XAPP670 (v1.0) June 10, 2003

Minimizing Receiver Elastic Buffer Delay in the Virtex-II Pro RocketIO Transceiver

Author: Jeremy Kowalczyk

Summary

This application note describes a design that reduces latency through the receive elastic buffer of the Virtex-II Pro™ RocketIO™ transceiver. This note is only applicable for designs that do not use the clock correction or channel bonding features of the RocketIO transceiver. (These operations can still be done in the fabric, if needed).

Applications

The design described in this application note can be used in any application that requires low latency through the receive side of the transmitter. This includes:

- *High-performance switches*. Latency to route a packet from one RocketIO block to another is reduced.
- *Storage Area Networks.* Latency to read/write data from/to disk is reduced. Reduced access time leads to a faster overall system.
- *Radio Network Controllers*. Latency to read/write configuration data to the controller is reduced.
- *Fibre Channel Arbitrated Loop.* The arbitrated loop topology requires very low latency through each node on the loop. Data must be received and retransmitted by any members of the loop that are between 2 communicating nodes. By using this application note, the overall latency of transmitting through a node is reduced, reducing data access time for the entire loop.

Introduction

The RocketIO receiver section contains an elastic buffer 64 words deep by 13 bits wide. The 13-bit data width comprises either 8 bits of decoded 8B/10B data or 10 bits of raw data, plus some status bits. Figure 1 shows how the elastic buffer fits into the RocketIO transceiver architecture.

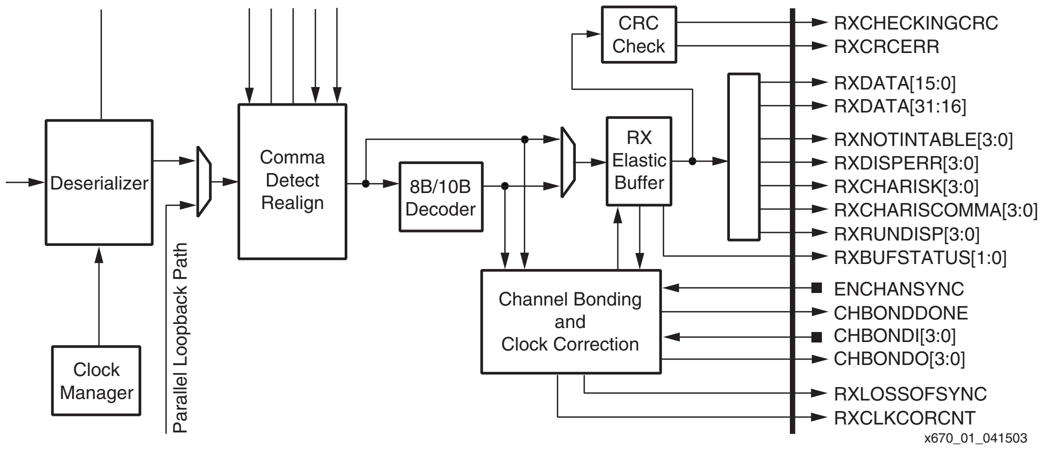

Figure 1: **RocketIO Receive Elastic Buffer and Associated Functional Blocks**

For the complete RocketIO transceiver block diagram and further information on the elastic buffer, refer to the *RocketIO Transceiver User Guide*, available on the Xilinx website.

The elastic buffer is a FIFO with movable read and write pointers to accommodate special features of the transceiver. Data is written and read two bytes at a time, so the FIFO actually holds either 32 x 2 bytes (8B/10B decoding used), or 32 x 2 10-bit words (8B/10B bypassed). On average, the elastic buffer adds about 18 RXUSRCLK cycles of delay from RXP/RXN to RXDATA. This FIFO has three purposes:

- Cross clock domains, from that of the clock recovered from the data stream (RXRECCLK) to the clock on the RocketIO block user interface (RXUSRCLK)
- Allow for channel bonding
- Allow for clock correction

Many applications that do not use clock correction and channel bonding cannot tolerate this amount of delay. The RocketIO attribute RX_BUFFER_USE could be set to FALSE to cause the buffer to be bypassed, but if this is done, the phase relationship between the RocketIO block RXRECCLK and the RXRECCLK fed back into the RXUSRCLK port will be unknown, due to the unavoidable routing delay presented by the FPGA fabric. This situation is illustrated in Figure 2. For this reason, the attributes table in Chapter 1 of the *RocketIO Transceiver User Guide* recommends that RX_BUFFER_USE always be set to TRUE. Bypassing the receive elastic buffer in this manner is normally a poor design choice.

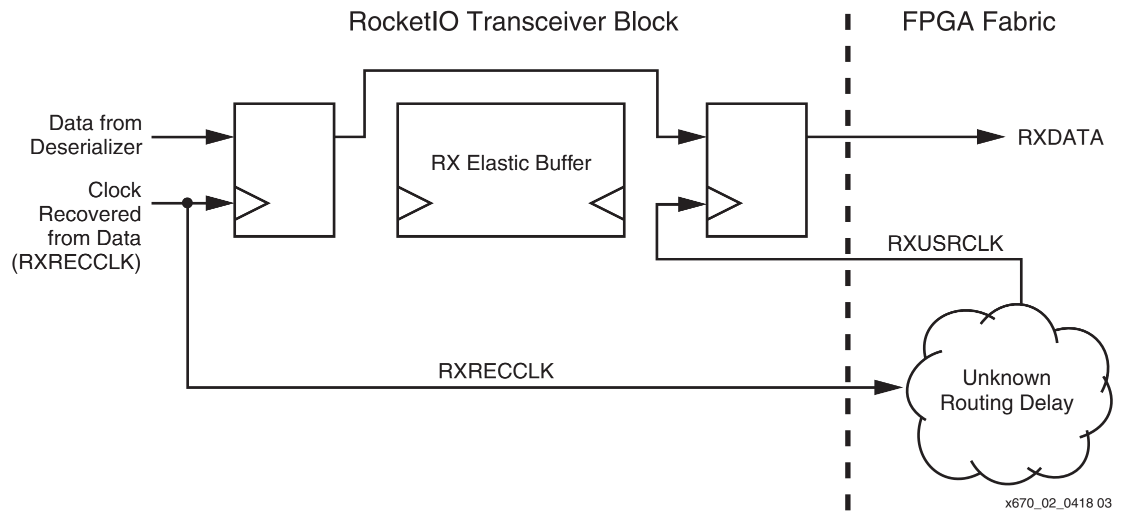

Figure 2: **Unknown FPGA Fabric Routing Delay (RX Elastic Buffer Bypassed)**

Latency can still be reduced through the buffer, however. Upon reset or comma alignment (if used), the receive elastic buffer's read and write pointers are set 32 characters (or 16 two-character words) apart. See Figure 3.

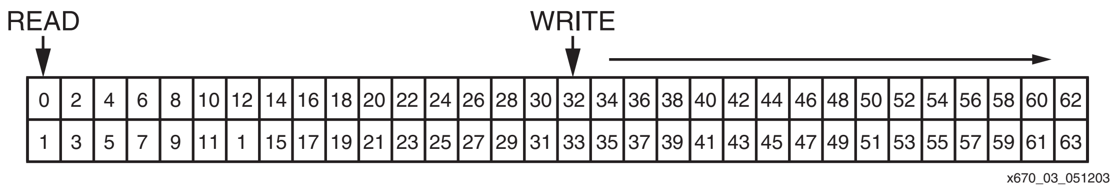

Figure 3: **Initial Read/Writer Pointer Positions**

By stopping the read pointer from moving while the write pointer wraps around in front of it, the read and write pointers can be placed one to two words apart, depending on the phase relationship between read and write clocks of the buffer. This reduces the latency to one or two RXUSRCLK cycles through the buffer. This is illustrated in Figure 4 and Figure 5.

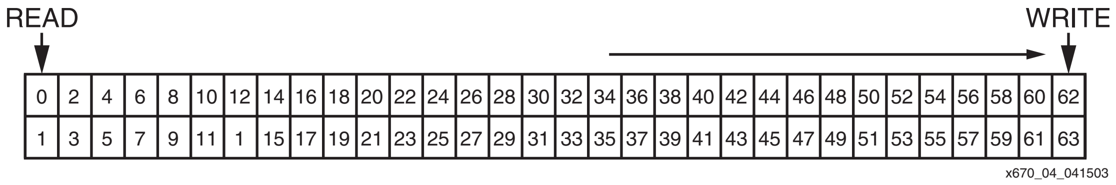

Figure 4: **Read/Write Pointer Positions After 15 Write Clocks without a Read**

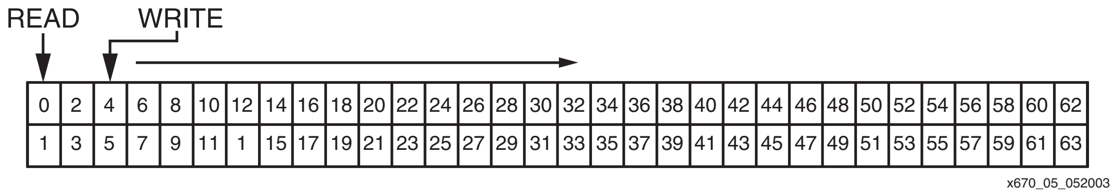

Figure 5: **Read/Write Pointer Positions After 18 Write Clocks without a Read**

To keep the read pointer from moving, the RXUSRCLK must be stopped for the appropriate number of cycles. Waiting 18 cycles will place the read and write pointers zero or one read away from each other, depending on the phase relationship between the RXUSRCLK and RXRECCLK. This minimizes latency, yet still safely crosses clock domains.

The reference design accompanying this application note contains the min_delay module, which accomplishes this buffer pointer manipulation.

Interface Connections

Figure 6 illustrates the interface provided for moving of the write pointer relative to the read pointer. Table 1 defines the signals used.

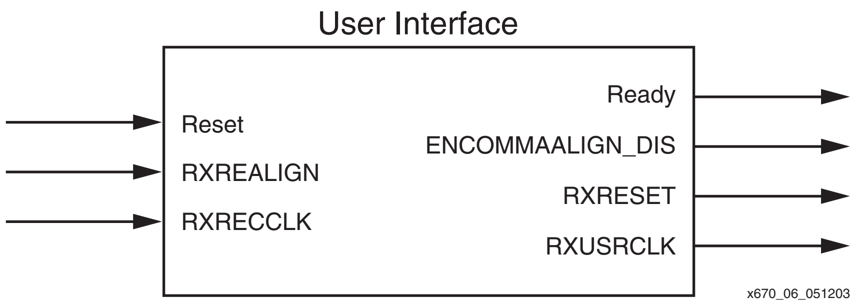

Figure 6: **min_delay Module User Interface**

Table 1: **User Interface Signals Defined**

Signal Name	Description
Reset	Used to reset the RX side of the transceiver. Each time the RX side is reset, the buffer pointers are reset, so their relative positions must be changed to minimize latency.
RXREALIGN	Connected to the RXREALIGN port of the RocketIO transceiver block. Each time realignment occurs, the buffer pointers are reset, so their relative positions must be changed to minimize latency. (Tied Low if comma alignment is not used.)
RXRECCLK	Can be connected directly from the RXRECCLK output of the RocketIO transceiver block, or via a BUFG or DCM. See "Design Usage Notes" for details. This is the main clock used by the design. It is used to clock data out of the RocketIO block and into the FPGA fabric.
Ready	When asserted High, this signals informs the user application that the buffer pointers have been moved and that the RocketIO block is ready for normal operation.
ENCOMMAALIGN_DIS	Following a reset or RXREALIGN, comma alignment must be disabled until the buffer pointers are moved. This signal informs the user design when it cannot dynamically enable/disable comma alignment via the ENP/ENMCOMMAALIGN ports of the RocketIO block. (Unused if comma alignment is not used.)
RXRESET	Connected to the RXRESET port of the RocketIO block. Asserts the RXRESET for two RXUSRCLK cycles, as required.
RXUSRCLK	Connected to the RXUSRCLK port of the RocketIO block, and optionally to related fabric logic. This is a global clock tree output, driven by a BUFGCE.

If comma alignment is used, the buffer pointers must be repositioned each time a realign occurs. If comma alignment is not used, the buffer pointers only need to be repositioned on a reset of the receive side.

Figure 7 shows the connections when comma alignment is used. Note that the FDRE in this diagram should be locked during implementation near the GT[1] block used. See "SERDES Alignment" in Chapter 3 of the *RocketIO Transceiver User Guide* for more information.

Figure 8 shows the connections when comma alignment is *not* used.

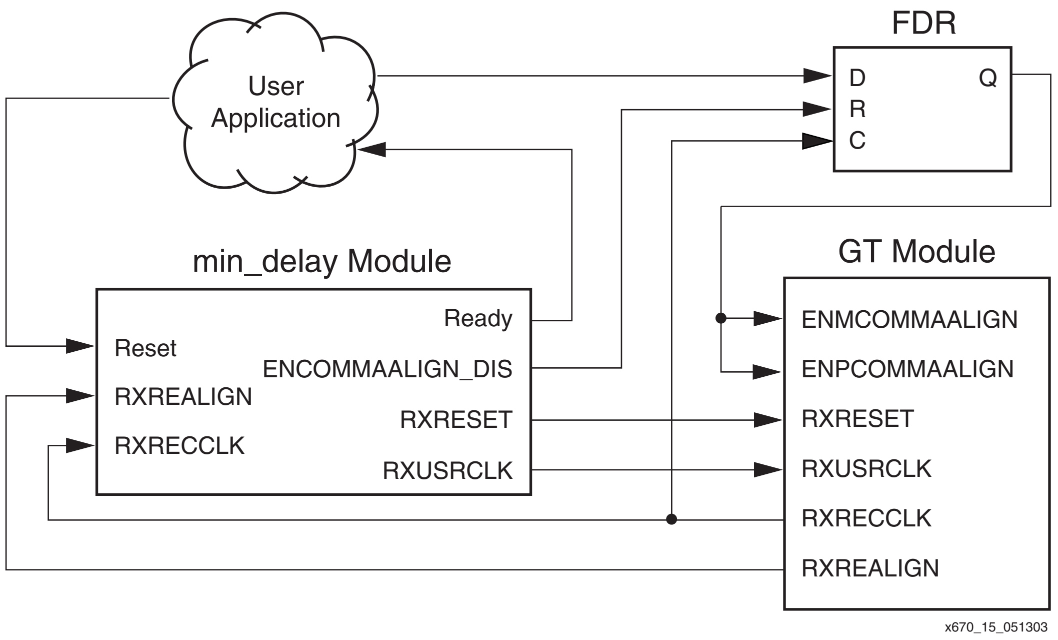

Figure 7: **Comma Alignment Used**

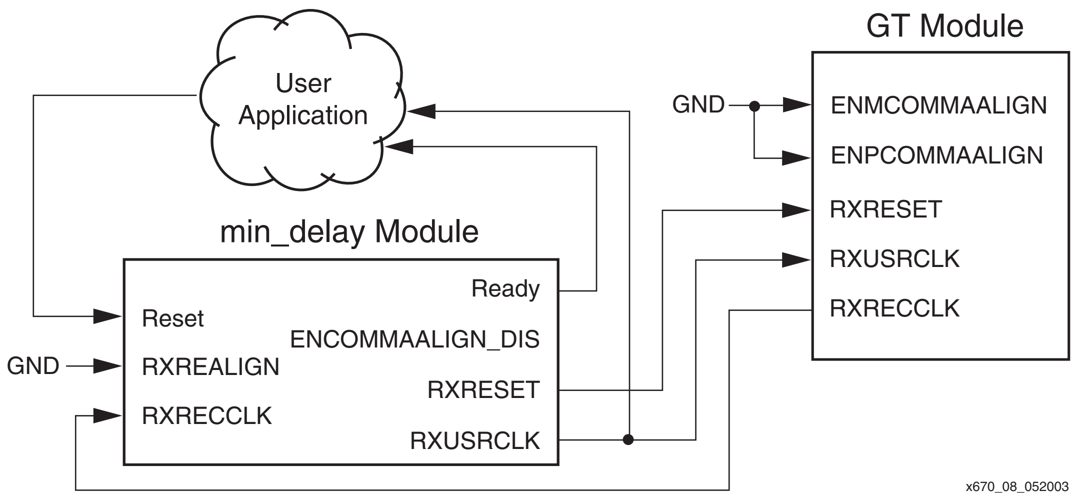

Figure 8: **Comma Alignment Not Used**

1. One of the "GT" primitives, such as GT_CUSTOM. See the RocketIO *Transceiver User Guide* for details.

Basic Operation

The design consists of a state machine which controls a BUFGCE (a global buffer with a clock enable). The input of that BUFGCE is the RXRECCLK. The output is fed to the RXUSRCLK port. The state machine makes sure the GT component:

a. is reset properly (by asserting RXRESET for two cycles);

b. waits for the internal pointer resets to deassert (by waiting twelve additional cycles);

c. realigns the write pointer as described above; and

d. asserts ready.

Figure 9 illustrates these four states and their respective timing.

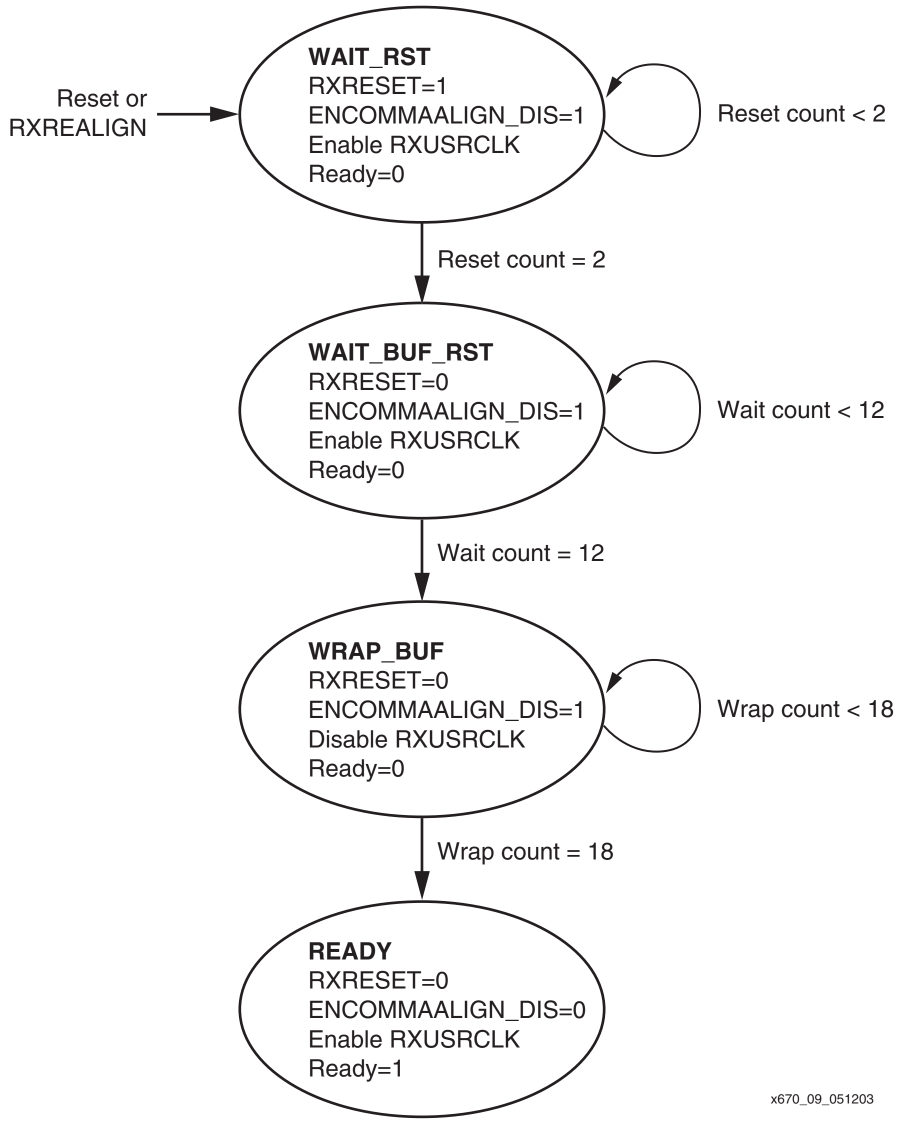

Figure 9: **State Machine for min_delay Module**

Operation Waveforms

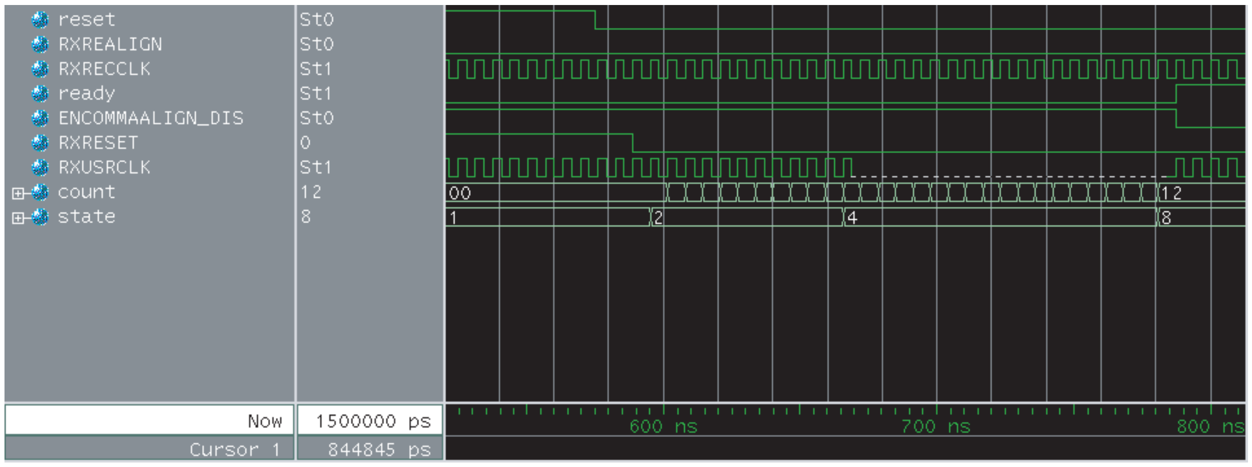

Figure 10: **On Reset**

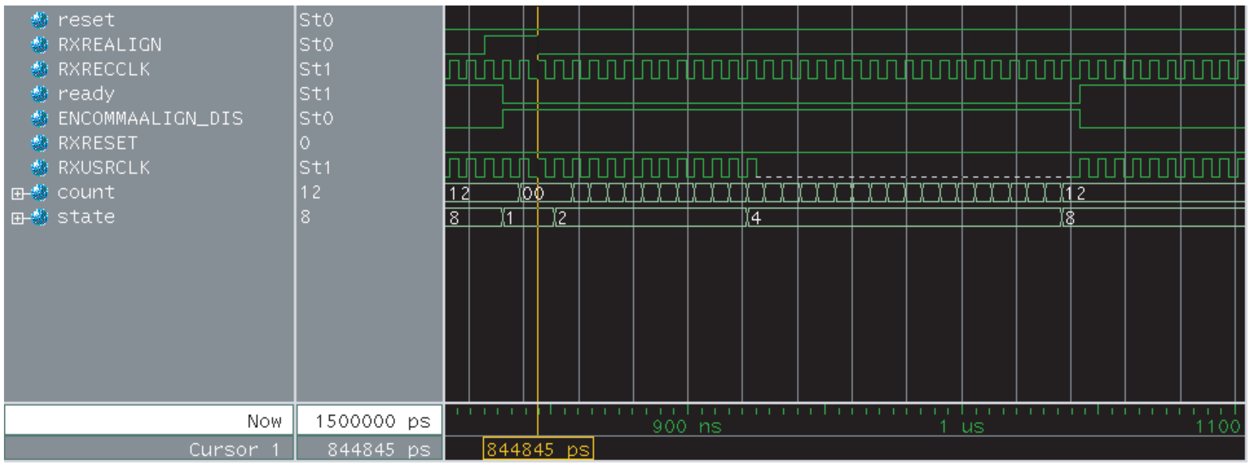

Figure 11: **On RXREALIGN**

Design Usage Notes

Normally, one min_delay module must be used for each transceiver where latency must be reduced. However, one min_delay module can be used for multiple RocketIO receivers if comma alignment is not used and the incoming serial rates of each receiver match exactly. This can only be the case when the transmitters that drive the RocketIO receivers serialize their data streams with the same clock. Figure 12 illustrates how to connect multiple transceivers to one min_delay module.

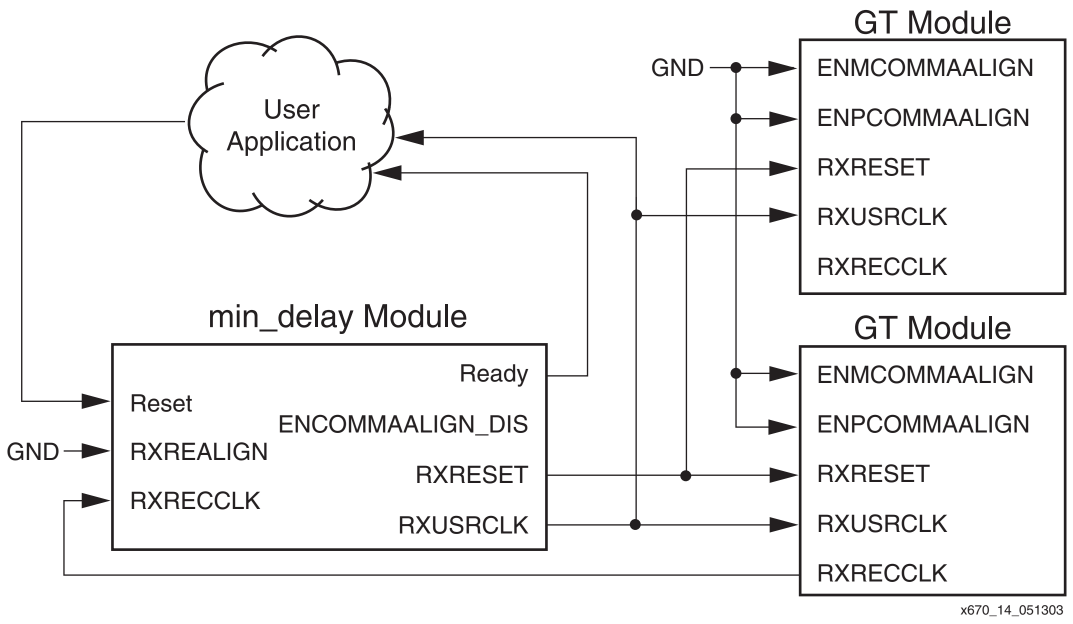

Figure 12: **Multiple Transceivers, Comma Alignment Not Used**

Due to the moving of buffer pointers, the user must wait about 30 RXUSRCLK cycles after a reset or RXREALIGN before the transceiver is ready for operation.

The logic that constitutes the main state machine is minimal, so low-skew lines have been used for the clock that controls it (the net 'CLK' in the design, which is driven by the RXRECCLK input port of the min_delay module). A global buffer (BUFG) can be used instead, if there is one available, as it provides more predictable skew characteristics. Note that the RXUSRCLK output of the module comes from a BUFGCE, and therefore is already on global routing.

RXUSRCLK must be aligned on the falling edge with RXUSRCLK2. (See the Clocking section in the *RocketIO Transceiver User Guide* for more details.) In the case of a 2-byte data path (GT component attribute RX_DATA_WIDTH set to 2), RXUSRCLK and RXUSRCLK2 should have the same frequency and phase. This allows the min_delay module's RXUSRCLK output to be fed to both RXUSRCLK and RXUSRCLK2 ports. If a 1-byte data path is used, RXUSRCLK2 is twice the frequency of RXUSRCLK and 180° out of phase. If a 4-byte data path is used, RXUSRCLK2 is half the frequency of RXUSRCLK, and 180° out of phase. A DCM can be used to produce these clocks in the configurations shown in Figure 13 and Figure 14.

Note: The RXBUFSTATUS[1] port of the RocketIO transceiver will indicate overflow, and should be ignored when using the min_delay module.

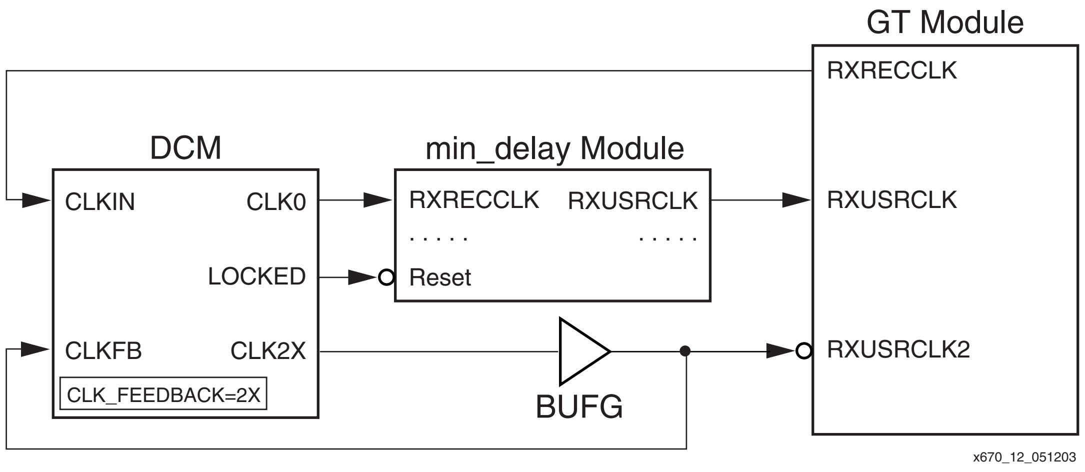

Figure 13: **Configuration for 1-Byte Data Path**

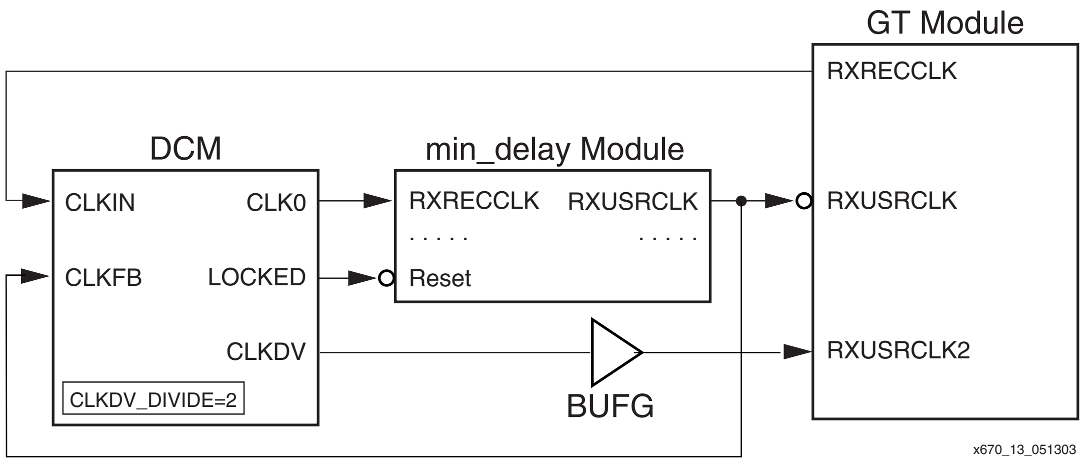

Figure 14: **Configuration for 4-Byte Data Path**

In all three data path width cases mentioned above (1-, 2-, and 4-byte), the GT component attribute SERDES_10B can be set to either TRUE or FALSE. When SERDES_10B = FALSE, REFCLK, RXRECCLK, and RXUSRCLK are all the same frequency. In the case of SERDES_10B = TRUE, RXUSRCLK is half the frequency of REFCLK (or BREFCLK). However, the RXRECCLK output is half the frequency of REFCLK (or BREFCLK) when SERDES_10B = TRUE. This will automatically divide by two the RXUSRCLK and RXUSRCLK2 outputs in Figure 13 and Figure 14.

Appropriate timing constraints must be placed on the RXRECCLK and RXUSRCLK/RXUSRCLK2 paths. The RXRECCLK must have the same frequency as the RXUSRCLK required for the desired data rate. The BUFGCE must be locked on the same side of the chip as the GT component that it drives. This is to minimize route delay to the enable pin of the BUFGCE from the logic that enables/disables it. A MAXDELAY constraint should be used on the enable net to direct the router to keep this delay to be less than half the clock period. This guarantees the clock is disabled at the appropriate time. An example constraint appears in the UCF file included with the design.

Logic Utilization (min_delay module):

Number of Slice Flip-Flops: 14
Number of 4 input LUTs: 11
Number of BUFGMUXs: 1

Conclusion

The receive elastic buffer in the RocketIO transceiver is present to accommodate the clock correction and channel bonding features while crossing clock domains from the recovered clock to the user clock. If clock correction and channel bonding are not needed, the buffer causes a large delay (18 RXUSRCLK cycles) with little benefit. Using the min_delay module described in this Application Note, the latency through the buffer can be reduced by moving the buffer's write pointer just in front of the read pointer.

Operation of both the Verilog and VHDL versions of this design has been verified on the ML321 Characterization Board for a single GT component with the following configurations:

- 1-byte, 2-byte, and 4-byte data widths with comma alignment and 8B/10B encoding enabled.
- 10-bit, 20-bit, and 40-bit data widths without using comma alignment and without 8B/10B encoding.

HDL Code

HDL code (Verilog and VHDL) for this design can be downloaded from the Xilinx FTP site at:

ftp://ftp.xilinx.com/pub/applications/xapp/xapp670.zip

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/10/03	1.0	Initial Xilinx release.