the user has an almost infinite amount of possibilities from which to choose in constructing the most advanced and easily configurable communication paths in the history of communication ICs.

The RocketIO X PCS also represents a shift in the configurability of transceivers. This allows the user to change not only speeds of the PMA in real time, but also protocols within the PCS. Internal data width, external data width, and data routing can all be configured on a clock-by-clock basis. With this advancement, users can initialize a communication channel at a low speed (for example, 2.5 Gb/s using 8b/10b (20-bit internal) and then auto-negotiate after the channel is stable to a 10.3125 Gb/s speed using 64b/66b (32-bit internal).

The information in this section is provided to RocketIO X users as a reference for understanding the individual attribute and control port settings within a primitive. Users have the choice of using the supported primitives in Table A-1-3, page 111, and ignoring this chapter, or using this chapter to better understand PCS configuration and/or to modify attribute and port values to create a user transceiver configuration.

Top-Level Architecture

Transmit Architecture

The transmit architecture for the PCS is shown in Figure 1-2. For information about bypassing particular blocks, consult the block function section for that particular block.

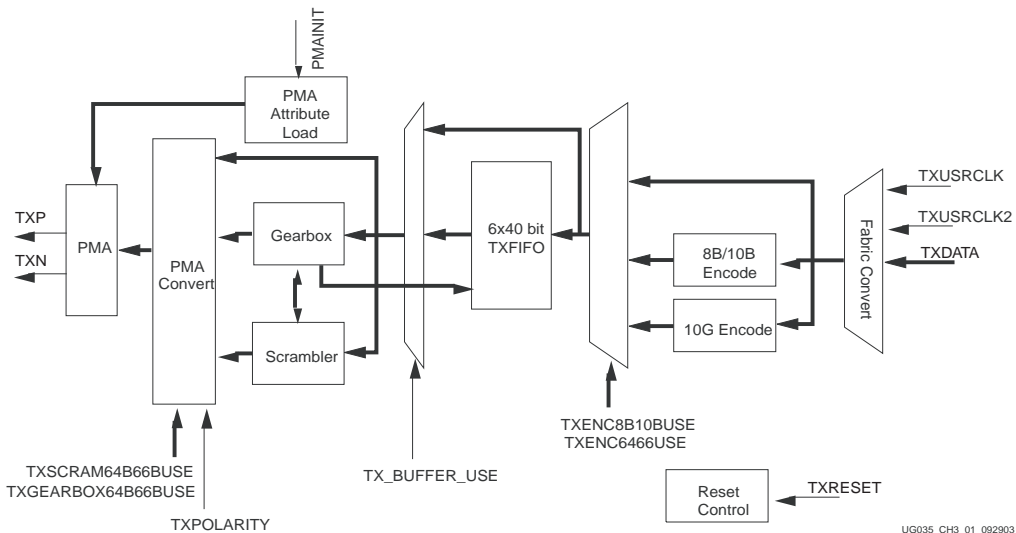


FIGURE 1-2: Transmit Architecture

Receive Architecture

The receive architecture for the PCS is shown in Figure 1-3. For information about bypassing particular blocks, consult the block function section for that particular block.

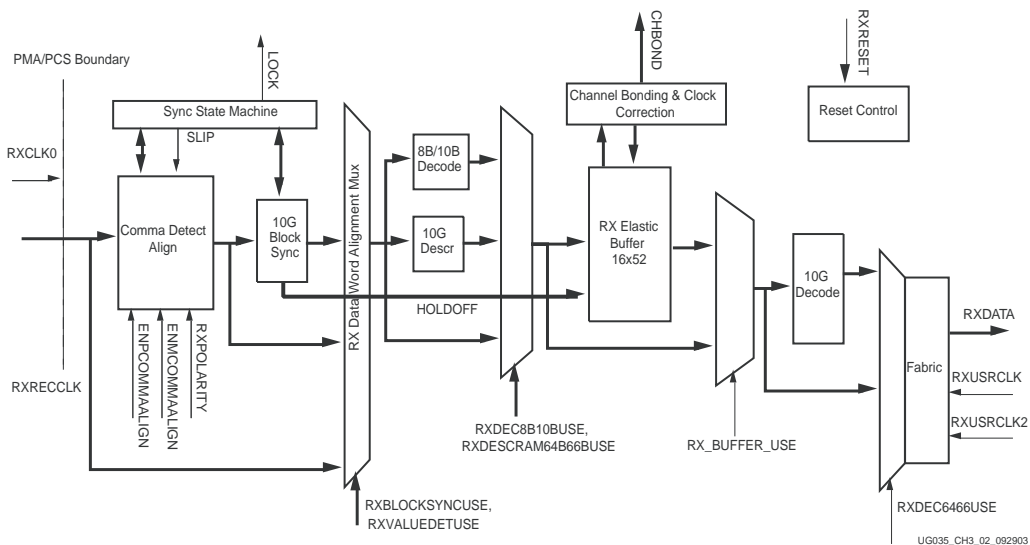


FIGURE 1-3: Receive Architecture

Operation Modes

Internally, there are four modes of operation within the PCS: 16-bit, 20-bit, 32-bit, and 40-bit.

The PCS fundamentally operates in either 2-byte mode, or 4-byte mode, with 2-byte mode corresponding to 16- and 20-bit mode, and with 4-byte mode corresponding to 32- and 40-bit mode. When in 2-byte mode, the external interface can either be one, two, or four bytes wide. When in 4-byte mode, the external interface can either be 4 or 8 bytes wide. It is not possible to have an internal 2-byte width and an 8-byte external interface. It is also not possible to have an internal 4-byte interface, along with a 1-byte external interface. See Table 1-7.

TABLE 1-7: PCS Interface Choice

| Speed | 2 Byte (internal mode) | 4 Byte (internal mode) |
|------------------|------------------------|------------------------|
| 2.488 Gb/s | Recommended | Do not use |
| 5 - 10.3125 Gb/s | Do not use | Recommended |

A general guide to use is that 2-byte mode should be used in the PCS when the serial speed is below 5 Gb/s, and the 4-byte mode should be used when the serial speed is greater than 5 Gb/s. In 2-byte mode, the PCS processes 4-byte data every other byte. This is transparent to the user, but skews between transceivers result in larger bit skews at the transmit interface as compared to Virtex-II Pro transceivers. Any one of the three encoding schemes (8b/10b and 64b/66b encode/decode, SONET,

and generic data modes) can be used in either 2- or 4-byte mode, with each block having a bypass ability.

For more information on setting the PCS mode, refer to the block functional definition of the bus interface in this guide.

Block Level Functions

Classification of Signals and Overloading

This section describes the pertinent signals at the interface of the PCS and how to prioritize them. For more information about a particular signal, refer to the I/O specification, or the particular block function of interest.

Static Signals (Control Inputs)

The following static signals are inputs that control the internal and external mode of operation in the PCS. Typically, these signals would be the first consideration after the mode of operation has been selected:

- RXDATAWIDTH[1:0]
- RXINTDATAWIDTH[1:0]
- TXDATAWIDTH[1:0]
- TXINTDATAWIDTH[1:0]

The following static signals are inputs that control the PCS interblock routing and bypass for particular blocks, which adjust the architecture of the PCS for the user's particular application:

- RXBLOCKSYNC64B66BUSE
- RXDEC64B66BUSE
- RXDEC8B10BUSE
- RXDESCRAM64B66BUSE
- RXCOMMADETUSE
- TXENC64B66BUSE
- TXENC8B10BUSE
- TXGEARBOX64B66BUSE
- TXSCRAM64B66BUSE

The following static signals are inputs that control various functions, but are usually set once at the beginning of a state machine, or after an auto-negotiation sequence. They are typically not altered on a clock-by-clock basis:

- ALIGN_COMMA_WORD
- ENMCOMMAALIGN
- ENPCOMMAALIGN
- RXPOLARITY
- TXPOLARITY
- PMAINIT
- RXIGNOREBTF
- PMARXLOCKSEL[1:0]

The following static signals are inputs that cause either major functional resets or are used in troubleshooting. These signals are mostly used at initialization, not during the functional operation of the circuit:

- LOOPBACK[1:0] (*Note: This signal can also be a dynamic signal.*)

- POWERDOWN
- RXRESET
- TXRESET
- TXINHIBIT

Dynamic Signals

The following dynamic signals indicate data received on the receive bus, along with status signals that indicate specific information about RXDATA. The set values of these signals define the application setup by the user and are the most important after the static signals are allocated:

- MCOMMA_10B_VALUE
- PCOMMA_10B_VALUE
- COMMA_10B_MASK
- RXCHARISCOMMA[7:0]
- RXCHARISK[7:0]
- RXDISPERR[7:0]
- RXNOTINTABLE[7:0]
- RXDATA[63:0]

The following dynamic signals indicate data to be transmitted on the transmit bus, along with status signals that indicate specific information about how TXDATA is to be handled while passing through the PCS. The set values of these signals define the application setup by the user and are the most important after the static signals are allocated:

- TXBYPASS8B10B[7:0]
- TXCHARDISPMODE[7:0]
- TXCHARDISPVAL[7:0]
- TXCHARISK[7:0]
- TXDATA[63:0]

The following dynamic signals indicate various status information about the current state or prior state of the PCS:

- CHBONDDONE, RXBUFSTATUS[1:0], RXCLKCORCNT[2:0]
- CHBONDO[4:0]
- PMARXLOCK
- RXLOSSOFFSYNC[1:0]
- RXREALIGN
- RXCOMMADET
- TXBUFERR
- TXKERR[7:0]
- TXRUNDISP[7:0]
- RXRUNDISP[7:0]

The following dynamic signals control internal states of the PCS:

- RXSLIDE
- CHBONDI[4:0]

The following dynamic signals affect the control registers of the PMA:

- PMAREGADDR[5:0]
- PMAREGDATAIN[7:0]

- PMAREGRW
- PMAREGSTROBE

Bus Interface

Selecting the External Configuration (Fabric Interface)

By using the signals TXDATAWIDTH[1:0] and RXDATAWIDTH[1:0], the fabric interface can be determined.

TABLE 1-8: Selecting the External Configuration

| RXDATAWIDTH/TXDATAWIDTH | Data Width | Internal Bus Requirements |
|--------------------------------|--------------------|----------------------------------|
| 2'b00 | 8/10-bit (1 byte) | 16-, 20-bit mode |
| 2'b01 | 16/20-bit (2 byte) | 16-, 20-bit mode |
| 2'b10 | 32/40-bit (4 byte) | 16-, 20-, 32-, 40-bit mode |
| 2'b11 | 64/80-bit (8 byte) | 32-, 40-bit mode |

Selecting the Internal Configuration

TABLE 1-9: Selecting the Internal Configuration

| RXINTDATAWIDTH/TXINTDATAWIDTH | Internal Data Width |
|--------------------------------------|----------------------------|
| 2'b00 | 16-bit |
| 2'b01 | 20-bit |
| 2'b10 | 32-bit |
| 2'b11 | 40-bit |

Clock Ratio

USRCLK2 clocks the data buffers. The ability to send parallel data to the transceiver at four different widths requires the user to change the frequency of USRCLK2. This creates a frequency ratio between USRCLK and USRCLK2. The falling edges of the clocks must align. See Table 1-10.

TABLE 1-10: Data Width Clock Ratios

| Fabric Data Width | Frequency Ratio of USRCLK/USRCLK2 | |
|--------------------------|--|-----------------------------------|
| | 2-Byte Internal Data Width | 4-Byte Internal Data Width |
| 1 byte | 1:2 ⁽¹⁾ | N/A |
| 2 byte | 1:1 | N/A |

TABLE 1-10: Data Width Clock Ratios (*Continued*)

| Fabric Data Width | Frequency Ratio of USRCLK\USRCLK2 | |
|-------------------|-----------------------------------|----------------------------|
| | 2-Byte Internal Data Width | 4-Byte Internal Data Width |
| 4 byte | 2:1 ⁽¹⁾ | 1:1 |
| 8 byte | N/A | 2:1 ⁽¹⁾ |

Notes:

1. Each edge of slower clock must align with falling edge of faster clock.

8b/10b

In the RocketIO transceiver, the most significant byte was sent first; in the RocketIO X transceiver the least significant byte is sent first.

The following sections categorize the ports and attributes of the transceiver according to specific functionality including 8b/10b encoding/decoding, 64b/66b encoding/decoding, SERDES alignment, clock correction (clock recovery), channel bonding, fabric interface, and other signals.

The 8b/10b encoding translates an 8-bit parallel data byte to be transmitted into a 10-bit serial data stream. This conversion and data alignment are shown in Figure 1-4. The serial port transmits the least significant bit of the 10-bit data, “a” first and proceeds to “j”. This allows data to be read and matched to the form shown in Appendix B, “8b/10b Valid Characters.”.

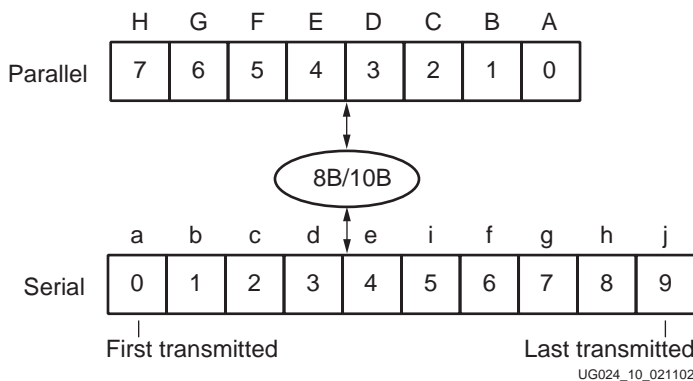


FIGURE 1-4: 8b/10b Parallel-to-Serial Conversion

The serial data bit sequence is dependent on the width of the parallel data. The least significant byte is always sent first regardless of the whether 1-byte, 2-byte, 4-byte, or 8-byte paths are used. The most significant byte is always last. Figure 1-5 shows a case when the serial data corresponds to each byte of the parallel data. TXDATA[7:0] is serialized and sent out first followed by TXDATA[15:8],

TXDATA[23:16], and finally TXDATA[31:24]. The 2-byte path transmits TXDATA[7:0] and then TXDATA[15:8].

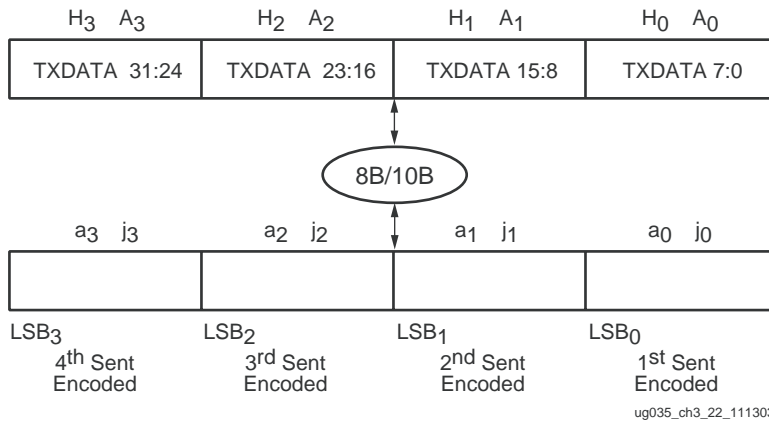


FIGURE 1-5: 4-Byte Serial Structure

Encoder

A bypassable 8b/10b encoder is included in the transmitter. The encoder uses the same 256 data characters and 12 control characters (shown in Appendix B, “8b/10b Valid Characters”) that are used for Gigabit Ethernet, XAUI, Fibre Channel, and InfiniBand.

The encoder accepts eight bits of data along with a K-character signal for a total of nine bits per character applied. If the K-character signal is High, the data is encoded into one of the 12 possible K-characters available in the 8b/10b code. If the K-character input is Low, the eight bits are encoded as standard data.

There are two ports that enable the 8b/10b encoding in the transceiver. The TXBYPASS8B10B is a byte-mapped port that is 1, 2, 4, or 8 bits depending on the data width of the transceiver primitive being used. These bits correlate to each byte of the data path. To enable the 8b/10b encoding of the transmitter, these bits should be set to a logic 0. In this mode, the transmit data that is input to the TXDATA port is non-encoded data of either 8, 16, 32, or 64 bits wide. However, if other encoding schemes are preferred, the encoder capabilities are bypassed by setting all bits to a logic 1. The extra bits are fed through the TXCHARDISPMODE and TXCHARDISPVAL buses.

TXCHARDISPVAL and TXCHARDISPMODE

TXCHARDISPVAL and TXCHARDISPMODE are dual-purpose ports for the transmitter depending whether 8b/10b encoding is done. Table 1-12 shows this dual functionality. When encoding is

enabled, these ports function as byte-mapped control ports controlling the running disparity of the transmitted serial data (Table 1-11).

TABLE 1-11: Running Disparity Control

| {txchardispmode, txchardispval} | Function |
|--|---|
| 00 | Maintain running disparity normally |
| 01 | Invert normally generated running disparity before encoding this byte |
| 10 | Set negative running disparity before encoding this byte |
| 11 | Set positive running disparity before encoding this byte |

In the encoding configuration, the disparity of the serial transmission can be controlled with the TXCHARDISPVAL and TXCHARDISPMODE ports. When TXCHARDISPMODE is set to a logic 1, the running disparity is set before encoding the specific byte. TXCHARDISPVAL determines if the disparity is negative (set to a logic 0) or positive (set to a logic 1).

When TXCHARDISPMODE is set to a logic 0, the running disparity is maintained if TXCHARDISPVAL is also set to a logic 0. However, the disparity is inverted before encoding the byte when the TXCHARDISPVAL is set to a logic 1.

Most applications use the mode where both TXCHARDISPMODE and TXCHARDISPVAL are set to logic 0. Some applications can use other settings if special running disparity configurations are required, such as in the “Vitesse Disparity Example,” page 139.

In the bypassed configuration, TXCHARDISPMODE[0] becomes bit 9 of the 10 bits of encoded data (TXCHARDISPMODE[1:7] are bits 19, 29, 39, 49, 59, 69, and 79 in the 20-bit and 40-bit and 80-bit wide buses). TXCHARDISPVAL becomes bits 8, 18, 28, 38, 48, 58, 68, and 78 of the transmit data bus while the TXDATA bus completes the bus. See Table 1-12.

TABLE 1-12: 8b/10b Bypassed Signal Significance

| Signal | Function | |
|------------------------------------|-----------------|---|
| TXBYPASS8B10B⁽¹⁾ | 0 | 8b/10b encoding is enabled (not bypassed) |
| | 1 | 8b/10b encoding bypassed (disabled) |

TABLE 1-12: 8b/10b Bypassed Signal Significance (*Continued*)

| Signal | Function | | |
|----------------------------------|---|--|---|
| TXCHARDISPMODE, TXCHARDISPVAL | | Function, 8b/10b Enabled | Function, 8b/10b Bypassed |
| | 00 | Maintain running disparity normally | Part of 10-bit encoded byte (see Figure 1-6): TXCHARDISPMODE[0], (or: [1] / [2] / [3] / [4] / [5] / [6] / [7]) TXCHARDISPVAL[0], (or: [1] / [2] / [3] / [4] / [5] / [6] / [7]) TXDATA[7:0] (or: [15:8] / [23:16] / [31:24] / [39:32] / [47:40] / [55:48] / [63:56]) |
| | 01 | Invert the normally generated running disparity before encoding this byte. | |
| | 10 | Set negative running disparity before encoding this byte. | |
| 11 | Set positive running disparity before encoding this byte. | | |
| TXCHARISK | Received byte is a K-character | Unused | |

Notes:

1. If 8b/10b is bypassed, this port can be defined if 64b/66b encoding is used.

During transmit, while 8b/10b encoding is enabled, the disparity of the serial transmission can be controlled with the TXCHARDISPVAL and TXCHARDISPMODE ports. When 8b/10b encoding is bypassed, these bits become Bits “b” and “a,” respectively, of the 10-bit encoded data that the transceiver must transmit to the receiving terminal. Figure 1-6 illustrates the TX data map during 8b/10b bypass.

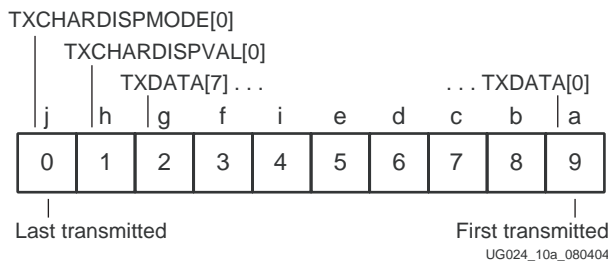


FIGURE 1-6: 10-Bit TX Data Map with 8b/10b Bypassed

TXCHARISK

TXCHARISK is a byte-mapped control port that is only used when the 8b/10b encoder is implemented. This port indicates whether the byte of TXDATA is to be encoded as a control (K) character when asserted and data character when de-asserted. When 8b/10b encoding is bypassed this port is undefined.

TXRUNDISP

TXRUNDISP is a status port that is byte-mapped to the TXDATA. This port indicates the running disparity after this byte of TXDATA is encoded. When asserted, the disparity is positive. When de-asserted, the disparity is negative.

Decoder

An optional 8b/10b decoder is included in the receiver. A programmable option allows the decoder to be bypassed. When the 8b/10b decoder is bypassed, the 10-bit character order is shown in Figure 1-7 for a graphical representation of the received 10-bit character.

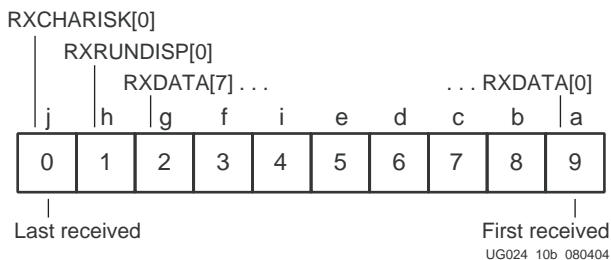


FIGURE 1-7: 10-Bit RX Data Map with 8b/10b Bypassed

The decoder uses the same table (see Appendix B, “8b/10b Valid Characters”) that is used for Gigabit Ethernet, Fibre Channel, and InfiniBand. In addition to decoding all data and K-characters, the decoder has several extra features. The decoder separately detects both “disparity errors” and “out-of-band” errors. A *disparity error* occurs when a 10-bit character is received that exists within the 8b/10b table, but has an incorrect disparity. An *out-of-band error* occurs when a 10-bit character is received that does not exist within the 8b/10b table. It is possible to obtain an out-of-band error without having a disparity error, or more commonly, a disparity error is possible without an out-of-band error. The proper disparity is always computed for both legal and illegal characters. The current running disparity is available at the RXRUNDISP signal.

The 8b/10b decoder performs a unique operation if out-of-band data is detected. If out-of-band data is detected, the decoder signals the error and passes the illegal 10-bits through and places them on the outputs. This can be used for debugging purposes if desired.

The decoder also signals reception of one of the 12 valid K-characters. In addition, a programmable comma detect is included. The comma detect signal registers a comma on the receipt of any comma+, comma–, or both. Since the comma is defined as a 7-bit character, this includes several out-of-band characters. Another option allows the decoder to detect only the three defined commas (K28.1, K28.5, and K28.7) as comma+, comma–, or both. In total, there are six possible options, three for valid commas and three for “any comma.”

Note that all bytes (1, 2, 4, or 8) at the RX FPGA interface each have their own individual 8b/10b indicators (K-character, disparity error, out-of-band error, current running disparity, and comma detect).

During receive, while 8b/10b decoding is enabled, the running disparity of the serial transmission can be read by the transceiver from the RXRUNDISP port, while the RXCHARISK port indicates presence of a K-character. When 8b/10b decoding is bypassed, these bits remain as Bits “b” and “a,” respectively, of the 10-bit encoded data that the transceiver passes on to the user logic. Table 1-13 illustrates the RX data map during 8b/10b bypass.

TABLE 1-13: 8b/10b Bypassed Signal Significance

| Signal | | Function | |
|-----------|---|--|---|
| RXCHARISK | | Received byte is a K-character | Part of 10-bit encoded byte (see Figure 1-7): |
| RXRUNDISP | 0 | Indicates running disparity is NEGATIVE | RXCHARISK[0], (or: {1} / {2} / {3} / {4} / {5} / {6} / {7}) |
| | 1 | Indicates running disparity is POSITIVE | RXRUNDISP[0], (or: {1} / {2} / {3} / {4} / {5} / {6} / {7}) RXDATA[7:0] (or: {15:8} / {23:16} / {31:24} / {39:32} / {47:40} / {55:48} / {63:56}) |
| RXDISPERR | | Disparity error occurred on current byte | Unused |

RXCHARISK and RXRUNDISP

RXCHARISK and RXRUNDISP are dual-purpose ports for the receiver depending whether 8b/10b decoding is enabled. Figure 1-10 shows this dual functionality. When decoding is enabled, these ports function as byte-mapped status ports of the received data.

In the encoding configuration, when RXCHARISK is asserted that byte of the received data is a control (K) character. Otherwise, the received byte of data is a data character. (See Appendix B, “8b/10b Valid Characters”). The RXRUNDISP port indicates the disparity of the received byte is either negative or positive. RXRUNDISP is asserted to indicate positive disparity. This is used in cases like the “Vitesse Disparity Example,” page 139.

In the bypassed configuration, RXCHARISK and RXRUNDISP are additional data bits for the 10-, 20-, 40-, or 80-bit buses. This is similar to the transmit side. RXCHARISK[0:7] relates to bits 9, 19, 29, 39, 49, 59, 69, and 79 while RXRUNDISP pertains to bits 8, 18, 28, 38, 48, 58, 68, and 78 of the data bus. See Figure 1-10.

RXDISPERR

RXDISPERR is a status port for the receiver that is byte-mapped to the RXDATA. When a bit is asserted, a disparity error occurred on the received data. This usually indicated that the data is corrupt

by bit errors, transmission of an invalid control character, or for cases when normal disparity is not required such as in the “Vitesse Disparity Example,” page 139.

RXNOTINTABLE

RXNOTINTABLE is asserted whenever the received data is not in the 8b/10b tables. The data received on bytes marked by RXNOTINTABLE are invalid. This port is also byte-mapped to RXDATA and is only used when the 8b/10b decoder is enabled.

Vitesse Disparity Example

To support other protocols, the transceiver can affect the disparity mode of the serial data transmitted. For example, Vitesse channel-to-channel alignment protocol sends out:

K28.5+ K28.5+ K28.5- K28.5-

or

K28.5- K28.5- K28.5+ K28.5+

Instead of:

K28.5+ K28.5- K28.5+ K28.5-

or

K28.5- K28.5+ K28.5- K28.5+

The logic must assert TXCHARDISPVAL to cause the serial data to send out two negative running disparity characters.

Transmitting Vitesse Channel Bonding Sequence

```

TXBYPASS8B10B
| TXCHARISK
| | TXCHARDISPMODE
| | | TXCHARDISPVAL
| | | | TXDATA
| | | |
0 1 0 0 10111100    K28.5+ (or K28.5-)
0 1 0 1 10111100    K28.5+ (or K28.5-)
0 1 0 0 10111100    K28.5- (or K28.5+)
0 1 0 1 10111100    K28.5- (or K28.5+)
    
```

The RocketIO X core receives this data but must have the CHAN_BOND_SEQ set with the disp_err bit set High for the cases when TXCHARDISPVAL is set High during data transmission.

Receiving Vitesse Channel Bonding Sequence

On the RX side, the definition of the channel bonding sequence uses the disp_err bit to specify the flipped disparity.

```

          10-bit literal value
          | disp_err
          | | char_is_k
          | | | 8-bit_byte_value
          | | | |
CHAN_BOND_SEQ_1_1 = 0 0 1 10111100    matches K28.5+ (or
K28.5-)
CHAN_BOND_SEQ_1_2 = 0 1 1 10111100    matches K28.5+ (or
K28.5-)
    
```

```

        CHAN_BOND_SEQ_1_3 = 0 0 1 10111100    matches K28.5- (or
K28.5+)
        CHAN_BOND_SEQ_1_4 = 0 1 1 10111100    matches K28.5- (or
K28.5+)
        CHAN_BOND_SEQ_LEN = 4
        CHAN_BOND_SEQ_2_USE = FALSE

```

Comma Detection

Summary

Comma detection has been expanded beyond 10-bit symbol detection and alignment to include 8-bit symbol detection and alignment for 16-, 20-, 32-, and 40-bit paths. The ability to detect symbols, and then either align to 1-word, 2-word, or 4-word boundaries is included. The RXSLIDE input allows the user to “slide” or “slip” the alignment by one bit in each 16-, 20-, 32-, and 40-bit mode at any time for SONET applications.

The following signals/attributes affect the function of the comma detection block:

- RXCOMMADETUSE
- ENMCOMMAALIGN
- ENPCOMMAALIGN
- ALIGN_COMMA_WORD[1:0]
- MCOMMA_10B_VALUE[9:0]
- DEC_MCOMMA_DETECT
- PCOMMA_10B_VALUE[9:0]
- DEC_PCOMMA_DETECT
- COMMA_10B_MASK[9:0]
- RXSLIDE
- RXINTDATAWIDTH[1:0]

Bypass

By de-asserting RXCOMMADETUSE Low, symbol detection is not enabled. If RXCOMMADETUSE is asserted High, symbol detection takes place.

Symbol Detection

By using the signals MCOMMA_10B_VALUE, DEC_MCOMMA_DETECT, PCOMMA_10B_VALUE, DEC_PCOMMA_DETECT, and COMMA_10B_MASK any 8-bit or 10-bit symbol detection can take place for two different symbol values.

To detect a 10-bit symbol COMMA_10B_MASK[9:0] should initially be set to 10'b11111_11111. Any bit can be changed to further affect the masking capability.

To detect an 8-bit symbol, the COMMA_10B_MASK[9:0] should be set to 10'b00_1111_1111. The first two bits must be set to zero. Any of the last 8 bits can be altered to change the mask further.

The MCOMMA_10B_VALUE[9:0] and PCOMMA_10B_VALUE[9:0] fields indicate the comma symbol definitions to be used by the comparison logic, i.e., the templates against which incoming data is compared in the search for commas to establish alignment.

The DEC_MCOMMA_DETECT and DEC_PCOMMA_DETECT indicate which symbol should be compared to the incoming data for alignment. See Table 1-14.

TABLE 1-14: Symbol Detection

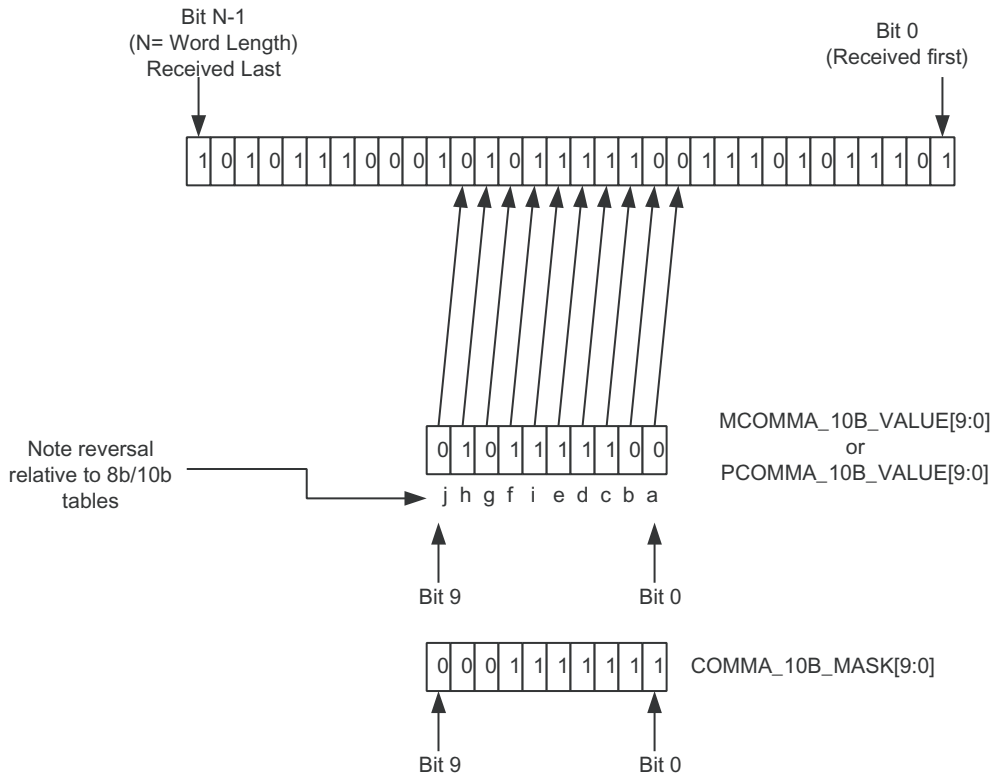
| MCOMMA_DETECT | PCOMMA_DETECT | Function |
|----------------------|----------------------|--|
| 0 | 0 | No symbol detection takes place. |
| 0 | 1 | RXCOMMADET is asserted if the incoming data is compared and aligned to the symbol defined by PCOMMA_10B_VALUE. |
| 1 | 0 | RXCOMMADET is asserted if the incoming data is compared and aligned to the symbol defined by MCOMMA_10B_VALUE. |
| 1 | 1 | RXCOMMADET is asserted if the incoming data is compared and aligned to the symbol defined by PCOMMA_10B_VALUE or MCOMMA_10B_VALUE. |

Setting MCOMMA_10B_VALUE, PCOMMA_10B_VALUE, and COMMA_10B_MASK (Special Note)

The attributes, MCOMMA_10B_VALUE, PCOMMA_10B_VALUE, and COMMA_10B_MASK are used by the MGT to indicate to the comma detection block the values to which the block should be aligned. Once set to a value, the comma detection block searches the data stream for these values and aligns the pipeline to the position where the value was detected in the data stream. Virtex-II Pro X users need to note that these values are reversed relative to Virtex-II Pro devices. The reason for this is that while Virtex-II Pro devices support mainly 8b/10b applications, Virtex-II Pro X devices can support many applications and use a more general approach.

Figure 1-8 shows a Virtex-II Pro X 8b/10b comma detection example relative to the data stream received at the PCS/PMA interface on the receive side. Note that with Virtex-II Pro devices, the M/PCOMMA_10B_VALUE[9:0] is set to 10'b0011111010, whereas in Virtex-II Pro X devices the value is set to 10'b0101111100. This also follows for the COMMA_10B_MASK, which in Virtex-II Pro devices is set to 10'b1111111000, whereas in Virtex-II Pro X devices, it is set to 10'b0001111111.

With this change, the block can be considered more of a value detection block, rather than a comma detection block. To detect values listed in the 8b/10b tables, simply reverse the values in the tables. To detect SONET type values, the exact value can be used without reversal.



UG035_CH2_12_110703

FIGURE 1-8: 8b/10b Comma Detection Example

Alignment

After the positive symbol or the negative symbol is detected, the data is aligned to that symbol. By using the signals ENMCOMMAALIGN, ENPCOMMAALIGN, ALIGN_COMMA_WORD, and RXSLIDE, alignment can be completely controlled for all data pipeline configurations. See Table 1-15.

TABLE 1-15: Data Alignment

| ENMCOMMAALIGN | ENPCOMMAALIGN | Function ⁽¹⁾ |
|---------------|---------------|--|
| 0 | 0 | No alignment takes place. |
| 0 | 1 | If a positive symbol is detected, alignment takes place at that symbol location. |

TABLE 1-15: Data Alignment (*Continued*)

| ENMCOMMAALIGN | ENPCOMMAALIGN | Function ⁽¹⁾ |
|---------------|---------------|--|
| 1 | 0 | If a negative symbol is detected, alignment takes place at that symbol location. |
| 1 | 1 | If a negative or positive symbol is detected, alignment takes place at that symbol location. |

Notes:

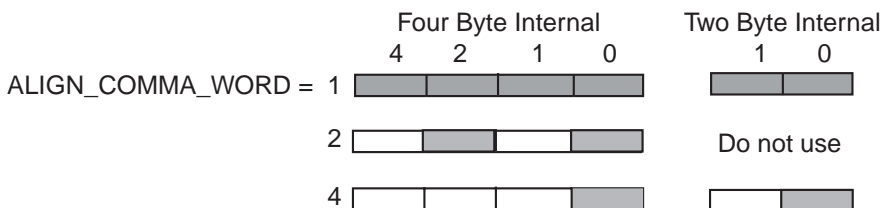
1. The symbol mentioned is defined by P/MCOMMA_10B_VALUE.

ALIGN_COMMA_WORD

The attribute ALIGN_COMMA_WORD controls when realignment takes place when the difference between symbols is on a byte-by-byte basis. If the current position of the symbol detected is some fraction of a byte different than the previous symbol position, alignment takes place regardless of the setting of ALIGN_COMMA_WORD.

- There are three options for ALIGN_COMMA_WORD: 1 byte, 2 byte, and 4 byte. When ALIGN_COMMA_WORD is set to a 1, the detection circuit allows detection symbols in contiguous bytes. When ALIGN_COMMA_WORD is set to a 2, the detection circuit allows detection symbols every other byte. When ALIGN_COMMA_WORD is set to a 4, the detection circuit allows detection symbols every fourth byte (Figure 1-9).

| | 16/20 | 32/40 |
|---|------------------|------------------|
| 1 | byte alignment | byte alignment |
| 2 | N/A | 2-byte alignment |
| 4 | 2-byte alignment | 4-byte alignment |



Note: Shaded blocks indicate where the comma can align to.

ug035_ch2_13_051904

FIGURE 1-9: ALIGN_COMMA_WORD Diagram

RXSLIDE

RXSLIDE can be used to “slide” the aligned data by one bit. The RXSLIDE function when asserted High, increments the alignment by one bit, until it reaches the most significant bit, equal to the maximum word length – 1. When RXSLIDE is asserted High, it must be asserted Low for two clock periods before it can be asserted High again. This functionality can be used for applications such as SONET.

64b/66b

Encoder

Bypassing

There are two types of bypassing regarding the 64b/66b encoder. The encoder block can either be entirely bypassed, or the 64b/66b encoder can be used and can be bypassed on a clock-by-clock basis.

If TXENC64B66BUSE is deasserted Low, the entire 64b/66b encoder is not used. If encoding is done in the fabric, the sync header [0:1] must be placed at TXCHARDISPV[0] and TXCHARDISPMODE[0] with the 32 TXDATA bits.

If TXENC64B66BUSE is asserted High, the TXBYPASS8B10B bit 0 signal bypasses the 64b/66b encoder on a clock basis, which means that two clock cycles are needed to do a full bypass of a block. The Sync Header is taken from the TXCHARDISPMODE[0:1]. To bypass on a block basis, the even boundary needs to be indicated at the fabric interface, which is contained in TXKERR bit 0. The TXCHARISK signal performs the function of TXC.

TABLE 1-16: 64b/66b Bypassing

| Signal | Function | |
|---------------------|--|--|
| TXENC64B66BUSE | 0 entire 64b/66b encoder bypassed | |
| | 1 bypass on a clock-to-clock basis | |
| TXBYPASS8B10B[0] | Function 64b/66b clock-to-clock bypass | Function 64b/66b entirely bypassed |
| | 0 indicates no bypass | defined by Table 1-18 |
| | 1 indicates bypass this block | |
| TXCHARDISPMODE[0:1] | Sync Header shown in Figure 1-11 (same as SH[0:1]) | |
| TXKERR[3] | indicates even boundary for bypassing on block basis | |
| TXCHARISK[3:0] | performs function of TXC | indicates character is a (K) control character |

The transmit 64b/66b encoder borrows four bits of the TXCHARISK bus (bits [3:0]) to convey the control signaling to the 64b/66b encoder. The four TXC bits track with the four bytes of TXDATA_IN (TXC[0] with TXDATA_IN[7:0], and so on) to signal data block formatting. The transmit fabric interface logic (which first monitors transmit data as it travels from the fabric interface to the PMA) drives the encoder with the four TXC bits as follows:

TABLE 1-17: Transmit 64b/66b Encoder Control Mapping

| TXC[3:0] (TXCHARISK[3:0]) | Block Formatting |
|----------------------------------|-------------------------------|
| 1111 | Idles OR terminate-with-idles |
| 0001 | Start-of-frame OR ordered-set |
| 1110 | Terminate in second position |
| 1100 | Terminate in third position |
| 1000 | Terminate in fourth position |
| 0000 | Data OR error (no k-chars) |

Each “one” in the TXC span represents a control-character-match -- recognition that the associated byte is a special control character of some type (idle, start, terminate, or ordered-set).

Normal Operation

The 64b/66b encoder implements the Encoding Block Format function shown in Figure 1-11.

| Input Data | S y n c | Block Payload | | | | | | | | | | |
|---|------------------|------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--|
| Bit Position | 01 | 65 | | | | | | | | | | |
| Data Block Format | 01 | D ₀ | D ₁ | D ₂ | D ₃ | D ₄ | D ₅ | D ₆ | D ₇ | | | |
| Control Block Formats | | Block Type Field | | | | | | | | | | |
| C ₀ C ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇ | 10 | 0x1e | C ₀ | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | | |
| C ₀ C ₁ C ₂ C ₃ O ₄ D ₅ D ₆ D ₇ | 10 | 0x2d | C ₀ | C ₁ | C ₂ | C ₃ | O ₄ | D ₅ | D ₆ | D ₇ | | |
| C ₀ C ₁ C ₂ C ₃ S ₄ D ₅ D ₆ D ₇ | 10 | 0x33 | C ₀ | C ₁ | C ₂ | C ₃ | | | D ₅ | D ₆ | D ₇ | |
| O ₀ D ₁ D ₂ D ₃ S ₄ D ₅ D ₆ D ₇ | 10 | 0x66 | D ₁ | D ₂ | D ₃ | O ₀ | | | D ₅ | D ₆ | D ₇ | |
| O ₀ D ₁ D ₂ D ₃ O ₄ D ₅ D ₆ D ₇ | 10 | 0x55 | D ₁ | D ₂ | D ₃ | O ₀ | O ₄ | D ₅ | D ₆ | D ₇ | | |
| S ₀ D ₁ D ₂ D ₃ D ₄ D ₅ D ₆ D ₇ | 10 | 0x78 | D ₁ | D ₂ | D ₃ | D ₄ | | D ₅ | D ₆ | D ₇ | | |
| O ₀ D ₁ D ₂ D ₃ C ₄ C ₅ C ₆ C ₇ | 10 | 0x4b | D ₁ | D ₂ | D ₃ | O ₀ | C ₄ | C ₅ | C ₆ | C ₇ | | |
| T ₀ C ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇ | 10 | 0x87 | | | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | |
| D ₀ T ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇ | 10 | 0x99 | D ₀ | | | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | |
| D ₀ D ₁ T ₂ C ₃ C ₄ C ₅ C ₆ C ₇ | 10 | 0xaa | D ₀ | D ₁ | | | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | |
| D ₀ D ₁ D ₂ T ₃ C ₄ C ₅ C ₆ C ₇ | 10 | 0xb4 | D ₀ | D ₁ | D ₂ | | | C ₄ | C ₅ | C ₆ | C ₇ | |
| D ₀ D ₁ D ₂ D ₃ T ₄ C ₅ C ₆ C ₇ | 10 | 0xcc | D ₀ | D ₁ | D ₂ | D ₃ | | | C ₅ | C ₆ | C ₇ | |
| D ₀ D ₁ D ₂ D ₃ D ₄ T ₅ C ₆ C ₇ | 10 | 0xd2 | D ₀ | D ₁ | D ₂ | D ₃ | D ₄ | | | C ₆ | C ₇ | |
| D ₀ D ₁ D ₂ D ₃ D ₄ D ₅ T ₆ C ₇ | 10 | 0xe1 | D ₀ | D ₁ | D ₂ | D ₃ | D ₄ | D ₅ | | | C ₇ | |
| D ₀ D ₁ D ₂ D ₃ D ₄ D ₅ D ₆ T ₇ | 10 | 0xff | D ₀ | D ₁ | D ₂ | D ₃ | D ₄ | D ₅ | D ₆ | | | |

UG035_ch3_23_091103

FIGURE 1-10: Block Format Function

The control codes are specified as follows in Table 1-18:

TABLE 1-18: Control Codes

| Control Character | Notation | XGMII Control Code | 10GBASE-R Control Code | 10GBASE-R 0 Code | 8b/10b Code |
|----------------------|----------|--------------------|---|------------------|-------------------------|
| idle | /I/ | 0x07 | 0x00 | | K28.0 or K28.3 or K28.5 |
| start | /S/ | 0xfb | Encoded by block type field | | K27.7 |
| terminate | /T/ | 0xfd | Encoded by block type field | | K29.7 |
| error | /E/ | 0xfe | 0x1e | | K30.7 |
| Sequence ordered_set | /Q/ | 0x9c | Encoded by block type field plus O mode | 0x0 | K28.4 |
| reserved0 | /R/ | 0x1c | 0x2d | | K28.0 |
| reserved1 | | 0x3c | 0x33 | | K28.1 |
| reserved2 | /N/ | 0x7c | 0x4b | | K28.3 |
| reserved3 | /K/ | 0xbc | 0x55 | | K28.5 |
| reserved4 | | 0xdc | 0x66 | | K28.6 |
| reserved5 | | 0xf7 | 0x78 | | K23.7 |
| Signal ordered_set | /Fsig/ | 0x5c | Encoded by block type field plus O mode | 0xF | K28.2 |

Scrambler

Bypassing

If the signal TXSCRAM64B66BUSE is deasserted Low, the scrambler is not used. Note that the scrambler operates on the read side of the transmit FIFO.

Normal Operation

If the signal TXSCRAM64B66BUSE is asserted High, the scrambler is enabled for use. The scrambler uses the polynomial:

$$G(x) = 1 + x^{39} + x^{58}$$

to scramble 64b/66b payload data. The scrambler works in conjunction with the gearbox to scramble and format data correctly.

| | |
|------------------|----------------------|
| TXSCRAM64B66BUSE | 0 scrambler not used |
| | 1 scrambler enabled |

When using the 64b/66b scrambler, the Gearbox must also be enabled
(Always set to TXSCRAM64BB66USE = TXGEARBOX64B66BUSE)

Gearbox

Bypassing

If the signal TXGEARBOX64B66BUSE is deasserted Low, the gearbox is not used. The gearbox should always be enabled when using the 64b/66b protocol.

Normal Operation

If the signal TXGEARBOX64B66BUSE is asserted High, the gear box is enabled. The gearbox frames 64b/66b data for the PMA.

| | |
|--------------------|---|
| TXGEARBOX64B66BUSE | 0 |
| | 1 always set to '1' when scrambler and descrambler are enabled. |

Decoder

Bypassing

If RXDEC64B66BUSE is deasserted Low, the entire 64b/66b decoder is not used.

Normal Operation

If RXDEC64B66BUSE is asserted High, the 64b/66b decoder decodes according to the 64b/66b block format table shown in Figure 1-7.

If the signal RXIGNOREBTF is asserted High, block type fields not recognized are passed on, whereas if the signal is asserted Low, the error block /E/ is passed on. RXCHARISK is equivalent to RXC when the decoder is enabled.

| | |
|----------------|--------------------|
| RXDEC64B66BUSE | 0 decoder not used |
| | 1 decoder used |

| | | |
|-------------|--|-----------------------------------|
| RXIGNOREBTF | Function 64b/66b decoder used | Function 64b/66b decoder bypassed |
| | 0 unrecognized field types cause /E/ passed on | Undefined |
| | 1 unrecognized field types passed on | |
| RXCHARISK | Equivalent to RXC | Defined by 8b/10b decoder use |

Descrambler

Bypassing

If the signal RXDESCRAM64B66BUSE is deasserted Low, the descrambler is not used.

Normal Operation

If the signal RXDESCRAM64B66BUSE is asserted High, the descrambler is enabled for use. The descrambler uses the polynomial:

$$G(x) = 1 + x^{39} + x^{58}$$

Block Sync

Normal Operation

This block sync design works hand-in-hand with the commaDet block. The commaDet takes as input 32 bits of scrambled and unaligned data from the PMA. It then sends to the block sync the 2-bit sync header, or what it thinks is the sync header based on the current tag value. It asserts `test_sh` which tells the block sync to test the value of the sync header. The block sync analyzes the sync header and if it is valid, increments the `sh_cnt` counter. If the sync header is not a legal value, `sh_cnt` is incremented as well as the counter `sh_invalid_cnt`, and then `bit_slip` is asserted for one clock. The bit slip signal feeds back to the commaDet block and tells it to shift the barrel shifter by one bit. This process of slipping and testing the sync header repeats until block lock is achieved.

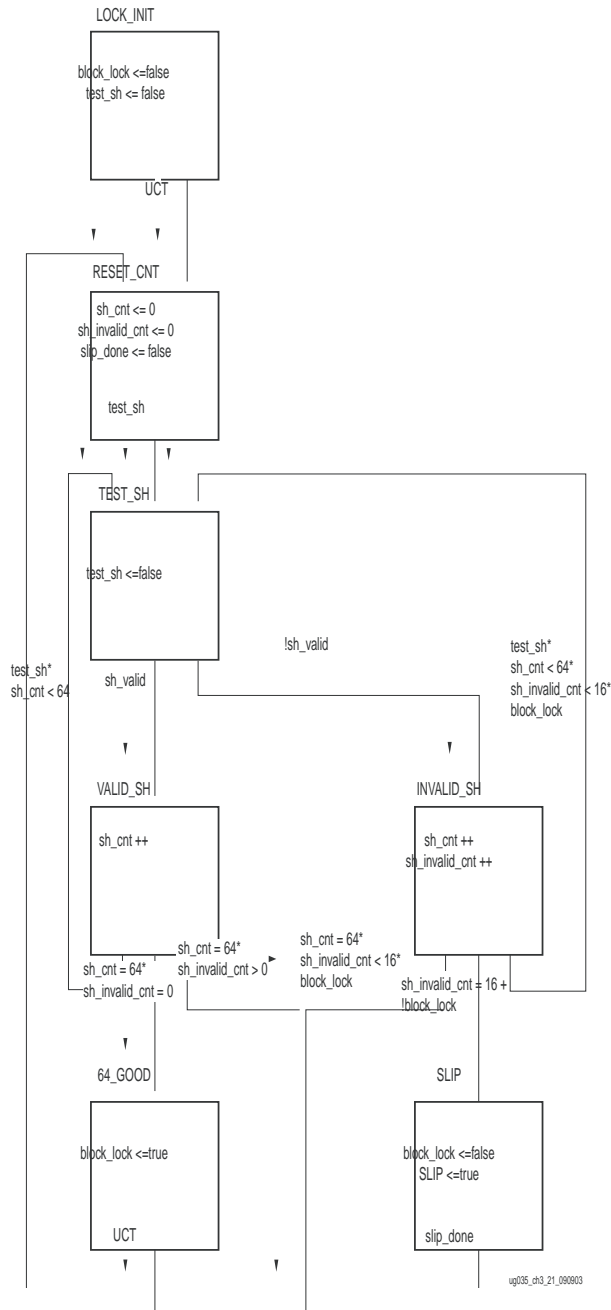


FIGURE 1-11: Block Sync State Machine

The state machine works by keeping track of valid and invalid sync headers. Upon reset, block lock is deasserted, and the state is LOCK_INIT. The next state is RESET_CNT where all counters are zeroed out. When test_sh is asserted, the next state is TEST_SH, which checks the validity of the sync header. If it is valid, the next state is VALID_SH, if not, the state changes to INVALID_SH.

From VALID_SH, if sh_cnt is less than the attribute value sh_cnt_max and test_sh is High, the next state is TEST_SH. If sh_cnt is equal to sh_cnt_max and sh_invalid_cnt equals 0, the next state is GOOD_64 and from there block_lock is asserted. Then the process repeats again and the counters are zeroed.

If at TEST_SH sh_cnt equals sh_cnt_max, but sh_invalid_cnt is greater than zero, then the next state is RESET_CNT. From INVALID_SH, if sh_invalid_cnt equals sh_invalid_cnt_max, or if block_lock is not asserted, the next state is SLIP, where bit_slip is asserted, and then on to RESET_CNT. If sh_cnt equals sh_cnt_max and sh_invalid_cnt is less than sh_invalid_cnt_max and block_lock is asserted, then go back to RESET_CNT without changing block_lock or bit_slip.

Finally, if test_sh is High and sh_cnt is less than sh_cnt_max, and sh_invalid_cnt is less than sh_invalid_cnt_max and block_lock is asserted, go back to the TEST_SH state. **The main thing to note with this state machine is that to achieve block lock, one must receive sh_cnt_max number of valid sync headers in a row without getting an invalid sync header.** However, once block lock is achieved, sh_invalid_cnt_max - 1 number of invalid sync headers can be received within sh_cnt_max number of valid sync headers. Thus, once locked, it is harder to break lock.

Functions Common to All Protocols

Clock Correction

Clock correction is needed when the rate that data is fed into the write side of the receive FIFO is either slower or faster than the rate that data is retrieved from the read side of the receive FIFO. The rate of write data entering the FIFO is determined by the frequency of RXRECCLK. The rate of read data retrieved from the read side of the FIFO is determined by the frequency of RXUSRCLK.

There is one clock correction mode: Append/Remove Idle Clock Correction.

Append/Remove Idle Clock Correction

When the attribute CLK_COR_SEQ_DROP is asserted Low and CLK_CORRECT_USE is asserted High, the Append/remove Idle Clock Correction mode is enabled.

The Append/remove Idle Clock Correction mode corrects for differing clock rates by finding idles in the bitstream, and then either appending or removing idles at the point where the idles were found.

There are a few attributes that need to be set by the user so that the append/remove function can be used correctly. The attribute CLK_COR_MAX_LAT sets the maximum latency through the receive FIFO. If the latency through the receive FIFO exceeds this value, idles are removed so that latency through the receive FIFO is less than CLK_COR_MAX_LAT.

The attribute CLK_COR_MIN_LAT sets the minimum latency through the receive FIFO. If the latency through the receive FIFO is less than this value, idles are inserted so that the latency through the receive FIFO are greater than CLK_COR_MIN_LAT. A correction to the latency due to a CLK_COR_MAX_LAT violation is never less than CLK_COR_MIN_LAT. This is also true for a correction to the latency due to a CLK_COR_MIN_LAT violation; the resulting latency after the correction is greater than CLK_COR_MAX_LAT.

Clock Correction Sequences

Searching within the bitstream for an idle is the core function of the clock correction circuit. The detection of idles starts the correction procedure.

Idles the clock correction circuit should detect are specified by the lower 10 bits of the attributes:

- CLK_COR_SEQ_1_1
- CLK_COR_SEQ_1_2
- CLK_COR_SEQ_1_3
- CLK_COR_SEQ_1_4
- CLK_COR_SEQ_2_1
- CLK_COR_SEQ_2_2
- CLK_COR_SEQ_2_3
- CLK_COR_SEQ_2_4

The 11th bit of each clock correction sequence attribute determines either an 8- or 10-bit compare.

Detection of the clock correction sequence in the bitstream is specified by eight words consisting of 10 bits each. Clock correction sequences can have lengths of 1, 2, 3, 4, or 8 bytes.

When the length specified by the user is between 1 and 4, CLK_COR_SEQ_1_* holds the first pattern to be searched for. CLK_COR_SEQ_1_1 is the least significant byte, which is transmitted first from the transmitter and detected first in the receiver. If CLK_COR_SEQ_2_USE is asserted High when the length is between 1 and 4, the sequence specified by CLK_COR_SEQ_2_* is specified as a second pattern to match. In that case, the pattern specified by sequence 1 *or* sequence 2 matches as a clock correction sequence.

The CLK_COR_SEQ_MASK must have the bits set to a logic 1 mask off the 2 or 3 unused bytes.

When the length specified by the user is eight, CLK_COR_SEQ_1_* holds the first four bytes, while CLK_COR_SEQ_2_* holds the last four bytes. CLK_COR_SEQ_1_1 is the least significant byte, which is transmitted first from the transmitter and detected first in the receiver. CLK_COR_SEQ_2_USE must be asserted High.

The clock correction sequence is a special sequence to accommodate frequency differences between the received data (as reflected in RXRECCLK) and RXUSRCLK. Most of the primitives have these defaulted to the respective protocols. Only the GT_CUSTOM allows this sequence to be set to any specific protocol. The sequence contains 11 bits including the 10 bits of serial data. The 11th bit has two different formats. The typical usage is:

- 0, disparity error required, char is K, 8-bit data value (after 8b/10b decoding, depends on CLK_COR_8B10B_DE)
- 0, 10-bit data value (without 8b/10b decoding, depends on CLK_COR_8B10B_DE)
- 1, xx, sync character (with 64b/66b encoding)
- 1, xx, 8-bit data value

Table 1-19 is an example of data 11-bit attribute setting, the character value, CHARISK value, and the parallel data interface, and how each corresponds with the other.

TABLE 1-19: Clock Correction Sequence/Data Correlation for 16-Bit Data Port

| Attribute Setting | Character | CHARISK | TXDATA (hex) |
|-------------------------------|-----------|---------|--------------|
| CLK_COR_SEQ_1_1 = 00110111100 | K28.5 | 1 | BC |
| CLK_COR_SEQ_1_2 = 00010010101 | D21.4 | 0 | 95 |
| CLK_COR_SEQ_1_3 = 00010110101 | D21.5 | 0 | B5 |
| CLK_COR_SEQ_1_4 = 00010110101 | D21.5 | 0 | B5 |

Notes:

1. CLK_COR_8B10B_DE = TRUE.

Determining Correct CLK_COR_MIN_LAT

To determine the correct CLK_COR_MIN_LAT value, several requirements must be met.

- CLK_COR_MIN_LAT must be less than or equal to 12.
- CLK_COR_MIN_LAT and CLK_COR_MAX_LAT must be multiples of CCS/CBS lengths and ALIGN_COMMA_WORD.
- For symbols less than 8 bytes, $(\text{CLK_COR_MIN_LAT} - \text{CHAN_BOND_LIMIT}) > 12$.
For symbols of 8 bytes, $(\text{CLK_COR_MIN_LAT} - \text{CHAN_BOND_LIMIT}) > 16$.

Channel Bonding

Channel bonding is the technique of tying several serial channels together to create one aggregate channel. Several channels are fed on the transmit side by one parallel bus and reproduced on the receive side as the identical parallel bus. The maximum number of serial differential pairs that can be bonded is 20. Channel bonding is supported by several primitives including GT10_CUSTOM, GT10_INFINIBAND, GT10_XAUI, and GT10_AURORA.

The channel bonding match logic finds CB characters across word boundaries and performs a “comma” style realignment of the data. The data path is byte scrambled until reset as shown below in the example (additional comma alignments will not realign the data). As a result, users should be careful when picking channel bonding characters and should use, in general, special characters that cannot appear in the normal data stream.

Example:

The channel bond character is 0x000000FF. If this sequence of data is sent:

```
000000FF
01020304
05060708
09000000
FF010203
04050607
```

The result is:

```
000000FF
```

```

01020304
05060708
000000FF
01020304
050607xx

```

The bonded channels consist of one master transceiver and 1 to 19 slave transceivers. The CHBONDI/CHBONDO buses of the transceivers are daisy-chained together as shown in Figure 1-12.

When the master transceiver detects a channel bond alignment sequence in its data stream, it signals the slave to perform channel bonding by driving its CHBONDO bus as follows in Table 1-20:

TABLE 1-20: Channel Bond Alignment Sequence

| Detected | CHBONDO Bus |
|-----------------------|--------------------|
| No Channel Bond | XX000 ₂ |
| Channel Bond - Byte 0 | XX100 ₂ |
| Channel Bond - Byte 1 | XX101 ₂ |
| Channel Bond - Byte 2 | XX110 ₂ |
| Channel Bond - Byte 3 | XX111 ₂ |

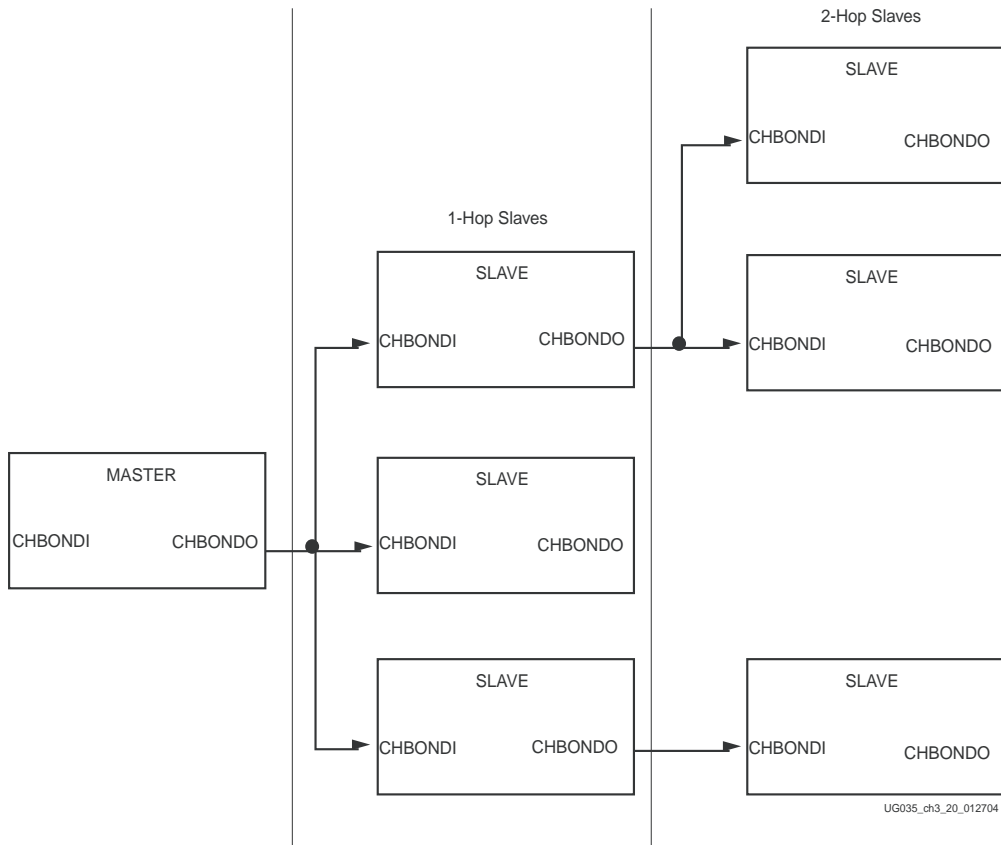


FIGURE 1-12: Daisy-Chained Transceiver CHBONDI/CHBONDO Buses

Whether a slave is a 1-hop or 2-hop slave, internal logic causes the data driven on the CHBONDO bus from the master to be recognized by the slaves at the same time and must be deterministic. Therefore, it is important that the interconnect of CHBONDO-to-CHBONDI not contain any pipeline stages. The data must transfer from CHBONDO to CHBONDI in one clock.

The data streams input to the channel bonded transceivers can be skewed in time from each other. The maximum byte skew that the channel bond logic should allow is set by the attribute MC_CHAN_BOND_LIMIT. During the channel bond operation, the slave receives notification of the master's alignment code location via the CHBONDO bus. If a slave detects the position of its alignment code to be outside the window of CHAN_BOND_LIMIT from the master, then the slave does not perform the channel bond and sets a channel bond error flag. If the channel bond is successful, the slave outputs its skew relative to the master. The skew and channel bond error flag are available on the RXBUFSTATUS bus.

For place and route, the transceiver has one restriction. This is required when channel bonding is implemented. Because of the delay limitations on the CHBONDO to CHBONDI ports, linking of the Master to a Slave_1_hop must run either in the X or Y direction, but not both.

In Figure 1-13, the two Slave_1_hops are linked to the master in only one direction. To navigate to the other slave (a Slave_2_hops), both X and Y displacement is needed. This slave needs one level of daisy-chaining, which is the basis of the Slave_2_hops setting.

Figure 1-13 and Figure 1-14 show the channel bonding mode and linking for an XC2VPX20 and XC2VPX70 devices, which (optionally) contain more transceivers (20) per chip. To ensure the timing is met on the link between the CHBONDO and CHBONDI ports, a constraint must be added to check the time delay.

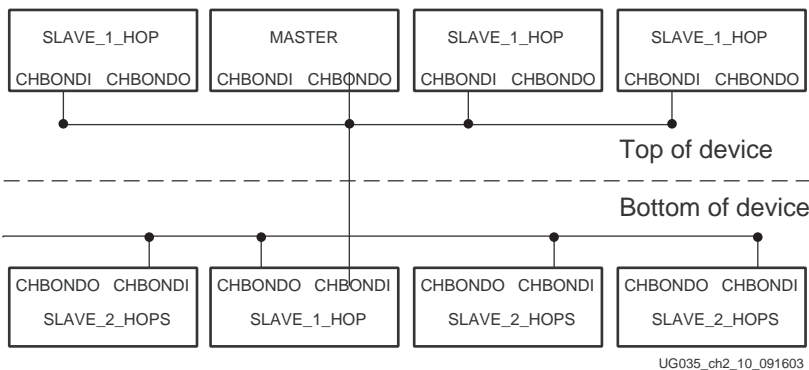


FIGURE 1-13: XC2VPX20 Device Implementation

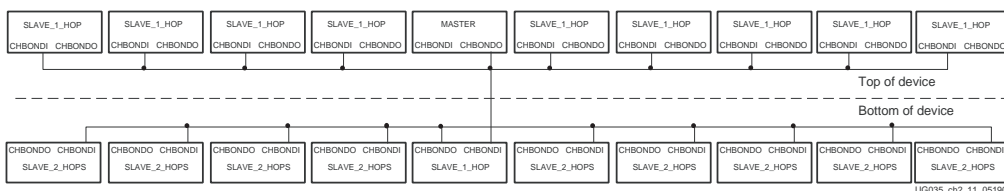


FIGURE 1-14: XC2VPX70 Device Implementation

Status and Event Bus

The Virtex-II Pro X design has merged several signals together to provide extra functionality over the Virtex-II Pro™ design. The signals CHBONDDONE, RXBUFSTATUS, and RXCLKCORCNT were previously used independently of each other to indicate status. In the Virtex-II Pro X design, these signals are concatenated together to provide a status and event bus.

There are two modes of this concatenated bus, status mode and event mode. In status mode, the bus indicates either the difference between the read and write pointers of the receive side FIFO or the skew of the last channel bond event.

Status Indication

In status mode, the RXBUFSTATUS and RXCLKCORCNT pins alternate between the buffer pointer difference and channel bonding skew. The protocol is described by three sequential clocks (STATUS and DATA are one clock in duration) when operating with a 32-bit or 40-bit internal data-width, or

six sequential clocks (STATUS and DATA are two clocks in duration) when operating with a 16-bit or 20-bit internal data width:

<STATUS INDICATOR> <DATA0><DATA1>

where

STATUS INDICATOR can indicate either pointer difference or channel bond skew, DATA0 indicates status data 5:3, and DATA1 indicates status data 2:0.

Table 1-21 shows the signal values for a pointer difference status where the variable pointerDiff[5:0] holds the pointer difference between the receive write and read pointers. If the pointerDiff[5:0] is < 6'b000110, then RXFIFO is almost under flown. If the pointerDiff[5:0] is > 6'b111001, then the RXFIFO is almost over flown.

TABLE 1-21: Signal Values for a Pointer Difference Status

| Status | CHBONDDONE | RXBUFSTATUS | RXCLKCORCNT |
|------------------|------------|-------------|------------------|
| STATUS INDICATOR | 1'b0 | 2'b01 | 3'b000 |
| DATA0 | 1'b0 | 2'b00 | pointerDiff[5:3] |
| DATA1 | 1'b0 | 2'b00 | pointerDiff[2:0] |

Table 1-22 shows the signal values for a channel bonding skew where the variable cbSkew[5:0] holds the pointer difference between the receive write and read pointers:

TABLE 1-22: Signal Values for a Channel Bonding Skew

| Status | CHBONDDONE | RXBUFSTATUS | RXCLKCORCNT |
|------------------|------------|-------------|-------------|
| STATUS INDICATOR | 1'b0 | 2'b01 | 3'b001 |
| DATA0 | 1'b0 | 2'b00 | cbSkew[5:3] |
| DATA1 | 1'b0 | 2'b00 | cbSkew[2:0] |

Event Indication

Two types of events can occur. See Table 1-23. When an event occurs, it can override a status indication. An event can only last for one clock and can be signaled by CHBONDDONE asserting High, or RXBUFSTATUS equating to 2'b10.

TABLE 1-23: Signal Values for Event Indication

| Event | CHBONDDONE | RXBUFSTATUS | RXCLKCORCNT |
|-------------------|------------|-------------|-------------|
| Channel Bond Load | 1'b1 | 2'b00 | 3'b111 |
| Clock Correction | 1'b0 | 2'b10 | 3'bxxx |

An event will always override status, but after an event is completed, status will continue to alternate between the pointer difference and the channel bond skew.

Sample Verilog

The following sample code is to determine underflow or overflow of the RX buffer when 32-bit or 40-bit internal data path is selected.:

```

module status_decoder (
    RXUSRCLK2,
    DCM_LOCKED_N,
    PMARXLOCK,
    CHBONDDONE,
    RXBUFSTATUS,
    RXCLKCORCNT,

    cc_event_insert, // Clock Correction Insertion Event
    cc_event_remove, // Clock Correction Removal Event
    cb_event_load, // Channel Bonding Load Event
    err_event_cc, // Clock Correction Error Event
    err_event_cb, // Channel Bonding Error Event

    pointerDiff, // RX Elastic Buffer Pointer Difference
    rxbuf_almost_err, // RX Elastic Buffer Almost Error

    cbSkew);

input RXUSRCLK2;
input DCM_LOCKED_N;
input PMARXLOCK;
input CHBONDDONE;
input [1:0] RXBUFSTATUS;
input [2:0] RXCLKCORCNT;
output cc_event_insert;
output cc_event_remove;
output cb_event_load;
output err_event_cc;

```

```

        output      err_event_cb;
        output [5:0] pointerDiff;
        output      rxbuf_almost_err;
        output [5:0] cbSkew;

////////////////////////////////////
//
//Signal declaration
////////////////////////////////////
//
reg      cc_event_insert;
reg      cc_event_remove;
reg      cb_event_load;
reg      err_event_cc;
reg      err_event_cb;
reg [5:0] pointerDiff;
reg [2:0] pointerDiff_hi;
reg [1:0] pointerDiff_valid;
reg [5:0] cbSkew;
reg [2:0] cbSkew_hi;
reg [1:0] cbSkew_valid;
reg      rxbuf_almost_err;

wire [5:0] status_event_bus;
wire [2:0] status_bus;

parameter CC_EVENT_INSERT_C = 6'b010001;
parameter CC_EVENT_REMOVE_C = 6'b010000;
parameter CB_EVENT_LOAD_C = 6'b100111;
parameter ERR_EVENT_CC_C = 6'b011000;
parameter ERR_EVENT_CB_C = 6'b011001;

parameter STATUS_INDICATOR_C= 3'b001;
parameter STATUS_DATA_C = 3'b000;

assign status_event_bus = {CHBONDDONE, RXBUFSTATUS[1],
RXBUFSTATUS[0], RXCLKCORCNT[2],
RXCLKCORCNT[1], RXCLKCORCNT[0]};
assign status_bus      = {CHBONDDONE, RXBUFSTATUS[1],
RXBUFSTATUS[0]};

////////////////////////////////////
//
//Logic to decode events
////////////////////////////////////
//
always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
begin

```

```
if (DCM_LOCKED_N) begin
    cc_event_insert <= 1'b0;
    cc_event_remove <= 1'b0;
    cb_event_load   <= 1'b0;
    err_event_cc    <= 1'b0;
    err_event_cb    <= 1'b0;
end
else begin
    cc_event_insert <= status_event_bus == CC_EVENT_INSERT_C;
    cc_event_remove <= status_event_bus == CC_EVENT_REMOVE_C;
    cb_event_load   <= status_event_bus == CB_EVENT_LOAD_C;
    err_event_cc    <= status_event_bus == ERR_EVENT_CC_C;
    err_event_cb    <= status_event_bus == ERR_EVENT_CB_C;

end
end

////////////////////////////////////
//
// Logic to decode the cbSkew value and pointerDiff value
////////////////////////////////////
//
always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
begin
    if (DCM_LOCKED_N) begin
        pointerDiff_valid <= 2'b00;
        cbSkew_valid      <= 2'b00;
    end
    else if ((status_bus == STATUS_INDICATOR_C) & ~RXCLKCORCNT[2] &
~RXCLKCORCNT[1] ) begin
        pointerDiff_valid <= {1'b0, ~RXCLKCORCNT[0]};
        cbSkew_valid      <= {1'b0, RXCLKCORCNT[0]};
    end
    else if (status_bus == STATUS_DATA_C) begin
        pointerDiff_valid[1] <= pointerDiff_valid[0];
        pointerDiff_valid[0] <= 1'b0;
        cbSkew_valid[1]      <= cbSkew_valid[0];
        cbSkew_valid[0]      <= 1'b0;
    end
    else begin // clear the valid signal if the status is interrupted
    by an event.
        pointerDiff_valid <= 2'b00;
        cbSkew_valid      <= 2'b00;
    end
end

always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
begin
```

```

    if (DCM_LOCKED_N || ~PMARXLOCK) begin // reset the value to neutral
position
        pointerDiff <= 32;
        pointerDiff_hi <= 4;
        cbSkew <= 32;
        cbSkew_hi <= 4;
    end
    else if (status_bus == STATUS_DATA_C) begin

        if (pointerDiff_valid[0]) // register higher 3 bits
            pointerDiff_hi <= RXCLKCORCNT;
        else if (pointerDiff_valid[1]) // update entire register when
all 6 bits
are acquired.
            pointerDiff <= {pointerDiff_hi , RXCLKCORCNT};

        if (cbSkew_valid[0]) // register higher 3 bits
            cbSkew_hi <= RXCLKCORCNT;
        else if (cbSkew_valid[1]) // update entire register when all 6
bits are acquired.
            cbSkew <= {cbSkew_hi , RXCLKCORCNT};
    end
end

////////////////////////////////////
//
// Generate RX Elastic Buffer almost error
////////////////////////////////////
//
always @(posedge RXUSRCLK2 or posedge DCM_LOCKED_N)
begin
    if (DCM_LOCKED_N)
        rxbuf_almost_err <= 1'b0;
    else
        rxbuf_almost_err <= (pointerDiff < 6) | (pointerDiff > 57);
end

endmodule

```