Guzik AXIe-1 Local Bus Description

Rev 1.0

Table of Contents

1 Overview 3

2 Architecture 3

2.1 Overview 3

2.2 Reference Clock 4

2.3 Synchronization Line 4

2.4 Transceiver 5

2.4.1 Protocol TX and RX 6

2.4.2 Synchronization 6

2.4.2.1 Synchronizer 6

2.4.2.2 Word Aligner 7

2.4.2.3 Channel Bonder 7

2.4.3 Diagnostics 8

2.4.4 Byte Serializer and Deserializer 9

2.4.5 8B/10B Encoder and Decoder 9

2.4.6 Serializer and Deserializer 9

2.4.7 Clock Data Recovery (CDR) 9

3 Guzik AXIe ADC 6044 Interface 9

3.1 Data Groups 9

3.2 Setup Sequence 10

3.2.1 Reset Sequence 10

3.2.2 Synchronization Sequence 11

3.2.2.1 Word Alignment Sequence 11

3.2.2.2 Channel Bonding Sequence 11

3.2.3 Diagnostics Sequence 12

4 Zone 2 Connector Specification 13

5 Zone 2 Pinout 13

1 Overview

Guzik local bus delivers 40 GBytes/sec data transfer between adjacent blades connected through AXIe zone 2. **Figure 1-1** illustrates the connections between multiple blades.

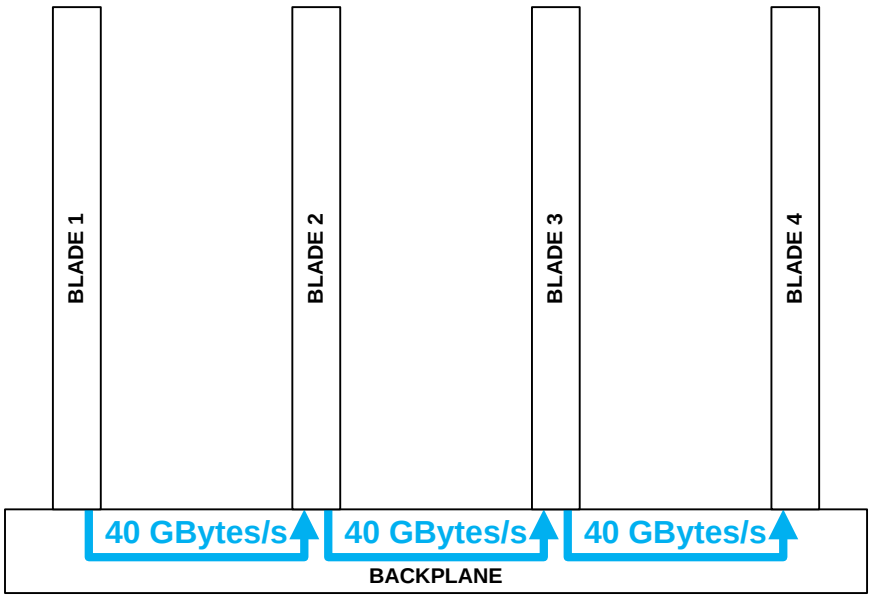

Figure 1-1. Guzik Local Bus Connections Between Multiple Blades

Each blade can transmit data to the next adjacent slot and receive data from the previous adjacent slot. A blade's transmit and receive operations are independent from each other. They can each operate at 40 GBytes/sec simultaneously. Guzik local bus uses dedicated transmitters and receivers, therefore data can only be transferred in one direction between two adjacent boards. However, with a modified backplane, two blades can transfer data between each other in both directions.

2 Architecture

2.1 Overview

Guzik Local bus is a point-to-point connection between two adjacent blades. It contains 62 differential lines. In order to achieve 40 GBytes/sec, 60 lines are used for data transfer. Each data line runs at 6.8 Gbits/sec. The data lines are DC coupled and operate on 1.4-V PCML I/O standard. Guzik local bus uses 8B/10B encoding and user bandwidth is 40 GBytes/sec. However, with a modified backplane, each transceiver can run up to 8.5 Gbits/sec yielding a user bandwidth of 51 GBytes/sec. Next, one differential line is the 340 MHz reference clock. The reference clock is described in ***Reference Clock*** on page 4. Lastly, one differential line is the synchronization signal, which is used for data control handshaking. This is described in ***Synchronization Line*** on page 4. Each blade has one local bus for transmit and one local bus for receive. Each bus is connected to a different adjacent blade. As shown in **Figure 2-1**, Guzik local bus signals are the following:

- 60 data differential lines (6.8 Gbits/sec each line)
- 1 reference clock differential line (340 MHz)
- 1 synchronization differential line

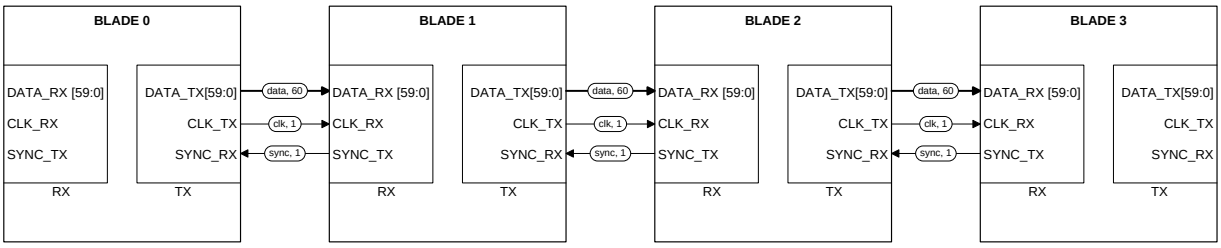

Figure 2-1. Guzik Local Bus

2.2 Reference Clock

The Guzik local bus reference clock is the reference clock for data transfer between adjacent blades. The clock frequency is 340 MHz. The transceiver PLL multiplies the reference clock by 20 to generate a high-speed clock. The high-speed clock is directly related to the Guzik local bus data rates (6.8 Gbits/sec). The reference clock transmitted to an adjacent blade is an AC coupled LVDS I/O standard signal. A blade's transmit and receive circuitry share the same reference clock. It is generated on the transmitter blade with an onboard 340 MHz oscillator. This reference clock is sent to an adjacent blade through zone 2. Receivers should use their adjacent transmitter blades' reference clock from zone 2 instead of their own onboard oscillator. **Figure 2-2** shows the Guzik local bus clock circuitry between two blades.

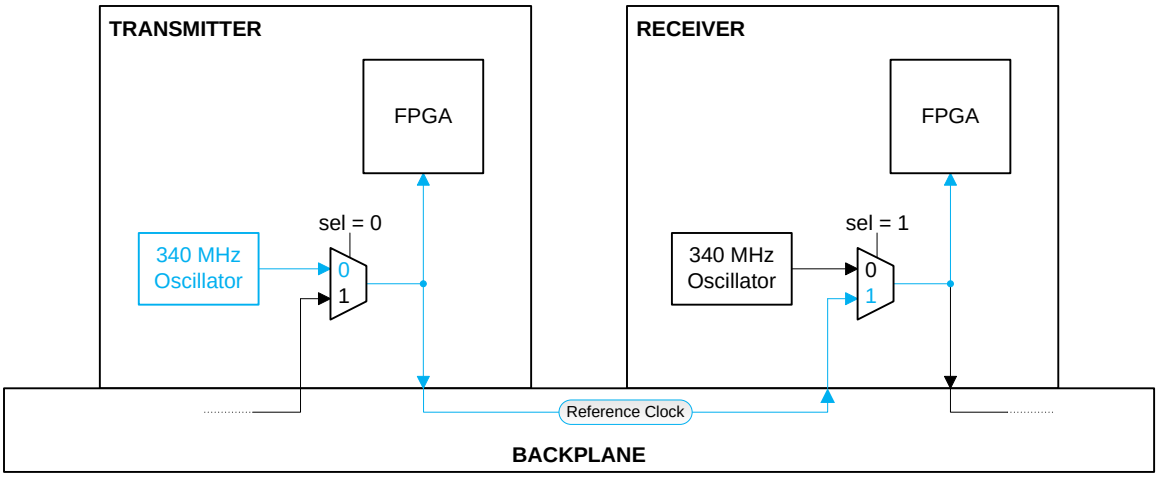

Figure 2-2. Guzik Local Bus Reference Clock

2.3 Synchronization Line

The Guzik local bus synchronization line is used for data control handshaking. This is a DC coupled unidirectional LVDS I/O standard communication line from the receiver to the transmitter. Handshaking can implement by two methods. First, it can be implemented with one asynchronous signal. For instance, a "stop" signal can be sent back to the transmitter to request a stop in data flow. Second, it can be implemented with serial data protocol. This method allows additional information to be sent back to the transmitter. Choosing an implementation method will depend on system requirements. **Figure 2-3** shows how the Guzik local bus synchronization line is connected between two boards.

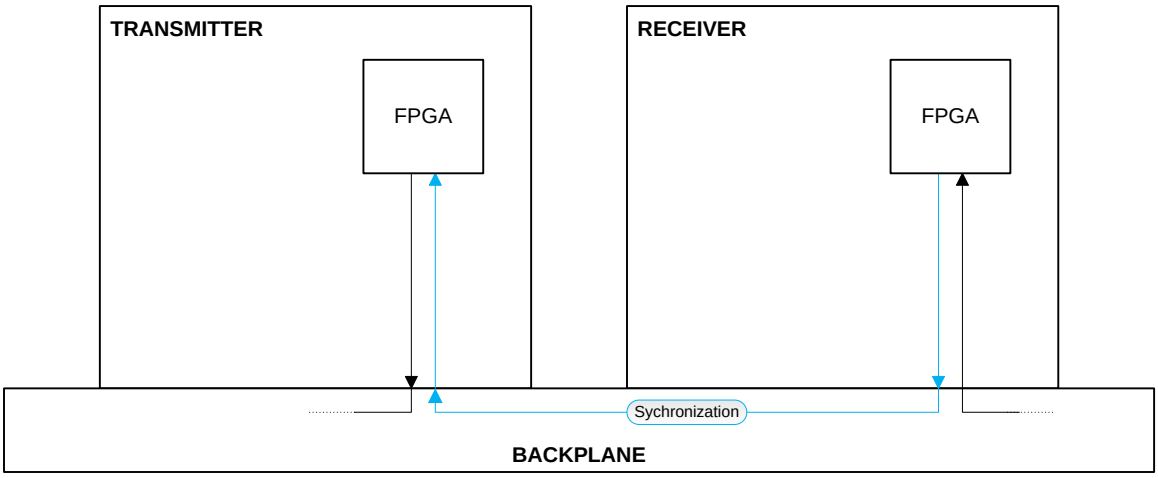

Figure 2-3. Guzik Local Bus Synchronization line

2.4 Transceiver

The transceiver logic behind the Guzik local bus consists of several transmit and receive blocks. A blade instantiates sixty transceivers to drive all sixty transmit and sixty receive data lines. However the user may operate each transceiver independently. The FPGA fabric is running at 170 MHz (half the reference clock speed). For simplicity, this section will describe one transceiver (one channel). **Figure 2-4** shows the block diagram and datapath for a transceiver.

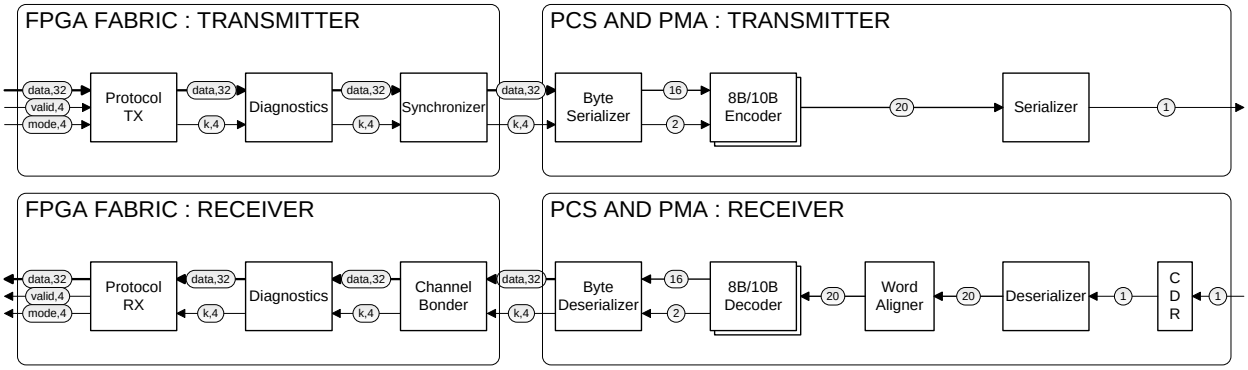

Figure 2-4. Transceiver Block Diagram

- **Transmitter Blocks:**
  - Protocol TX
  - Diagnostics
  - Synchronizer
  - Byte Serializer
  - 8B/10B Encoder
  - Serializer

- **Receiver Blocks:**
  - Protocol RX
  - Diagnostics
  - Channel Bonder
  - Byte Deserializer
  - 8B/10B Decoder
  - Word Aligner
  - Deserializer
  - CDR

2.4.1 Protocol TX and RX

The protocol blocks provide a simple user interface to the transceiver datapath. Both blocks' interfaces include 32-bit data, 4-bit valid and 4-bit mode. The protocol blocks data word size is 32 bits. This word is evenly divided into four groups (one byte per group). Each byte group has a corresponding valid and mode bit. When the valid bit is asserted, the corresponding byte group contains meaningful data. When the mode bit is asserted, the corresponding byte group is converted to a control tag. Typically the mode bit for highest byte group is used to determine that the entire word is a control word. In this case, the mode bit is asserted for highest byte group, while the 3 remaining lower byte groups are de-asserted. Data on the highest byte group is discarded and used as a control tag. The 3 remaining lower byte groups can carry user specified control data. In order to send a control word, all 4 valid bits must also be asserted. Refer to **Figure 2-5** for protocol timing diagram.

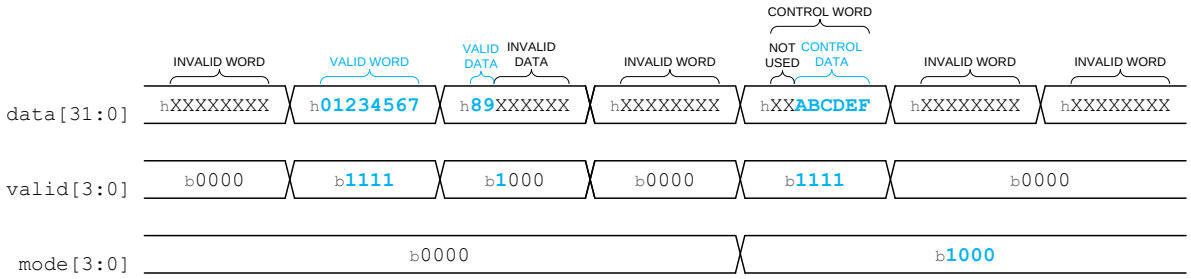

Figure 2-5. Protocol Timing Diagram

2.4.2 Synchronization

Guzik local bus synchronization between two blades is required to ensure data passed into one blade's protocol TX will be the same data out of the adjacent blade's protocol RX. Since data is transmitted serially, certain data misalignment may occur during transmission. This can be due to skew in the physical medium or other electrical inconsistencies. The synchronizer, word aligner, and channel bonder blocks work together to restore the proper alignment. All three blocks utilizes data and k-characters to send and receive special 8B/10B control values for alignment purposes. Synchronization must be performed before any user data transfers. It is the user's responsibility to synchronize the Guzik local bus prior to usage.

2.4.2.1 Synchronizer

The synchronizer block sends special 8B/10B control values to the receiver in order to perform word alignment and channel bonding. A rising edge on `tx_sendwordalign` will send the word alignment pattern (32'h00005C3C) and its k-character (4'b0011) for one clock cycle. Thereafter the synchronizer will send zeros on both data and k-character busses until `tx_sendwordalign` is de-asserted. A rising edge on `tx_sendchanbond` will send channel alignment pattern (32'h0000007C) and its k-character (4'0001) for one clock cycle. Thereafter the synchronizer will send zeros on both data and k-character busses until `tx_sendchanbond` is de-asserted. **Figure 2-6** shows its timing diagram.

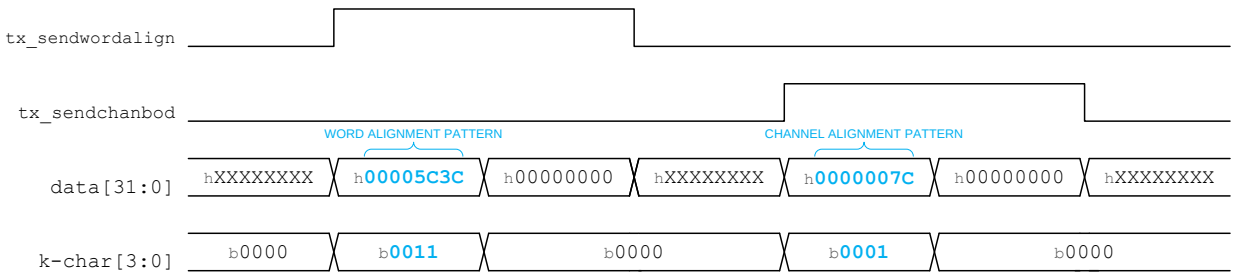

Figure 2-6. Synchronizer Timing Diagram

2.4.2.2 Word Aligner

The word aligner block detects the word boundary on the received parallel data from the deserializer. Since data is transmitter serially, the word boundary of the upstream transmitter is lost upon deserialization. The word aligner restores the word boundary based on a pre-defined alignment pattern. It is located before the 8B/10B decoder, therefore it must word align to a 20-bit 8B/10B encoded pattern. The word alignment pattern (32'h00005C3C) 8B/10B encoded is 20'b10101111001001111100. The word aligner is controlled by the rising edge of `rx_enapatternalign`. A rising edge will trigger the word aligner to look for the alignment pattern. Word alignment will only be performed once after each rising edge. Once word alignment is successful, the `rx_syncstatus` signal will be asserted. It is recommended to de-assert `rx_enapatternalign` after alignment is successful. **Figure 2-7** shows the word aligner input and output timing diagram.

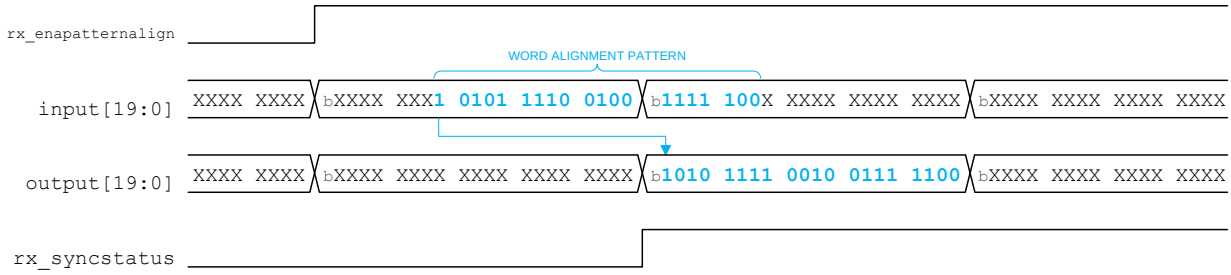

Figure 2-7. Word Aligner Timing Diagram

2.4.2.3 Channel Bonder

The channel bonder block is used to align skewed data across all receiver channels. Data in each channel can be misaligned with respect to one another due to skew in the physical medium. In order to align the channels, a channel alignment pattern 32'h0000007C along with k-character 4'b0001 is defined. All transmitters will synchronously send the channel alignment pattern for one clock cycle (refer to **Figure 2-8**). The channel alignment pattern may arrive at the receivers in different clock cycles (refer to **Figure 2-9**). Channel bonder blocks in each transceiver work together in order to perform channel bonding. First, the channel bonder blocks will count the clock cycle delays relative to each other. Then all channels are bonded based on the delays. Once bonded, the `channelsbonded` signal will be asserted and the channel alignment pattern will be clocked in the same clock cycle for all channels (refer to **Figure 2-10**). All data words received after the channel alignment pattern will also be bonded. Additionally, each k-character words also are bonded along with their respective data words.

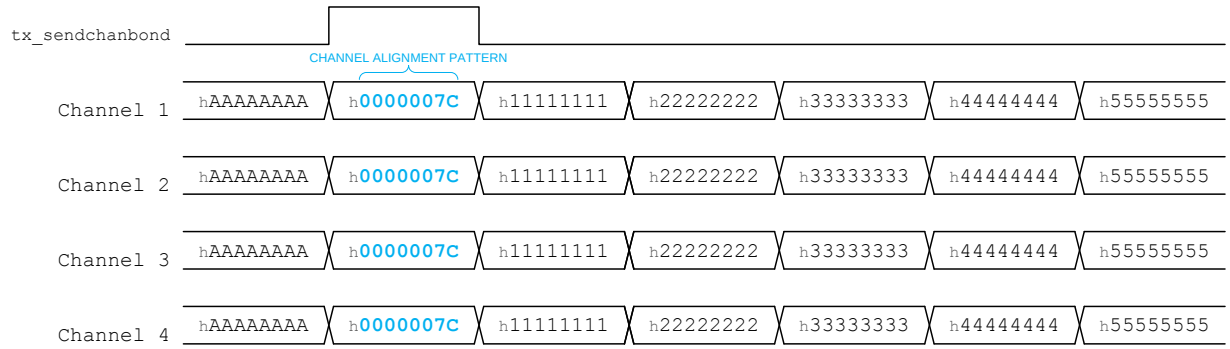

Figure 2-8. Transmitter Output Data

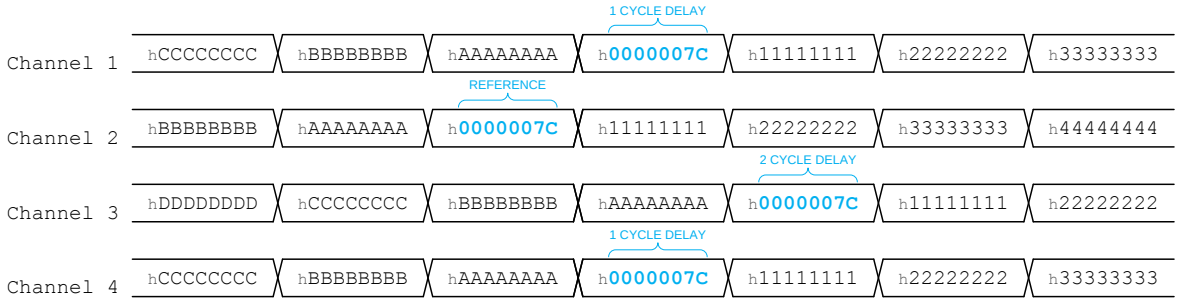

Figure 2-9. Channel Bonder Input Data (Skewed)

Figure 2-10. Channel Bonder Output Data (Bonded)

2.4.3 Diagnostics

Diagnostics is used to assess transmitter and receiver functionality. This is necessary before any data transfers. Both transmitter and receiver have independent diagnostics blocks. The transmitter diagnostics block is enabled by asserting `tx_diag_en`; the receiver diagnostics block is enabled by asserting `rx_diag_en`. The transmitter diagnostics block will send a series of pseudo-random numbers. These pseudo-random numbers are calculated by a predetermined formula. When the receiver diagnostics block receives a value in the series, it will calculate the next value using the predetermined formula. Then it will compare the calculated value to the next value it receives, and so forth. If the received data matches the expected value, then the data is received correctly. Otherwise, it will increment an error counter. The error counter will reset to zero on every rising edge of `rx_diag_en`. In order for correct operation, diagnostics data must arrive at receiver diagnostics block before the receiver diagnostics block begins assessment.

2.4.4 Byte Serializer and Deserializer

The byte serializer converts four bytes datapaths into two bytes datapaths. It will also convert the 4-bit k-characters to 2-bit k-characters. This allows the transceivers to run at higher data rates while the FPGA fabric runs at a lower clock frequency. The byte deserializer has the reversed operation which converts two bytes datapaths to four bytes datapaths. Additionally, it will convert the 2-bit k-characters to 4-bit k-characters.

2.4.5 8B/10B Encoder and Decoder

The 8B/10B encoder generates a 10-bit value from an 8-bit data and 1-bit k-character. The 8B/10B decoder has the reversed operation which generates an 8-bit data and 1-bit k-character from a 10-bit value. Two encoders are used on the transmit side to match the byte serializer output, while two decoders are used on the receiver side to match the byte deserializer input. A typical user does not need to interface directly with the 8B/10B encoders and decoders. Instead, the user will interface directly with the protocol blocks. This simplifies the Guzik local bus usage.

2.4.6 Serializer and Deserializer

The serializer block converts low-speed 20-bit parallel data to high-speed serial data. The output is directly connected to the transmitter buffers. The deserializer block clocks in high-speed serial data from the receiver buffer and converts it to low-speed 20-bit parallel data.

2.4.7 Clock Data Recovery (CDR)

The clock data recovery block is necessary to recover the clock and data from the high-speed serial data stream. The CDR provides high-speed and low-speed clocks used throughout the transceiver.

3 Guzik AXIe ADC 6044 Interface

3.1 Data Groups

The Guzik AXIe ADC 6044 evenly divides the 60 Guzik local bus data lines into four data groups. **Figure 3-1** illustrates how the data groups are divided and connected. Each data group is independently connected to an Altera Stratix IV FPGA device through 15 transceivers. The first data group (Group 1) is comprised of data lines from index 14:0. The second data group (Group 2) is comprised of data lines from 29:15. The third data group (Group 3) is comprised of data lines from 44:30. The fourth data group (Group 4) is comprised of data lines from 59:45. Refer to **Figure 5-1** for a detailed zone 2 pinout. The reference clock and synchronization lines are common for all data groups. The ***Setup Sequence*** on page 10 must be performed separately for each data group. Therefore, channel bonding is also performed independently in each FPGA device.

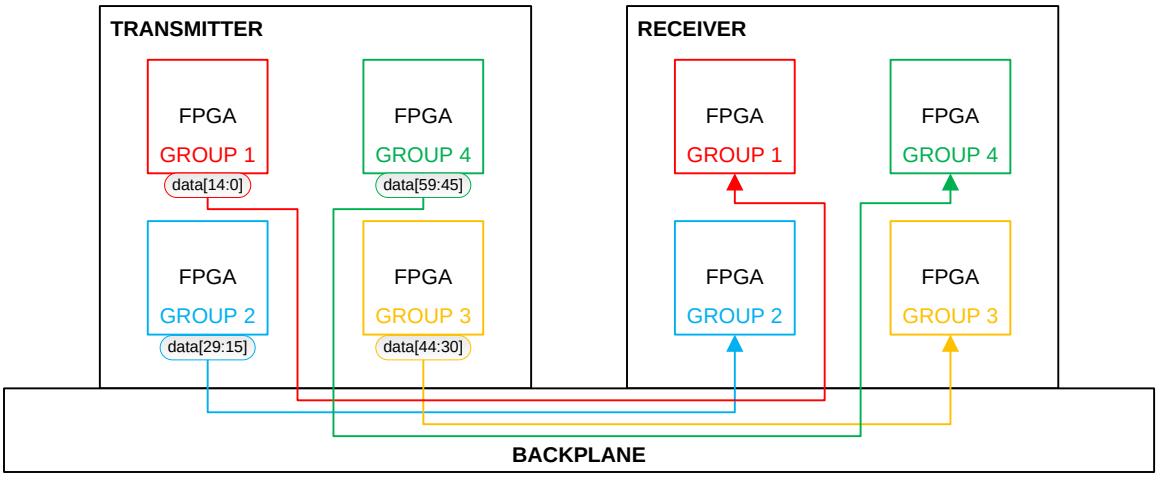

Figure 3-1. Guzik AXIe ADC 6044 Data Groups

3.2 Setup Sequence

This section describes an example setup sequence to prepare two adjacent blades for data transfer. The setup sequence involves a reset sequence, synchronization sequence and a diagnostics sequence. Each sequence must be performed for each of the four data groups. This example uses two Guzik AXIe ADC 6044s.

3.2.1 Reset Sequence

The reset sequence is necessary to properly reset all analog and digital logic. This sequence must be performed prior to any other operation.

1. On the transmitter side, reset all digital logic in transmitter PCS by asserting `tx_digitalreset`.
2. On the receiver side, reset all analog circuitry in receiver PMA by asserting `rx_analogreset`.
3. On the receiver side, reset all digital logic in receiver PCS by asserting `rx_digitalreset`.
4. On the transmitter side, power down the CMU PLLs by asserting `pll_powerdown`.
5. On the receiver side, power down the CMU PLLs by asserting `pll_powerdown`.
6. Wait 1us.
7. On the transmitter side, power up the CMU PLLs by de-asserting `pll_powerdown`.
8. On the receiver side, power up the CMU PLLs by de-asserting `pll_powerdown`.
9. On the transmitter side, poll the PLL status `pll_locked` until CMU PLL is locked to the incoming reference clock frequency
10. On the transmitter side, release digital logic reset for the transmitter PCS by de-asserting `tx_digitalreset`.
11. On the receiver side, poll the reconfiguration clock status `reconfig_busy` until busy signal de-asserts which indicates the reconfiguration clock offset cancellation is complete.
12. On the receiver side, release analog circuitry reset for PMA by de-asserting `rx_analogreset`.
13. On the receiver side, poll the receiver status `rx_freqlocked` until the receiver is in lock-to-data mode.
14. Wait 4000ns.
15. On the receiver side, release digital logic reset for receiver PCS by de-asserting `rx_digitalreset`.

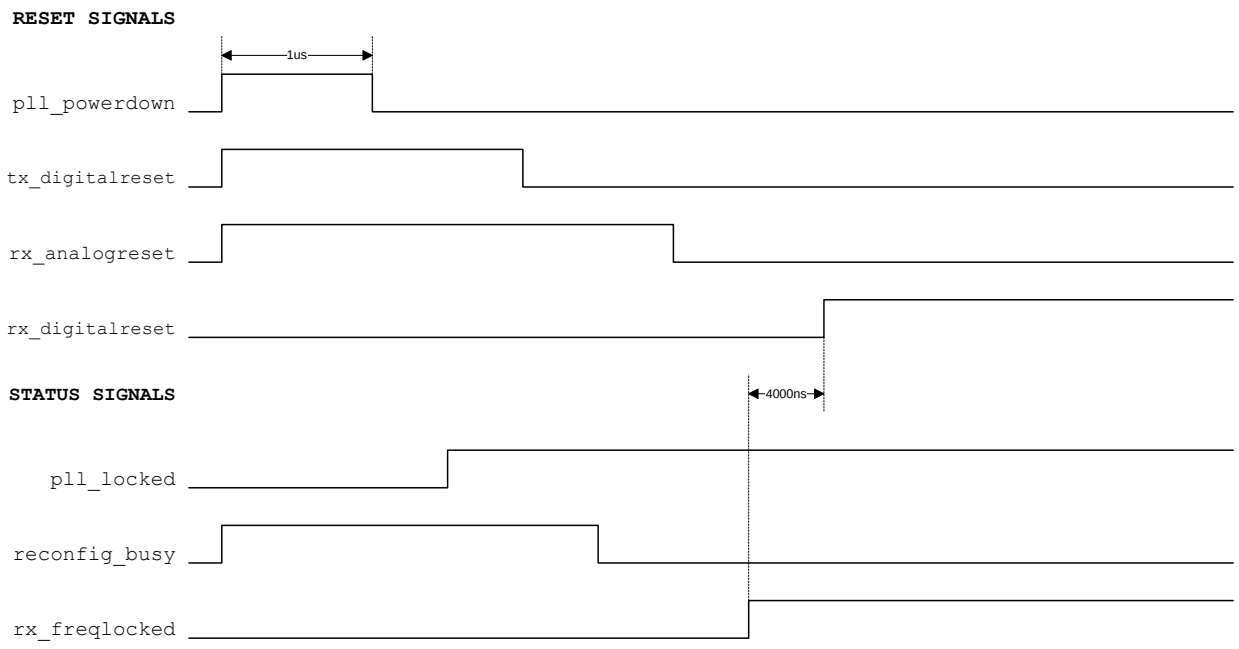

Figure 3-2. Reset Sequence Timing Diagram

3.2.2 Synchronization Sequence

The synchronization sequence comprises of two parts. The first part performs word alignment as described in ***Word Aligner*** on page 7. The second part performs channel bonding as described in ***Channel Bonder*** on page 7.

3.2.2.1 Word Alignment Sequence

1. On the receiver side, enable the word alignment block by asserting `rx_enapatternalign`.
2. On the transmitter side, send a word alignment pattern 32'h00005C3C by asserting `tx_sendwordalign`.
3. On the receiver side, poll word alignment status `rx_syncstatus` until it verifies word boundary has been found.
4. On the receiver side, disable the word alignment block by de-asserting `rx_enapatternalign`. Since the word boundary has been found, disable the word alignment block to avoid accidentally realignment.
5. On the transmitter side, de-assert `tx_sendwordalign`.

3.2.2.2 Channel Bonding Sequence

1. On the transmitter side, send a channel alignment pattern 32'h0000007C by asserting `tx_sendchanbond`.
2. On the receiver side, poll channel alignment status `channelsbonded` until channels are bonded.
3. On the transmitter side, de-assert `tx_sendchanbond`.

3.2.3 Diagnostics Sequence

The diagnostics sequence uses the diagnostics blocks on both blades to perform data verification.

1. On the transmitter side, enable the transmitter diagnostics block to send diagnostics data by asserting `tx_diag_en`.
2. Wait minimum 500ns for data reach receiver. (minimum wait time will vary based on system layout)
3. On the receiver side, enable the receiver diagnostics block to begin data assessment by asserting `rx_diag_en`.
4. Run diagnostics for desired amount of time.
5. On the receiver side, disable the receiver diagnostics block by de-asserting `rx_diag_en`.
6. On the transmitter side, disable the transmitter diagnostics block by de-asserting `tx_diag_en`.

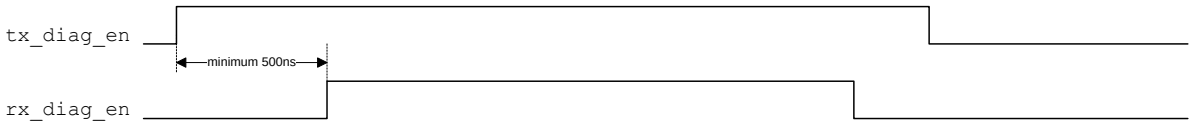

Figure 3-3. Diagnostics Sequence Timing Diagram

4 Zone 2 Connector Specification

The zone 2 connector follows the same mechanical specifications as the AXIe-1 specification. In order to improve the reliability between boards, ZD Plus connectors are used. ZD Plus connectors are rated at 15 Gbits/sec. **Figure 4-1** shows the Guzik local bus connector front view of a blade.

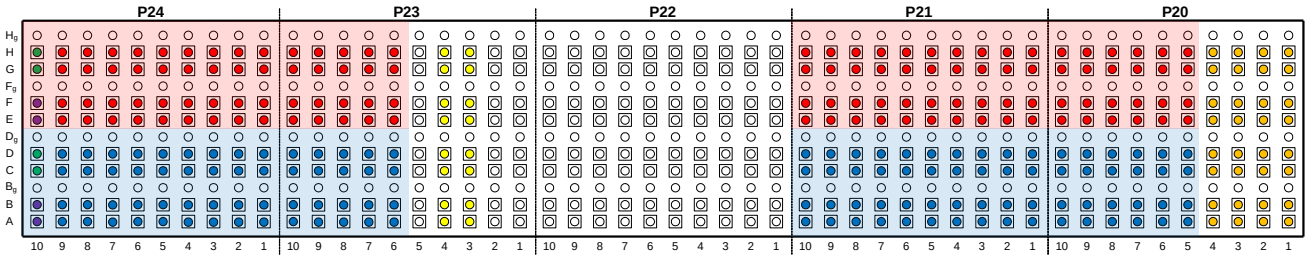

Figure 4-1. Guzik Local Bus Connector (Blade Front View)

5 Zone 2 Pinout

An AXIe zone 2 connector includes 2 sets of Guzik local bus. One bus used to transmit data to the next adjacent blade. The second bus used to receive data from the previous adjacent blade. Refer to **Figure 5-1** for pinout.

		10	9	8	7	6	5	4	3	2	1	
P24	H	GLB_TX_CLK_P	GLB_TX_P<14>	GLB_TX_P<12>	GLB_TX_P<1>	GLB_TX_P<3>	GLB_TX_P<5>	GLB_TX_P<7>	GLB_TX_P<9>	GLB_TX_P<11>	GLB_TX_P<28>	H
	G	GLB_TX_CLK_N	GLB_TX_N<14>	GLB_TX_N<12>	GLB_TX_N<1>	GLB_TX_N<3>	GLB_TX_N<5>	GLB_TX_N<7>	GLB_TX_N<9>	GLB_TX_N<11>	GLB_TX_N<28>	G
	F	GLB_RX_SYNC_P	GLB_TX_P<13>	GLB_TX_P<0>	GLB_TX_P<2>	GLB_TX_P<4>	GLB_TX_P<6>	GLB_TX_P<8>	GLB_TX_P<10>	GLB_TX_P<29>	GLB_TX_P<27>	F
	E	GLB_RX_SYNC_N	GLB_TX_N<13>	GLB_TX_N<0>	GLB_TX_N<2>	GLB_TX_N<4>	GLB_TX_N<6>	GLB_TX_N<8>	GLB_TX_N<10>	GLB_TX_N<29>	GLB_TX_N<27>	E
	D	GLB_RX_CLK_P	GLB_RX_P<14>	GLB_RX_P<12>	GLB_RX_P<1>	GLB_RX_P<3>	GLB_RX_P<5>	GLB_RX_P<7>	GLB_RX_P<9>	GLB_RX_P<11>	GLB_RX_P<28>	D
	C	GLB_RX_CLK_N	GLB_RX_N<14>	GLB_RX_N<12>	GLB_RX_N<1>	GLB_RX_N<3>	GLB_RX_N<5>	GLB_RX_N<7>	GLB_RX_N<9>	GLB_RX_N<11>	GLB_RX_N<28>	C
	B	GLB_TX_SYNC_P	GLB_RX_P<13>	GLB_RX_P<0>	GLB_RX_P<2>	GLB_RX_P<4>	GLB_RX_P<6>	GLB_RX_P<8>	GLB_RX_P<10>	GLB_RX_P<29>	GLB_RX_P<27>	B
	A	GLB_TX_SYNC_N	GLB_RX_N<13>	GLB_RX_N<0>	GLB_RX_N<2>	GLB_RX_N<4>	GLB_RX_N<6>	GLB_RX_N<8>	GLB_RX_N<10>	GLB_RX_N<29>	GLB_RX_N<27>	A
P23	H	GLB_TX_P<15>	GLB_TX_P<17>	GLB_TX_P<19>	GLB_TX_P<21>	GLB_TX_P<23>	RSV_TX_P<4>	CONN_PCIE_RX_N<1>	CONN_PCIE_RX_N<3>	RSV_TX_P<2>	RSV_TX_P<0>	H
	G	GLB_TX_N<15>	GLB_TX_N<17>	GLB_TX_N<19>	GLB_TX_N<21>	GLB_TX_N<23>	RSV_TX_N<4>	CONN_PCIE_RX_P<1>	CONN_PCIE_RX_P<3>	RSV_TX_N<2>	RSV_TX_N<0>	G
	F	GLB_TX_P<16>	GLB_TX_P<18>	GLB_TX_P<20>	GLB_TX_P<22>	GLB_TX_P<24>	RSV_TX_P<5>	CONN_PCIE_TX_N<1>	CONN_PCIE_TX_N<3>	RSV_TX_P<3>	RSV_TX_P<1>	F
	E	GLB_TX_N<16>	GLB_TX_N<18>	GLB_TX_N<20>	GLB_TX_N<22>	GLB_TX_N<24>	RSV_TX_N<5>	CONN_PCIE_TX_P<1>	CONN_PCIE_TX_P<3>	RSV_TX_N<3>	RSV_TX_N<1>	E
	D	GLB_RX_P<15>	GLB_RX_P<17>	GLB_RX_P<19>	GLB_RX_P<21>	GLB_RX_P<23>	RSV_RX_P<4>	CONN_PCIE_RX_N<0>	CONN_PCIE_RX_N<2>	RSV_RX_P<2>	RSV_RX_P<0>	D
	C	GLB_RX_N<15>	GLB_RX_N<17>	GLB_RX_N<19>	GLB_RX_N<21>	GLB_RX_N<23>	RSV_RX_N<4>	CONN_PCIE_RX_P<0>	CONN_PCIE_RX_P<2>	RSV_RX_N<2>	RSV_RX_N<0>	C
	B	GLB_RX_P<16>	GLB_RX_P<18>	GLB_RX_P<20>	GLB_RX_P<22>	GLB_RX_P<24>	RSV_RX_P<5>	CONN_PCIE_TX_N<0>	CONN_PCIE_TX_N<2>	RSV_RX_P<3>	RSV_RX_P<1>	B
	A	GLB_RX_N<16>	GLB_RX_N<18>	GLB_RX_N<20>	GLB_RX_N<22>	GLB_RX_N<24>	RSV_RX_N<5>	CONN_PCIE_TX_P<0>	CONN_PCIE_TX_P<2>	RSV_RX_N<3>	RSV_RX_N<1>	A
P22	H											H
	G											G
	F											F
	E											E
	D											D
	C											C
	B											B
	A											A
P21	H	GLB_TX_P<25>	GLB_TX_P<59>	GLB_TX_P<57>	GLB_TX_P<46>	GLB_TX_P<48>	GLB_TX_P<50>	GLB_TX_P<52>	GLB_TX_P<54>	GLB_TX_P<56>	GLB_TX_P<43>	H
	G	GLB_TX_N<25>	GLB_TX_N<59>	GLB_TX_N<57>	GLB_TX_N<46>	GLB_TX_N<48>	GLB_TX_N<50>	GLB_TX_N<52>	GLB_TX_N<54>	GLB_TX_N<56>	GLB_TX_N<43>	G
	F	GLB_TX_P<26>	GLB_TX_P<58>	GLB_TX_P<45>	GLB_TX_P<47>	GLB_TX_P<49>	GLB_TX_P<51>	GLB_TX_P<53>	GLB_TX_P<55>	GLB_TX_P<44>	GLB_TX_P<42>	F
	E	GLB_TX_N<26>	GLB_TX_N<58>	GLB_TX_N<45>	GLB_TX_N<47>	GLB_TX_N<49>	GLB_TX_N<51>	GLB_TX_N<53>	GLB_TX_N<55>	GLB_TX_N<44>	GLB_TX_N<42>	E
	D	GLB_RX_P<25>	GLB_RX_P<59>	GLB_RX_P<57>	GLB_RX_P<46>	GLB_RX_P<48>	GLB_RX_P<50>	GLB_RX_P<52>	GLB_RX_P<54>	GLB_RX_P<56>	GLB_RX_P<43>	D
	C	GLB_RX_N<25>	GLB_RX_N<59>	GLB_RX_N<57>	GLB_RX_N<46>	GLB_RX_N<48>	GLB_RX_N<50>	GLB_RX_N<52>	GLB_RX_N<54>	GLB_RX_N<56>	GLB_RX_N<43>	C
	B	GLB_RX_P<26>	GLB_RX_P<58>	GLB_RX_P<45>	GLB_RX_P<47>	GLB_RX_P<49>	GLB_RX_P<51>	GLB_RX_P<53>	GLB_RX_P<55>	GLB_RX_P<44>	GLB_RX_P<42>	B
	A	GLB_RX_N<26>	GLB_RX_N<58>	GLB_RX_N<45>	GLB_RX_N<47>	GLB_RX_N<49>	GLB_RX_N<51>	GLB_RX_N<53>	GLB_RX_N<55>	GLB_RX_N<44>	GLB_RX_N<42>	A
P20	H	GLB_TX_P<30>	GLB_TX_P<32>	GLB_TX_P<34>	GLB_TX_P<36>	GLB_TX_P<38>	GLB_TX_P<40>	CLK100_N	TRIG_N<10>	PCIE_REF_CLK_N	TRIG_N<3>	H
	G	GLB_TX_N<30>	GLB_TX_N<32>	GLB_TX_N<34>	GLB_TX_N<36>	GLB_TX_N<38>	GLB_TX_N<40>	CLK100_P	TRIG_P<10>	PCIE_REF_CLK_P	TRIG_P<3>	G
	F	GLB_TX_P<31>	GLB_TX_P<33>	GLB_TX_P<35>	GLB_TX_P<37>	GLB_TX_P<39>	GLB_TX_P<41>	SYNC100_N	TRIG_N<9>	TRIG_N<6>	TRIG_N<2>	F
	E	GLB_TX_N<31>	GLB_TX_N<33>	GLB_TX_N<35>	GLB_TX_N<37>	GLB_TX_N<39>	GLB_TX_N<41>	SYNC100_P	TRIG_P<9>	TRIG_P<6>	TRIG_P<2>	E
	D	GLB_RX_P<30>	GLB_RX_P<32>	GLB_RX_P<34>	GLB_RX_P<36>	GLB_RX_P<38>	GLB_RX_P<40>	STRIG_N	TRIG_N<8>	TRIG_N<5>	TRIG_N<1>	D
	C	GLB_RX_N<30>	GLB_RX_N<32>	GLB_RX_N<34>	GLB_RX_N<36>	GLB_RX_N<38>	GLB_RX_N<40>	STRIG_P	TRIG_P<8>	TRIG_P<5>	TRIG_P<1>	C
	B	GLB_RX_P<31>	GLB_RX_P<33>	GLB_RX_P<35>	GLB_RX_P<37>	GLB_RX_P<39>	GLB_RX_P<41>	TRIG_N<11>	TRIG_N<7>	TRIG_N<4>	TRIG_N<0>	B
	A	GLB_RX_N<31>	GLB_RX_N<33>	GLB_RX_N<35>	GLB_RX_N<37>	GLB_RX_N<39>	GLB_RX_N<41>	TRIG_P<11>	TRIG_P<7>	TRIG_P<4>	TRIG_P<0>	A
		10	9	8	7	6	5	4	3	2	1	

Figure 5-1. Guzik Local Bus Pinout

2443 Wyandotte Street
Mountain View, CA 94043
Phone: (650) 625-8000
Fax: (650) 625-9325
E-mail: modularsales@guzik.com
http://www.guzik.com/
https://www.twitter.com/GuzikTest
http://www.facebook.com/GuzikTest