Nutaq® INNOVATION TODAY FOR TOMORROW®

RTDEx

Programmer's Reference Guide

May 2016

Revision history

Revision	Date	Comments
0.1	October 2012	First draft.
1.0	November 2012	First release
1.1	December 2012	Added flow-control
1.2	February 2013	Linguistic revision
1.3	March 2013	Added PCIe documentation
1.4	April 2013	Syntax and layout corrections
1.5	September 2013	Added paragraphs on how to use core signals
1.6	October 2013	Updated PCIE performance figures
1.7	January 2014	Added timing diagrams
1.8	November 2014	Added the “Augmenting the throughput of the PCIe RTDEx“ section Up to date for Software Tools Release 6.6
1.9	October 2015	New glossary Added section 7: Host Applications using the RTDEx Library Up to date for Software Tools Release 7.0
1.10	February 2016	Up to date for PicoSDR official release
1.11	May 2016	Updated to reflect new configurable channel width feature

Version 1.11

Trademarks

WARNING

Do not use Nutaq products in conjunction with life-monitoring or life-critical equipment. Failure to observe this warning relieves Nutaq of any and all responsibility.

FCC WARNING

This equipment is intended for use in a controlled environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of personal computers and peripherals pursuant to subpart J of part 15 of the FCC rules. These rules are designed to provide reasonable protection against radio frequency interference. Operating this equipment in other environments may cause interference with radio communications, in which case the user must, at his/her expense, take whatever measures are required to correct this interference.

This page was left intentionally blank.

Table of Contents

List of Figures and Tables

1 Introduction

Congratulations on the purchase of Nutaq's Perseus AMC.

This document contains all the information necessary to understand and use the RTDEx module of your product. It should be read carefully before using the card and stored in a handy location for future reference.

1.1 Conventions

In a procedure containing several steps, the operations that the user has to execute are numbered (1, 2, 3…). The diamond (♦) is used to indicate a procedure containing only one step, or secondary steps. Lowercase letters (a, b, c…) can also be used to indicate secondary steps in a complex procedure.

The abbreviation NC is used to indicate no connection.

Capitals are used to identify any term marked as is on an instrument, such as the names of connectors, buttons, indicator lights, etc. Capitals are also used to identify key names of the computer keyboard.

All terms used in software, such as the names of menus, commands, dialog boxes, text boxes, and options, are presented in bold font style.

The abbreviation N/A is used to indicate something that is not applicable or not available at the time of press.

Note:
The screen captures in this document are taken from the software version available at the time of press. For this reason, they may differ slightly from what appears on your screen, depending on the software version that you are using. Furthermore, the screen captures may differ from what appears on your screen if you use different appearance settings.

1.2 Glossary

This section presents a list of terms used throughout this document and their definition.

Term	Definition
Advanced Mezzanine Card (AMC)	AdvancedMC is targeted to requirements for the next generation of "carrier grade" communications equipment. This series of specifications are designed to work on any carrier card (primarily AdvancedTCA) but also to plug into a backplane directly as defined by MicroTCA specification.
Advanced Telecommunications Computing Architecture (or AdvancedTCA, ATCA)	AdvancedTCA is targeted primarily to requirements for "carrier grade" communications equipment, but has recently expanded its reach into more ruggedized applications geared toward the military/aerospace industries as well.http://en.wikipedia.org/wiki/Advanced_Telecommunications_Computing_Architecture - cite_note-3 This series of specifications incorporates the latest trends in high speed interconnect technologies, next-generation processors, and improved Reliability, Availability and Serviceability (RAS).
Application Programming Interface (API)	An application programming interface is the interface that a computer system, library, or application provides to allow requests for services to be made of it by other computer programs or to allow data to be exchanged between them.
Board Software Development Kit (BSDK)	The board software development kit gives users the possibility to quickly become fully functional developing C/C++ for the host computer and HDL code for the FPGA through an understanding of all Nutaq boards major interfaces.
Boards and Systems (BAS)	Refers to the division part of Nutaq which is responsible for the development and maintenance of the hardware and software products related to the different Perseus carriers and their different FMC daughter cards.
Carrier	Electronic board on which other boards are connected. In the FMC context, the FMC carrier is the board on which FMC connectors allow a connection between an FMC card and an FPGA. Nutaq has two FMC carriers, the Perseus601x (1 FMC site) and the Perseus611x (2 FMC sites).
Central Communication Engine (CCE)	The Central Communication engine (CCE) is an application that executes on a virtual processor called a MicroBlaze in the FPGA of the Perseus products. It handles all the behavior of the Perseus such as module initialization, clock management, as well as other tasks.
Chassis	Refers to the rigid framework onto which the CPU board, Nutaq development platforms, and other equipment are mounted. It also supports the shell-like case—the housing that protects all the vital internal equipment from dust, moisture, and tampering.
Command Line Interface (CLI)	The Command Line Interface (or CLI) is a basic client interface for Nutaq's FMC carriers. It runs on a host device. It consists of a shell where commands can be typed, interacting with the different computing elements connected to the system.
FPGA Mezzanine Card (FMC)	FPGA Mezzanine Card is an ANSI/VITA standard that defines I/O mezzanine modules with connection to an FPGA or other device with re-configurable I/O capability. It specifies a low profile connector and compact board size for compatibility with several industry standard slot card, blade, low profile motherboard, and mezzanine form factors.
HDL	Stands for hardware description language.
Host	A host is defined as the device that configures and controls a Nutaq board. The host may be a standard computer or an embedded CPU board in the same chassis system where the Nutaq board is installed. You can develop applications on the host for Nutaq boards through the use of an application programming interface (API) that comprises protocols and functions necessary to build software applications. These API are supplied with the Nutaq board.
MicroTCA (or μTCA)	The MicroTCA (μTCA) specification is a PICMG Standard which has been devised to provide the requirements for a platform for telecommunications equipment. It has been created for AMC cards.
Model-Based Design	Refers to all the Nutaq board-specific tools and software used for development with the boards in MATLAB and Simulink and the Nutaq model-based design kits.
Model-Based Development Kit (MBDK)	The model-based development kit gives users the possibility to create FPGA configuration files, or bitstreams, without the need to be fluent in VHDL. By combining Simulink from Matlab, System Generator from Xilinx and Nutaq's tools, someone can quickly create fully-functional FPGA bitstreams for the Perseus platforms.
NTP	Network Time Protocol. NTP is a protocol to synchronize the computer time over a network.

Term	Definition
Peer	A host peer is an associated host running RTDEx on either Linux or Windows. An FPGA peer is an associated FPGA device.
PicoDigitizer / PicoSDR Systems	Refers to Nutaq products composed of Perseus AMCs and digitizer or SDR FMCs in a table top format.
PPS	Pulse per second. Event to indicate the start of a new second.
Reception (Rx)	Any data received by the referent is a reception.
Reference Design	Blueprint of an FPGA system implemented on Nutaq boards. It is intended for others to copy and contains the essential elements of a working system (in other words, it is capable of data processing), but third parties may enhance or modify the design as necessary.
Transmission (Tx)	Any data transmitted by the referent is a transmission. Abbreviated TX.
µDigitizer / µSDR Systems	Any Nutaq system composed of a combination of µTCA or ATCA chassis, Perseus AMCs and digitizer or SDR FMCs.
VHDL	Stands for VHSIC hardware description language.

Table 1 Glossary

1.3 Technical Support

Nutaq is firmly committed to providing the highest level of customer service and product support. If you experience any difficulties using our products or if it fails to operate as described, first refer to the documentation accompanying the product. If you find yourself still in need of assistance, visit the technical support page in the Support section of our Web site at www.nutaq.com.

2 Product Description

Nutaq's RTDEx FPGA core allows a high speed transfer of data between a host PC and an FPGA device, or between FPGA devices. The RTDEx uses the Gigabit Ethernet-base channel 0 or PCI Express Gen1 4x. The PCI Express is only usable with the Fedora 17 Linux distribution on an embedded AMC processor.

2.1 Outstanding Features

Selection of transmission peer

The RTDEx allows data to be transmitted either between an FPGA device and a host PC or between two FPGA devices. In the case of a device to device transfer, the transfer can be monitored through the RTDEx core statistics.

The device associated to the targeted FPGA is named "peer". In the scope of this document, an associated host running RTDEx on either Linux or Windows will be called a "host peer" and an associated FPGA device will be called an "FPGA peer".

Full-duplex

The RTDEx allows data transfers in both to FPGA and from FPGA directions at the same time.

- ***To FPGA***: Data is received by the targeted FPGA. It is either transmitted by the host or by another FPGA device.
- ***From FPGA***: Data is transmitted by the targeted FPGA. It is either received by a host or another FPGA device.

Transfer modes

The RTDEx offers two transfer modes:

- ***Single transfer***: The host specifies a transfer size. The transfer is considered complete once the number of bytes received is equal to the specified transfer size. It is the transmitter responsibility to stop sending data once the transfer size has been reached. In the ***to FPGA*** direction, the host peer or FPGA peer will have to stop transmitting. In the ***from FPGA*** direction, the targeted FPGA will have to stop transmitting.
- ***Continuous transfer***: No transfer size is specified. The transfer continues until the transfer is stopped by the host peer or the FPGA peer.

Multi-channel

The RTDEx allows data transfers on 8 ***to FPGA*** channels and 8 ***from FPGA*** channels concurrently. The receiver and transmitter channels can be connected to different FPGA cores or peripherals and are operated independently on the host PC.

3 Ethernet RTDEx FPGA Core Description

The RTDEx core uses a FIFO-based interface on the user's side and a Xilinx AXI Ethernet interface on the Ethernet side. The user's FIFO depth is selectable independently for RX and TX directions.

The core is connected to the AXI-bus for register access. For clarity, the AXI parameters and ports were omitted from the descriptions.

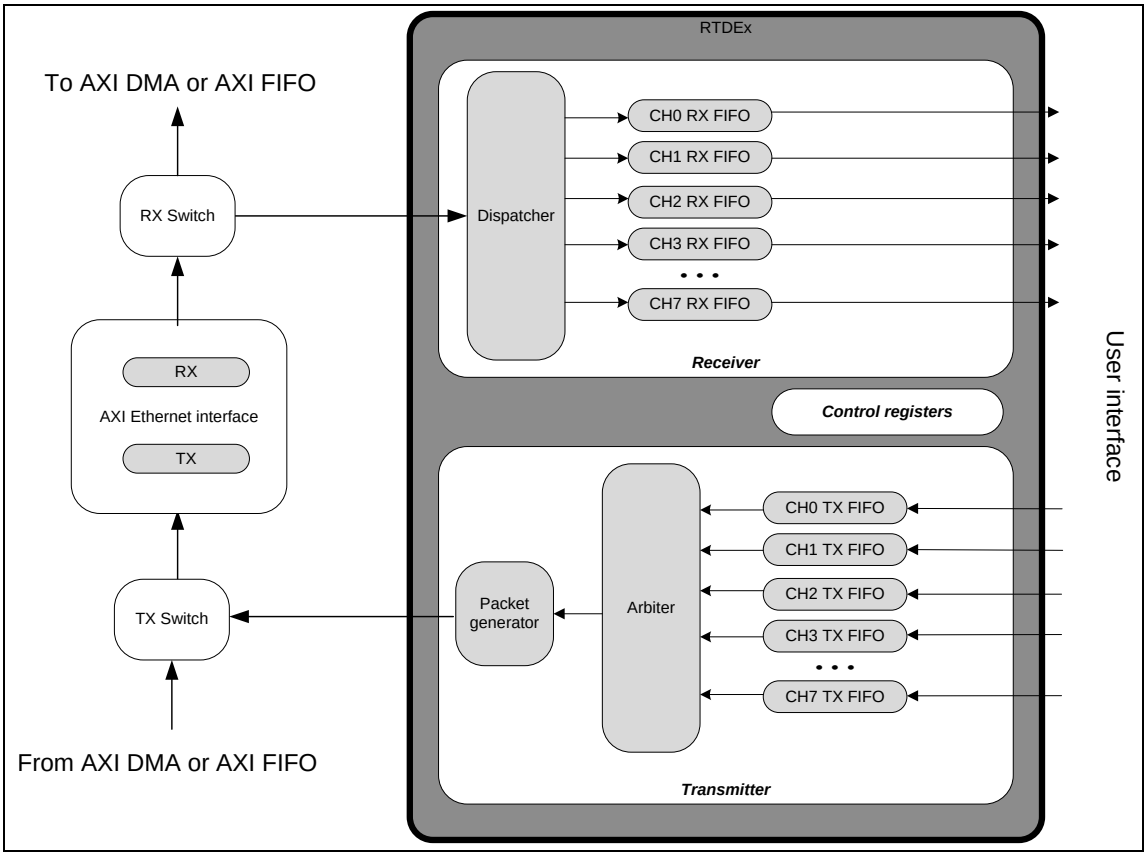

Figure 3-1 RTDEx Ethernet core architecture

3.1 Core Level Features

The RTDEx FPGA core consists of 8 channels with independent receivers and transmitters. All channels offer a FIFO interface to the user logic.

Reset: The core can be fully reset, emptying all FIFOs and resetting all registers to default values.

Receiver dispatcher: The receiver dispatcher parses the incoming RTDEx packets for the channel ID. The data is then sent to the channel RX FIFO.

- **Time to drop:** In the case the FPGA is not capable to receive data fast enough, the receiver FIFO can overflow and create an Ethernet jam. This jam will prevent the user to access the FPGA through the TCP/IP control link (EAPI and CCE). To prevent this, a mechanic with a programmable timeout has been implemented. Once a jam occurs, a counter will count for the programmed time value and drop frames after the timeout expires if the jam still exists. A dropped frames counter will be incremented by 1 for each dropped frame. An API function is dedicated to this feature.

Transmitter arbiter: In a round-robin pattern, the arbiter verifies if the channel TX FIFO data size has reached the channel specified data size. If the data is present, the arbiter forwards it to the packet generator.

- **TX burst size:** The user can select the maximum data size the arbiter handles for each channel in each round. The burst size is specified in number of packets. The following table illustrates the behavior of the arbiter depending on the set burst size. An API function is dedicated to this feature.

FIFO Data Size in Packets	Burst Size in Packets	Arbiter Action	Remaining Packets in FIFO
18	16	Copies 16 packets to the packet generator.	2
13	16	Copies 13 packets to the packet generator.	0

Table 2 TX burst size parameter

Packet generator: The packet generator adds the RTDEx Ethernet header to the data payload and transfers the fully formed packet to the AXI MAC interface.

- **Frame gap:** The packet generator can allow for a delay in increment of 10 ns between each frame sent out to the MAC. This feature allows the user to slow down the RTDEx interface and prevent data loss in the case the Ethernet flow control is not supported. An API function is dedicated to this feature.

Peer and FPGA MAC Address: The peer and FPGA MAC addresses must be specified for the RTDEx link to function. All channels use the same peer and FPGA MAC address.

Flow control capability: When enabled, it allows the receiver to send pause request to the sender when necessary to prevent loss of data. In the case of a host peer, its TX flow control should be enabled in order for this feature to work.

3.2 Receiver Channel Level Features

Channel FIFO reset

The RX channels FIFO can be reset individually. This resets only the user FIFO.

Direction

RTDEx channels must be instantiated with a direction. To indicate an RX channel, the direction must be set to ***to FPGA.***

Packet/Frame size

The packet or frame size refers to the Ethernet payload size of each packet. It can be different for each channel. Packets of up to 1472 bytes of payload are considered normal frames. Jumbo frames support up to 8960 bytes of payload.

Transfer mode

An RX channel can be used for single transfers or continuous transfers. When the channel is set to continuous transfer mode, the transfer size parameter is ignored.

Transfer size

The transfer size refers to the number of bytes exchanged before a transfer is considered complete. In the case the targeted FPGA is the RTDEx receiver, the transmitter (host peer or FPGA peer) is responsible to manage the transfer size and completion. The receiver does not count incoming packets nor verify transfer completion.

Transfer start

The transfer start refers to the moment the channel receiver is enabled and ready to receive. From this moment on, the received packets will be copied to the FIFO. Packets received before the receiver channel is enabled are dropped.

Transfer stop

A transfer can be stopped before completion by clearing the transfer start bit. If desired, the channel individual FIFO can be reset.

Statistics

The RTDEx core gives access to a number of channel level statistics. The statistics available for the receiver are:

- Channel frame count: number of packets received by the channel since the last transfer start.
- Channel frames lost: number of packets lost somewhere in the link.
- Channel frames dropped: number of packets dropped because the channel FIFO did not have enough space for a full incoming frame while the timeout counter expired.
- Bad frames: number of invalid frames received on a core level.

3.3 Transmitter Channel Level Features

Channel FIFO reset

FIFOs of TX channels can be reset individually. This only resets the user FIFO.

Direction

RTDEx channels must be instantiated with a direction. To indicate a TX channel, the direction must be set to ***from fpga.***

Packet/Frame size

The packet or frame size refers to the Ethernet payload size of each packet. It can be different for all channels. Packets of up to 1472 bytes of payload are considered normal frames. Jumbo frames support up to 8960 bytes of payload.

Transfer mode

A TX channel can be used for single transfers or continuous transfers. When the channel is set to continuous transfer mode, the transfer size parameter is ignored.

Transfer size

The transfer size refers to the number of bytes exchanged before a transfer is considered complete. In the case the targeted FPGA is the RTDEx transmitter, the RTDEx core is in charge of managing the transfer size and completion. The transmitter will count outgoing packets and verify transfer completion.

Transfer start

The transfer start refers to the moment the channel transmitter is enabled. From this moment on, the data in the FIFO will be sent to the arbiter for transmission. Data being written to the TX FIFO before the transmitter channel is enabled is dropped.

Transfer stop

A transfer can be stopped before completion by clearing the transfer start bit. If desired, the channel individual FIFO can be reset.

Statistics

The RTDEx core gives access to a number of channel level statistics. The statistics available for the transmitter are:

- Channel frame count: number of packets transmitted by the channel since the last transfer start.
- Channel FIFO count: number of data bytes actually in the channel TX FIFO.

3.4 RTDEx Parameters

Parameter	Type	Values Allowed	Default Value	Description
C_RTDEX_RX_NUMER_OF_CHANNELS	Integer	1, 2, 3, 4, 5, 6, 7, 8	1	Number of RTDEx RX channels
C_RX_CH_FIFO_DEPTH (1)	Integer	1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576	4096	RTDEx RX channel FIFO depth, in number of 4-byte samples
C_RX_WIDTH_CH# (2)	Integer	32, 64, 128, 256	32	Channel width, on the user logic interface side
C_RX_STATS_COUNTERS (3)	Integer	1 = Statistics counters enabled 0 = Statistics counters disabled	0	RTDEx RX statistics counters
C_ENABLE_FLOW_CTRL (4)	Integer	1 = RX can send pause requests 0 = RX never sends pause requests	0	RX request pause enable

Table 3 RTDEx RX generic parameters

Parameter	Type	Values Allowed	Default Value	Description
C_RTDEX_TX_NUMER_OF_CHANNELS	Integer	1, 2, 3, 4, 5, 6, 7, 8	1	Number of RTDEx TX channels
C_TX_CH_FIFO_DEPTH (1)	Integer	1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576	4096	RTDEx TX channel FIFO depth, in number of 4-byte samples
C_TX_WIDTH_CH# (2)	Integer	32, 64, 128, 256	32	Channel width, on the user logic interface side
C_TX_STATS_COUNTERS (3)	Integer	1 = Statistics counters enabled 0 = Statistics counters disabled	0	RTDEx TX statistics counters
C_SUPPORT_JUMBO_FRM (4)	Integer	1 = TX jumbo frames enabled 0 = TX jumbo frames disabled	1	RTDEx TX jumbo frames support

Table 4 RTDEx TX generic parameters

NOTES:

(1) An element of the FIFO is 4 bytes wide. A FIFO depth of 4096 (default value) will result in a FIFO of size 4096 x 4 bytes = 16 384 bytes FIFO.

(2) Where # is the channel number (0 – 7).

(3) When not enabled the counters will read 0xDEADBEEF.

(4) When enabling this feature, make sure to enable the Ethernet TX flow control on the host PC. This way the system can make use of the Ethernet flow control feature.

(5) For jumbo frames support, see Jumbo frames support.

3.4.1 Jumbo Frame Support

The RTDEx RX core supports jumbo frames by default without any configuration from the user. The core will accept any valid frame size whether it is a normal or a jumbo frame size.

To transmit packets of jumbo frame size (1472 < frame size <= 8960), the jumbo frame option of the RTDEx TX core must be enabled. Once enabled, the TX core sends a jumbo frame if the requested (configured) frame size is greater than 1472 bytes of payload size. If the requested frame size is less or equal to 1472 bytes of payload, the TX core sends a frame of a regular size.

When the jumbo frame support option is disabled, the RTDEx TX core will always send regular size frames. If the requested frame size is greater than 1472 bytes of payload, the TX core will still send frames as requested but with a regular header instead of a jumbo frame header.

3.4.2 RX Flow Control Support

The RTDEx RX core supports the use of Ethernet flow control. When enabled, the FIFO of each channel can be programmed with an assert threshold and a negate threshold value. These values are used to determine when the flow control signals will be sent. If the amount of data in the FIFO becomes higher than the assert threshold value, the RX core triggers the sending of a pause frame with the maximum allowable value, 0xFFFF. Once the amount of data in the FIFO becomes lower than the negate threshold, the RX core triggers the sending of a pause frame with the value 0, cancelling the previous pause request and resuming communication. Registers exist to set a value for these two thresholds. Please refer to the “Register Definition“ section for details.

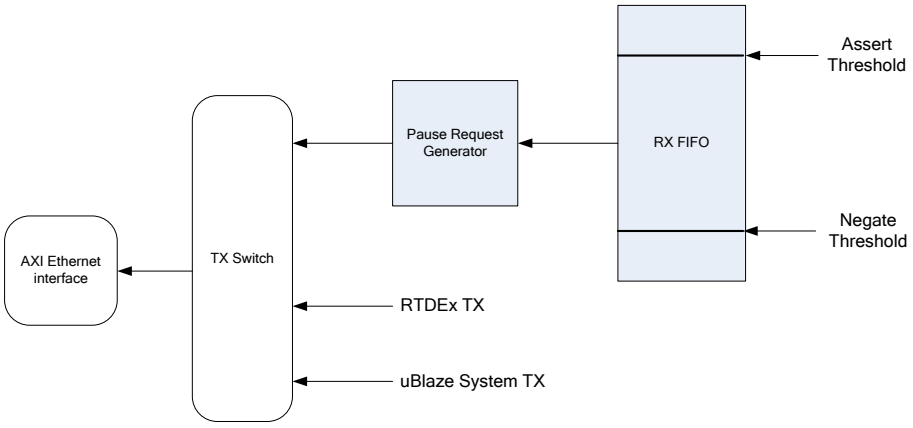

Figure 3-2 RX flow mechanism diagram

3.4.3 TX Flow Control Support

The RTDEx TX core supports Ethernet flow control at all time because the TX flow control is done at the Xilinx Ethernet core level and the option is permanently enabled in the Linux drivers. That is, if the Xilinx Ethernet core receives a pause request from an Ethernet link partner, it will react on it.

3.5 Core Ports

Port	Range	Direction	Description
i_TxUserClk_p	–	In	User clock of the RTDEx TX interface. All the FIFOs are in the same clock domain.
o_TxReadyCh#_p	–	Out	Indicates whether the channel TX FIFO is ready for new data write.
i_TxWeCh#_p	–	In	Used to send a write signal to the channel TX FIFOs.
iv_TxDataCh#_p	[N:0]	Out	The channel TX FIFO data input ports, where N is the channel width specified using generic parameter C_TX_WIDTH_CH#

Table 5 TX core user ports

Port	Range	Direction	Description
i_RxUserClk_p	–	In	User clock of the RTDEx RX interface. All the FIFOs are in the same clock domain.
o_RxReadyCh#_p	–	Out	Indicates whether the channel RX FIFO has data available for reading.
i_RxReCh#_p	–	In	Used to send a read signal to the channel RX FIFOs.
ov_RxDataCh#_p	[N:0]	Out	The channel RX FIFO data input ports, where N is the channel width specified using generic parameter C_RX_WIDTH_CH#
o_RxDataValidCh#_p	[7:0]	Out	Indicates that the data on ov32_RxDataCh#_p is valid.

Table 6 RX core user ports

3.6 Core Registers

Register name	Address Offset	Direction	Description
CORE_ID_VERS	0×000	R	RTDEx core ID and version identifier
CORE_RST	0x004	R/W	RTDEx core reset
RX_TX_FIFO_RST	0x008	R/W	Receiver and transmitter FIFO reset
PEERMACADDRESSLOWWORD	0×00C	R/W	Specifies the peer MAC address.
PEERMACADDRESSHIGHWORD	0×010	R/W	
FPGAMACADDRESSLOWWORD	0×014	R/W	Specifies the FPGA MAC address.
FPGAMACADDRESSHIGHWORD	0×018	R/W	
RX_TX_MODE	0x01C	R/W	RX and TX mode configuration
RX_TIMEOUT_2DROPFRM	0x020	R/W	Time before dropping a frame in case the RX FIFO is not ready for the incoming frame.
RX_CONFIG_INFO	0x024	R/W	RX configuration status
RX_STARTNEWTRANSFER	0x028	R/W	Rx transfer start
RX_FRAMECNT_CH#	0x02C to 0x048	R/W	Received frame count since last start
RX_ERR_STATUS_REG	0x04C		RX error status register
RX_BADFRAMECNT	0×050	R	Indicates the number of bad captured frames (used for statistical purposes).
RX_FRAMELOSTCNTCH#	0×054 to 0×070	R	
RX_DROPEDFRMCNT_CH#	0x074 to 0x090	R	Indicates the number of dropped frames by channel (used for statistical purposes).
RX_FIFO_UNDERRUN	0x94	R	Indicates if there was a RX FIFO underrun.
RX_FOR_FUTURE_USE	0x098		Reserved

Register name	Address Offset	Direction	Description
TX_FIFO_OVERRUN	0x9C	R	Indicates if there was a TX FIFO overrun.
TX_FRAMESIZE_CH#	0×0A0 to 0×0BC	R/W	Indicates the frame size in bytes for each channel when performing TX transfers.
TX_TRANSFERSIZE_CH#	0×0C0 to 0×0DC	R/W	Indicates the transfer size in packets for each channel when performing TX transfers.
TX_FRAME_GAP	0x0E0	R/W	Specifies the inter-frame gap time on the transmitter.
TX_FRAME_BURST	0x0E4	R/W	Number of packets for each channel at each round of the arbiter
TX_CONFIG_INFO	0x0E8	R/W	RX configuration status
TX_STARTNEWTRANSFER	0×0EC	R/W	When a non-zero value is written to the register bit 7 to bit 0, an FPGA2HOST transfer is performed on the channel corresponding to the bit value.
TX_FRAMECNT_CH#	0×0F0 to 0x10C	R/W	Number of packets transmitted for each channel since the last start.
TX_NBDATAINFIFO_CH#	0×110 to 0x12C	R/W	Number of data bytes that are actually in the channel TX FIFO.
RX_THRESHOLD_H	0x130	R/W	RX FIFO programmable full assert.
RX_THRESHOLD_L	0x134	R/W	RX FIFO programmable full negate.

Table 7 Ethernet register map

3.6.1 Register Definitions

Offset 0x000 — CORE_ID_VERS

This register indicates the RTDEx core ID and version identifier.

31 to 16	15 to 0
Core ID	**Version ID**
R	R/W
0xDE88	0x0200

Table 8 CORE_ID_VERS register

Offset 0x004 — CORE_RST

This register allows the user to apply a core level reset.

31 to 1	0
Reserved	**CoreReset**
R	R/W
00000000000000000000000000000000	0

Table 9 CORE_RST register

Bit	Description	Configuration
CoreReset	This bit allows the user to reset the RTDEx core.	1 for reset, the bit is automatically cleared.

Table 10 CORE_RST register field description

Offset 0x008— RX_TX_FIFO_RST

This register allows the user to reset receiver and transmitter FIFOs.

31 to 24	23	22	21	20	19	18	17	16
Reserved	Channel 7 TX FIFO Reset	Channel 6 TX FIFO Reset	Channel 5 TX FIFO Reset	Channel 4 TX FIFO Reset	Channel 3 TX FIFO Reset	Channel 2 TX FIFO Reset	Channel 1 TX FIFO Reset	Channel 0 TX FIFO Reset
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0	0

15 to 8	7	6	5	4	3	2	1	0
Reserved	Channel 7 RX FIFO Reset	Channel 6 RX FIFO Reset	Channel 5 RX FIFO Reset	Channel 4 RX FIFO Reset	Channel 3 RX FIFO Reset	Channel 2 RX FIFO Reset	Channel 1 RX FIFO Reset	Channel 0 RX FIFO Reset
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0	0

Table 11 RX_TX_FIFO_RST register

Bit	Description	Configuration
Channel X TX FIFO Reset	This bit allows the user to reset the RTDEx TX channel FIFO.	1 = Reset FIFO 0 = Release FIFO reset
Channel X RX FIFO Reset	This bit allows the user to reset the RTDEx RX channel FIFO.	1 = Reset FIFO 0 = Release FIFO reset

Table 12 RX_TX_FIFO_RST register fields description

Offset 0x00C— PEERMACADDRESSLOWWORD

This register specifies the peer MAC address.

31 to 0
MACAddress Low Bytes
R/W
0

Table 13 PEERMACADDRESSLOWWORD register

Offset 0x010— PEERMACADDRESSHIGHWORD

This register specifies the peer MAC address.

31 to 16	15 to 0
Reserved	MAC Address High Bytes
R	R/W
0	0

Table 14 PEERMACADDRESSHIGHWORD register

Offset 0x014— FPGAMACADDRESSLOWWORD

This register specifies the FPGA MAC address.

31 to 0
MAC Address Low Bytes
R/W
0

Table 15 FPGAMACADDRESSLOWWORD register

Offset 0x018— FPGAMACADDRESSHIGHWORD

This register specifies the FPGA MAC address.

31 to 16	15 to 0
Reserved	**MAC Address High Bytes**
R	R/W
0	0

Table 16 FPGAMACADDRESSHIGHWORD register

Offset 0x01C— RTDEX_MODE

This register configures the RTDEx transfer mode.

31 to 2	1	0
Reserved	**RX Mode**	**TX Mode**
R	R/W	R/W
0	0	0

Table 17 RTDEX_MODE register

Bit	Description	Configuration
TX mode	This bit allows the user to set the RTDEx TX transfer mode.	1 = Continuous transfer mode 0 = Predetermined size transfer mode
RX mode	This bit allows the user to set the RTDEx RX transfer mode.	This bit is ignored and the RX core always accepts packets either from a continuous or predetermined size transfer mode.

Table 18 RTDEX_MODE register fields description

Offset 0x020— RX_TIMEOUT_2DROPFRM

This register specifies the timeout value before a received frame is dropped when the channel FIFO is full. Special care should be taken when choosing the value of this register. Too big a value can result in slow and even momentary loss of TCP/IP link. Too small a value increases the risk of dropping frames too quickly. This value must be adjusted according to the transfer rate.

31 to 0
Timeout Value in a 10-ns Count
R/W
0x1FE (this corresponds to 5 µs)

Table 19 RX_TIMEOUT_2DROPFRM register

Offset 0x024— RX_CONFIG_INFO

This register gives information about the core configuration parameters at the compile time.

31 to 11	10	9 To 4	3 to 0
Reserved	**Statistics Counters**	**Channel FIFO Depth**	**Number of Channels**
R	R/W	R/W	R/W
0	0	4	1

Table 20 RX_CONFIG_INFO register

Bit	Description	Configuration
Number of channels	Shows the core number of channels configuration.	Number of channels
Channel FIFO depth	Shows the core individual channel FIFO depth configuration. Note: these are 32-bit width FIFO (4 bytes).	1 = 1024 FIFO 2 = 2048 FIFO 4 = 4096 FIFO 8 = 8192 FIFO 16 = 16384 FIFO 32 = 32768 FIFO
Statistics counters	Shows whether the statistics counters are enabled.	1 = Counters enabled 0 = Counters disabled

Table 21 RX_CONFIG_INFO register fields description

Offset 0x028— RX_STARTNEWTRANSFER

This register allows the user to start an RX transfer.

31 to 8	7	6	5	4	3	2	1	0
Reserved	**Channel 7 RX Start**	**Channel 6 RX Start**	**Channel 5 RX Start**	**Channel 4 RX Start**	**Channel 3 RX Start**	**Channel 2 RX Start**	**Channel 1 RX Start**	**Channel 0 RX Start**
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0	0

Table 22 RX_STARTNEWTRANSFER register

Bit	Description	Configuration
Channel x RX start	This bit allows the user to start a new transfer in the FPGA direction	1 = Start channel 0 = Disable channel

Table 23 RX_STARTNEWTRANSFER register fields description

Offset 0x02C to 0x48— RX_FRAMECNT_CH#

This register specifies the received frames on channel X.

31 to 0
Channel X Received Frame Count **Note: These counters are only cleared upon core reset.**
R
0

Table 24 RX_FRAMECNT_CH# register

Offset 0x04C— RX_ERR_STATUS_REG

This register holds the current errors status.

31 to 3	2 to 0
Reserved	**RX Error Status** **Note: These bits are only cleared upon core reset.**
R	R
0	0

Table 25 RX_ERR_STATUS_REG register

Bits	Description	Configuration
Bit 0	Shows whether one or more bad frames were received.	1 = Bad frame error occurred 0 = No Bad frame error
Bit 1	Shows whether one or more frames were lost.	1 = Frames lost 0 = No frame lost
Bit 2	Shows whether one or more frames were dropped.	1 = Frames dropped 0 = No frame dropped.

Table 26 RX_ERR_STATUS_REG register fields description

Offset 0x050— RX_BADFRAMECNT

This register specifies the number of bad frames received. These are malformed frames that were received at the core level. It can also be an RTDEx frame with a destination channel number that is not started yet (not enabled).

31 to 0
Received Bad Frame Count **Note: These counters are only cleared upon core reset.**
R
0

Table 27 RX_BADFRAMECNT register

Offset 0x054 to 0x70— RX_FRAMELOSTCNT_CH#

This registers shows the Lost Frame count for channel X.

31 to 0
Channel X Lost Frames Count **Note: These counters are only cleared upon core reset.**
R
0

Table 28 RX_FRAMELOSTCNT_CH# register

Offset 0x074 to 0x90— RX_DROPPEDFRAMECNT_CH#

This register specifies the dropped frames due to a FIFO timeout on channel X. When a FIFO does not have enough space to accommodate a complete incoming frame a timeout counter is started with the configured value (see offset 0x020 register). Once this timeout counter is expired and if the given channel FIFO still does not have the required space, the frame is dropped and the Dropped Frames counter is

incremented. This is to prevent the link from stalling and resulting in a TCP/IP connection loss with the FPGA.

31 to 0
Channel X Dropped Frames Count **Note: These counters are only cleared upon core reset.**
R
0

Table 29 RX_DROPPEDFRAMECNT_CH# register

Offset 0x094— RX_FIFO_UNDERRUN

This register shows the channel FIFO underrun latched status.

31 to 8	7	6	5	4	3	2	1	0
Reserved	**Channel 7 RX Underrun Status**	**Channel 6 RX Underrun Status**	**Channel 5 RX Underrun Status**	**Channel 4 RX Underrun Status**	**Channel 3 RX Underrun Status**	**Channel 2 RX Underrun Status**	**Channel 1 RX Underrun Status**	**Channel 0 RX Underrun Status**
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0	0

Table 30 RX_FIFO_UNDERRUN register

Bit	Description	Configuration
Channel x RX underrun status	This bit shows if there was an RX FIFO underrun. It self clears upon read access.	1 = FIFO underrun occurred 0 = No FIFO underrun

Table 31 RX_FIFO_UNDERRUN register field description

Offset 0x0A0 to 0x0BC— TX_FRAMESIZE_CH#

This register specifies the frames payload size of the RTDEx frames sent by the transmitter for channel X.

31 to 15	14 to 0
Reserved	**Channel X TX Frame Size (Max. Size: 8960 Bytes)**
R	R/W
0	0

Table 32 TX_FRAMESIZE_CH# register

Offset 0x09C— TX_FIFO_OVERRUN

This register shows the channel FIFO overrun latched status.

31 to 8	7	6	5	4	3	2	1	0
Reserved	**Channel 7 TX Overrun Status**	**Channel 6 TX Overrun Status**	**Channel 5 TX Overrun Status**	**Channel 4 TX Overrun Status**	**Channel 3 TX Overrun Status**	**Channel 2 TX Overrun Status**	**Channel 1 TX Overrun Status**	**Channel 0 TX Overrun Status**
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0	0

Table 33 RX_FIFO_OVERRUN register

Bit	Description	Configuration
Channel x TX overrun status	This bit shows if there was a TX FIFO overrun. It self clears upon read access.	1 = FIFO overrun occurred 0 = No FIFO overrun

Table 34 RX_FIFO_OVERRUN register fields description

Offset 0x0C0 to 0x0DC— TX_TRANSFERSIZE_CH#

These registers specify the transfer size in packets sent by the transmitter for channel X.

31 to 0
Channel X Transfer Size in Frames
R
0

Table 35 TX_TRANSFERSIZE_CH# register

Offset 0x0E0— TX_FRAMEGAP

This register specifies the inter-frame gap time on the transmitter in increments of 10 ns. By default this register is set to zero and there is no frame gap. The frame gap value must be adjusted according to the application transfer rate. Too big a value, the transfer speed is reduced. Too small a value can lead to overflow if the link partner is incapable of keeping up with the transmitter rate.

31 to 0
Transmitter Frame Gap in Increments of 10 ns
R
0

Table 36 TX_FRAMEGAP register

Offset 0x0E4— TX_FRAMEBURST

This register specifies the number of packets sent for each channel on each round of the transmitter arbiter.

31 to 8	7 to 0
Reserved	**Transmitter Frame Burst Size in Packets**
R	R/W
0	0

Table 37 TX_FRAMEBURST register

Offset 0x0E8— TX_CONFIG_INFO

Refer to the “offset 0x24- RX_CONFIG_INFO” section.

Offset 0x0EC— TX_STARTNEWTRANSFER

This register allows the user to start a TX transfer.

31 to 8	7	6	5	4	3	2	1	0
Reserved	Channel 7 TX Start	Channel 6 TX Start	Channel 5 TX Start	Channel 4 TX Start	Channel 3 TX Start	Channel 2 TX Start	Channel 1 TX Start	Channel 0 TX Start
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0	0

Table 38 TX_STARTNEWTRANSFER register

Bit	Description	Configuration
Channel X TX start	This bit allows the user to start a new outgoing transfer from the FPGA	1 = Start channel 0 = Disable channel

Table 39 TX_STARTNEWTRANSFER register field description

Offset 0x0F0 to 0x10C— TX_FRAMECNT_CH#

These registers show the transmitted frames count on channel X.

31 to 0
Channel X Transmitted Frame Count
R
0

Table 40 TX_FRAMECNT_CH# register

Offset 0x110 to 0x12C— TX_NBDATAINFIFO_CH#

These registers show the number of data bytes present in the FIFO for channel X.

31 to 0
Channel X Number of Bytes in the TX FIFO
R
0

Table 41 TX_NBDATAINFIFO_CH# register

Offset 0x130— RX_THRESHOLD_H

This register allows the user to set the RX FIFO programmable full assert threshold upon which the RX core will send a pause request to the sender.

31 to 0
RX FIFO Programmable Full Assert Threshold to Trigger a Pause Frame Request
R/W
0

Table 42 RX_THRESHOLD_H register

Offset 0x130— RX_THRESHOLD_L

This register allows the user to set the RX FIFO programmable full negate threshold upon which the RX core will send a resume from pause request to the sender.

31 to 0
RX FIFO Programmable Full Negate Threshold to Trigger a Resume from Pause Frame Request
R/W
0

Table 43 RX_THRESHOLD_L register

4 PCIe RTDEx FPGA Core Description

The PCIe RTDEx core uses a FIFO-based interface on the user's side. The Xilinx AXI Data Mover and AXI CDMA cores are used to exchange data between the FIFO-based interface and the PCIe AXI bridge. The user's FIFO depth is selectable independently for RX and TX directions.

The PCIe RTDEx core is connected to the uBlaze AXI-bus for register access. For clarity, the AXI parameters and ports were omitted from the descriptions.

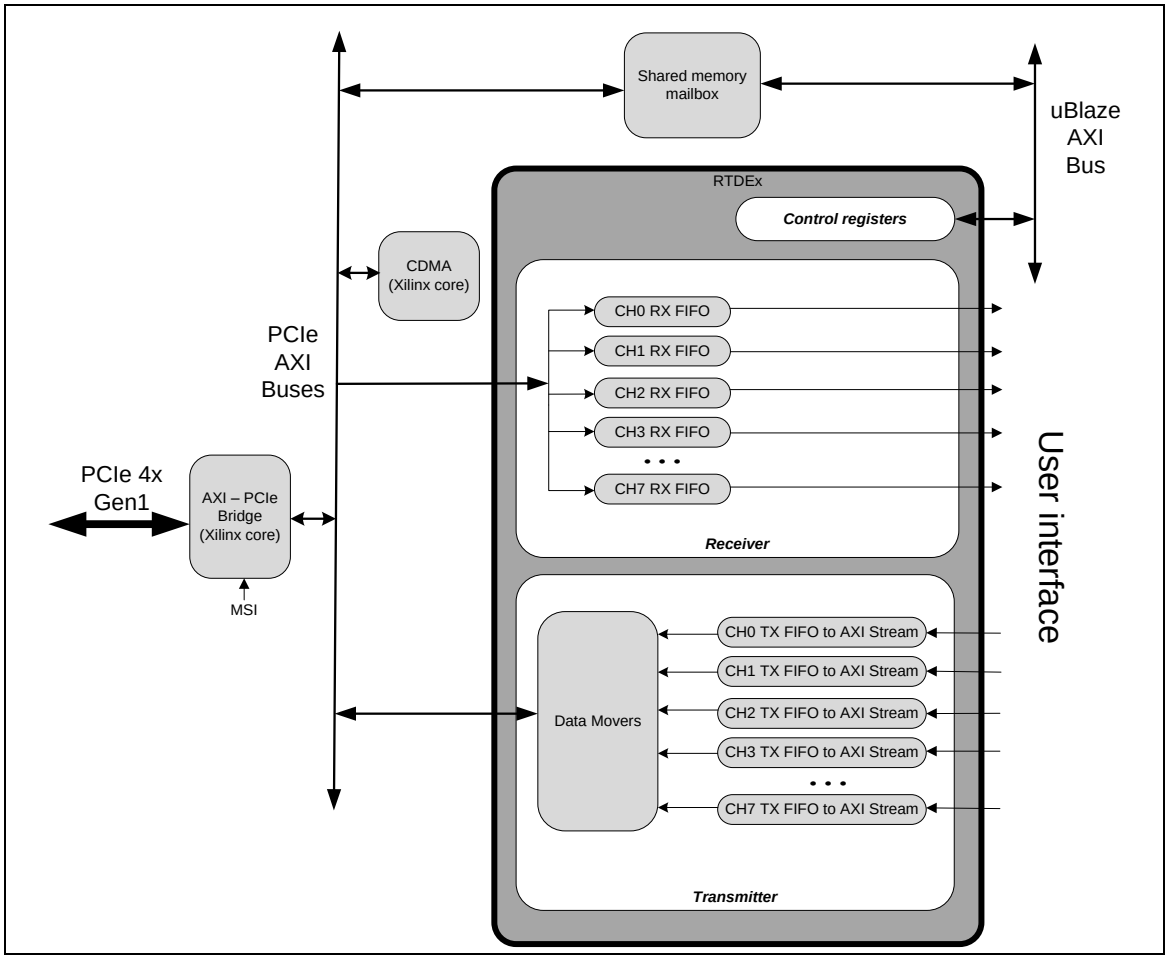

Figure 4-1 RTDEx PCIe core architecture

4.1 Core Level Features

The PCIe RTDEx FPGA core consists of 8 channels with independent receivers and transmitters. Every channel offers a FIFO interface to the user logic.

Transmitter arbiter

In a round-robin pattern, the arbiter provided with the AXI bus alternatively grants the AXI bus between the Data Movers.

TX Flow control

In transmission (to a host peer or to a FPGA peer), the AXI bus itself limits the bus access according to the PCIe bandwidth.

RX Flow control

In reception, the AXI bus access might be blocked if no space is available in the accessed FIFO. The access will then progress as the FIFO is emptying.

CDMA

A CDMA core is used to transfer data from a peer host memory to the CH RX FIFOs. It can be removed if such transfers are not required.

PCIe MSI

The PCIe MSI signal is used to interrupt the peer host through PCIe. The CDMA and the Data Movers use it to inform the host from transfer status changes. The uBlaze is also connected to this MSI signal. It is used to implement a mailbox between the host and the uBlaze through the shared memory mailbox.

4.2 Receiver Channel Level Features

Channel FIFO reset

The RX channels FIFO can be reset individually. This resets only the user FIFO.

Transfer size

The transfer size refers to the number of bytes exchanged before a transfer is considered complete. In the case the targeted FPGA is the RTDEx receiver, the transmitter (host peer or FPGA peer) is responsible to manage the transfer size and completion. The receiver does not count incoming packets nor verify transfer completion. The transfer size must be a multiple of 8 bytes.

Transfer start

The transfer start refers to the moment the channel receiver is enabled and ready to receive. From this moment on, the received data will be written to the FIFO. Data received before the receiver channel is enabled are dropped.

Transfer stop

A transfer can be stopped before completion by clearing the transfer start bit. If desired, the channel individual FIFO can be reset.

4.3 Transmitter Channel Level Features

Channel FIFO reset

FIFOs of TX channels can be reset individually. This only resets the user FIFO.

Transfer mode

A TX channel can be used for single transfers or continuous transfers. When the channel is set to continuous transfer mode, the transfer size parameter is ignored.

Transfer size

The transfer size refers to the number of bytes exchanged before a transfer is considered complete. In the case the targeted FPGA is the RTDEx transmitter, the RTDEx core is in charge of managing the transfer size and completion. The transmitter (the Data Mover) will count outgoing data and verify transfer completion.

Transfer start

The transfer start refers to the moment the channel transmitter is enabled. From this moment on, the data in the FIFO will be sent to the Data mover for transmission. Data being written to the TX FIFO before the transmitter channel is enabled is dropped.

Transfer stop

A transfer can be stopped before completion by clearing the transfer start bit. If desired, the channel individual FIFO can be reset.

4.4 RTDEx Parameters

The following tables list the user parameters for the PCIe RTDEx core. All other parameters should not be modified without care.

Parameter	Type	Values Allowed	Default Value	Description
C_RTDEX_RX_NUMER_OF_CHANNELS	Integer	0, 1, 2, 3, 4, 5, 6, 7, 8	1	Number of RTDEx RX channels
C_RX_CH_FIFO_DEPTH (1)	Integer	1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576	4096	RTDEx RX channel FIFO depth, in number of 4-byte samples
C_RX_WIDTH_CH# (2)	Integer	32, 64, 128, 256	32	Channel width, on the user logic interface side
C_CDMA_PRESENT (3)	Integer	Checkbox : 0 (uncheck), 1 (check)	1	CDMA present

Table 44 RTDEx RX generic parameters

Parameter	Type	Values Allowed	Default Value	Description
C_RTDEX_TX_NUMER_OF_CHANNELS	Integer	0, 1, 2, 3, 4, 5, 6, 7, 8	1	Number of RTDEx TX channels
C_TX_CH_FIFO_DEPTH (1)	Integer	1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576	4096	RTDEx TX channel FIFO depth, in number of 4-byte samples
C_TX_WIDTH_CH# (2)	Integer	32, 64, 128, 256	32	Channel width, on the user logic interface side

Table 45 RTDEx TX generic parameters

Notes:

(1) An element of the FIFO is 4 bytes wide. A FIFO depth of 4096 (default value) will result in a FIFO of size 4096 x 4 bytes = 16 384 bytes FIFO.

(2) Where # is the channel number (0 – 7).

(3) This should be checked unless the CDMA has been removed from the platform.

4.5 Core Ports

Port	Range	Direction	Description
i_TxUserClk_p	–	In	User clock of the RTDEx TX interface. All the FIFOs are in the same clock domain.
o_TxReadyCh#_p	–	Out	Indicates whether the channel TX FIFO is ready for new data write.
i_TxWeCh#_p	–	In	Used to send a write signal to the channel TX FIFOs.
iv_TxDataCh#_p	[N:0]	Out	The channel TX FIFO data input ports, where N is the channel width specified using generic parameter C_TX_WIDTH_CH#

Table 46 TX core user ports

Port	Range	Direction	Description
i_RxUserClk_p	–	In	User clock of the RTDEx RX interface. All the FIFOs are in the same clock domain.
o_RxReadyCh#_p	–	Out	Indicates whether the channel RX FIFO has data available for reading.
i_RxReCh#_p	–	In	Used to send a read signal to the channel RX FIFOs.
ov_RxDataCh#_p	[N:0]	Out	The channel RX FIFO data input ports, where N is the channel width specified using generic parameter C_RX_WIDTH_CH#
o_RxDataValidCh#_p	[7:0]	Out	Indicates that the data on ov32_RxDataCh#_p is valid.

Table 47 RX core user ports

4.6 Core Registers

Register name	Address Offset	Direction	Description
CORE_ID_VERS	0×000	R	RTDEx core ID and version identifier
CORE_CTRL_STAT	0x004	R/W	RTDEx core control statistics
RX_TX _RST	0x008	R/W	Receiver and transmitter FIFO reset
RX_CONFIG_INFO	0×00C	R	
RX_FIFO_STATUS	0×010	R	
RX_TX_FIFO_ENABLE	0×014	R/W	
RX_WORDCNT_CH#	0×018 to 0×034	R	
TX_CONFIG_INFO	0x040	R	
TX_FIFO_STATUS	0x044	R	
TX_TRANSFERCTRL	0x048	R/W	
TX_IRQCTRL	0x04C	R/W	
TX_DATAMOVERCTRL	0x050	R/W	
TX_TRANSFERCNT_CH#	0x54 + # x 0x20	R/W	
TX_CURRENTTRANSFERCNT_CH#	0x58 + # x 0x20	R	
TX_DATAMOVERCTRL_CH#	0x5C + # x 0x20	R/W	
TX_DATAMOVERADDR_CH#	0x60 + # x 0x20	R/W	

Register name	Address Offset	Direction	Description
TX_DATAMOVERTAG_CH#	0x64 + # x 0x20	R/W	
TX_DATAMOVERSTATUS_CH#	0x68 + # x 0x20	R	

Table 48 PCIe register map

4.6.1 Register Definitions

Offset 0x000 — CORE_ID_VERS

This register indicates the RTDEx PCIe core ID and version identifier.

31 to 16	15 to 0
Core ID	**Core Version**
R	R/W
0xC1E8	0x0000

Table 49 CORE_ID_VERS register

Offset 0x004 — CORE_CTRL_STAT

This register allows the user to apply a core level reset.

31 to 3	2	1	0
Reserved	**cdmaPresent**	**mbIrq**	**CoreResetPulse**
R	R	R/W	R/W
0x0000	0	0	0

Table 50 CORE_CTRL_STAT register

Bit	Description	Configuration
cdmaPresent	This bit allows the user to know when a CDMA for PCIe is present	1 = CDMA is present
mbIrq	Generate a MicroBlaze. IRQ (connected to the interrupt controller)	
CoreResetPulse	This bit allows the user to reset the RTDEx core.	1 for reset, a reset pulse is generated and the bit is automatically cleared.

Table 51 CORE_CTRL_STAT register field description

Offset 0x008— RX_TX_RST

31 to 24	23:16	15:8	7:0
Reserved	**TxFifoReset**	**RxWordCntReset**	**RxFifoReset**
R	R/W	R/W	R/W
0x00	0x00	0x00	0x00

Table 52 RX_TX_RST register

Bit	Description	Configuration
TxFifoReset	Reset bits for the TX FIFOs	Write 1 to reset TX FIFO for channel x. Bit 16 = channel 0 …

Bit	Description	Configuration
		Bit 23 = channel 7
RxWordCntReset	Reset bits for the RX word counters	Write 1 to reset RX word counter for channel x. Bit 8 = channel 0 … Bit 15 = channel 7
RxFifoReset	Reset bits for the RX FIFOs	Write 1 to reset RX FIFO for channel x. Bit 0 = channel 0 … Bit 7 = channel 7

Table 53 RX_TX_RESET register field description

Offset 0x00C — RX_CONFIG_INFO

31 to 0
Config info
R
0x0000

Table 54 RX_CONFIG_INFO register

Offset 0x010 — RX_FIFO_STATUS

31 to 16	15:8	7:0
Reserved	**RxFifoOverflow**	**RxFifoUnderflow**
R	R	R
0x0000	0x00	0x00

Table 55 RX_FIFO_STATUS register

Bit	Description	Configuration
RxFifoOverflow	RX FIFO overflow bit status	Overflow status for channel x Bit 8 = channel 0 … Bit 15 = channel 7
RxFifoUnderflow	RX FIFO underflow bit status	Underflow status for channel x Bit 0 = channel 0 … Bit 7 = channel 7

Table 56 RX_FIFO_STATUS register field description

Offset 0x010 — RX_TX_FIFO_ENABLE

31 to 24	23:16	15:8	7:0
Reserved	**TxFifoWrEn**	**RxFifoRdEn**	**RxFifoWrEn**
R	R/W	R/W	R/W
0x00	0x00	0x00	0x00

Table 57 RX_TX_FIFO_ENABLE register

Bit	Description	Configuration
TxFifoWrEn	Enable writing into the TX FIFOs	Enable writes to channel x TX FIFO Bit 16 = channel 0 … Bit 23 = channel 7
RxFifoRdEn	Enable reading from the RX FIFOs	Enable reads from channel x RX FIFO Bit 8 = channel 0 … Bit 15 = channel 7
RxFifoWrEn	Enable writing into the RX FIFOs	Enable writes to channel x RX FIFO Bit 0 = channel 0 … Bit 7 = channel 7

Table 58 RX_TX_FIFO_ENABLE register field description

Offset 0x018 to 0x34 — RX_WORDCNT_CH#

31 to 0
RX word count
R
0x00000000

Table 59 RX_CONFIG_INFO register

Offset 0x040 — TX_CONFIG_INFO

31 to 0
Config info
R
0x00000000

Table 60 TX_CONFIG_INFO register

Offset 0x044 — TX_FIFO_STATUS

31 to 16	15:8	7:0
Reserved	**TxFifoOverflow**	**TxFifoUnderflow**
R	R	R
0x0000	0x00	0x00

Table 61 RX_FIFO_STATUS register

Bit	Description	Configuration
TxFifoOverflow	TX FIFO overflow bit status	Overflow status for channel x Bit 8 = channel 0 … Bit 15 = channel 7
TxFifoUnderflow	TX FIFO underflow bit status	Underflow status for channel x Bit 0 = channel 0 … Bit 7 = channel 7

Table 62 RX_FIFO_STATUS register field description

Offset 0x048 — TX_TRANSFERCTRL

31 to 24	23:16	15:8	7:0
Reserved	**TxTransferDone**	**TxStreamingTransfer**	**TxStartNewTransfer**
R	R	R/W	R/W
0x00	0x00	0x00	0x00

Table 63 TX_TRANSFERCTRL register

Bit	Description	Configuration
TxTransferDone	Transfer completion status for each channel	Indicates if the last transfer is complete on channel x Bit 16 = channel 0 … Bit 23 = channel 7
TxStreamingTransfer	Enables infinite transfer size for each channel	Enable infonote transfer size for channel x Bit 8 = channel 0 … Bit 15 = channel 7
TxStartNewTransfer	Transfer start bit for each channel	Start transfer on channel x Bit 0 = channel 0 … Bit 7 = channel 7

Table 64 TX_TRANSFERCTRL register field description

Offset 0x04C — TX_IRQCTRL

31 to 8	7:0
Reserved	**TxIrqLastTransferEn**
R	R/W
0x000000	0x00

Table 65 TX_IRQCTRL register

Bit	Description	Configuration
TxIrqLastTransferEn	Enable interrupt on the last DataMover transfer completion	Enable interrupt on channel x Bit 0 = channel 0 … Bit 7 = channel 7

Table 66 TX_IRQCTRL register field description

Offset 0x050— TX_DATAMOVERCTRL

31 to 24	23:16	15:8	7:0
TxDataMoverRst	**TxDataMoverErr**	**TxDataMoverHaltComplt**	**TxDataMoverHaltReq**
R/W	R	R	R/W
0x00	0x00	0x00	0x00

Table 67 TX_DATAMOVERCTRL register

Bit	Description	Configuration
TxDataMoverRst	DataMover reset	1 = reset active on channel x Bit 24 = channel 0 … Bit 31 = channel 7
TxDataMoverErr	DataMover error status	1 if an error occurred on channel x Bit 16 = channel 0 … Bit 23 = channel 7
TxDataMoverHaltComplt	DataMover soft shutdown complete	1 if a soft shutdown completed on channel x Bit 8 = channel 0 … Bit 15 = channel 7
TxDataMoverHaltReq	DataMover soft shutdown request	1 = perform a soft shutdown on channel x Bit 0 = channel 0 … Bit 7 = channel 7

Table 68 TX_DATAMOVERCTRL register field description

Offset 0x054 + (x * 0x20) — TX_TRANSFERCNT_CHx

31 to 24	23:0
Reserved	**TxTransferCntChx**
R	R
0x00	0x000000

Table 69 TX_TRANSFERCNT_CHx register

Bit	Description	Configuration
TxTransferCntChx	DataMover transfer count command for channel x	DataMover count command for channel x

Table 70 TX_TRANSFERCNT_CHx register field description

Offset 0x058 + (x * 0x20) — TX_CURRENTTRANSFERCNT_CHx

31 to 24	23:0
Reserved	**TxCurrentTransferCntChx**
R	R
0x00	0x000000

Table 71 TX_CURRENTTRANSFERCNT_CHx register

Bit	Description	Configuration
TxCurrentTransferCntChx	DataMover current transfer count for channel x	DataMover current count for channel x

Table 72 TX_CURRENTTRANSFERCNT_CHx register field description

Offset 0x05C + (x * 0x20) — TX_DATAMOVERCTRL_CHx

31:0
TxDataMoverCtrlChx

R/W
0x000000

Table 73 TX_DATAMOVERCTRL_CHx register

Bit	Description	Configuration
TxDataMoverCtrlChx	DataMover control bit command	

Table 74 TX_DATAMOVERCTRL_CHx register field description

Offset 0x060 + (x * 0x20) — TX_DATAMOVERADDR_CHx

31:0
TxDataMoverAddrChx
R/W
0x000000

Table 75 TX_DATAMOVERADDR_CHx register

Bit	Description	Configuration
TxDataMoverCtrlChx	DataMover address bit command	

Table 76 TX_DATAMOVERADDR_CHx register field description

Offset 0x064 + (x * 0x20) — TX_DATAMOVERTAG_CHx

31 to 4	3:0
Reserved	**TxDataMoverTagChx**
R	R/W
0x0000000	0x0

Table 77 TX_DATAMOVERTAG_CHx register

Bit	Description	Configuration
TxDataMoverTagChx	DataMover tag bit command for channel x	

Table 78 TX_DATAMOVERTAG_CHx register field description

Offset 0x068 + (x * 0x20) — TX_DATAMOVERSTATUS_CHx

31 to 8	7:0
Reserved	**TxDataMoverStatusChx**
R	R
0x000000	0x00

Table 79 TX_DATAMOVERSTATUS_CHx register

Bit	Description	Configuration
TxDataMoverStatusChx	DataMover status for channel x	

Table 80 TX_DATAMOVERSTATUS_CHx register field description

5 FPGA Core Interfacing

5.1 Using the RTDEx Core

The RTDEx core has an AXI streaming interface similar to the one that Xilinx AXI Ethernet core and AXI DMA core have. The RTDEx core is meant to be Plug and Play to these cores. The RTDEx core can be inserted between the AXI Ethernet and AXI DMA (or AXI FIFO) cores. It has an internal switch to route the regular traffic between AXI DMA and AXI Ethernet. If RTDEx traffic is detected, this switch routes it between RTDEx RX and TX cores and AXI Ethernet without even reaching AXI DMA.

An example is given to show how the core can be inserted between AXI Ethernet and AXI DMA. Refer to the Perseus example section of the Perseus6x1x User's Guide.

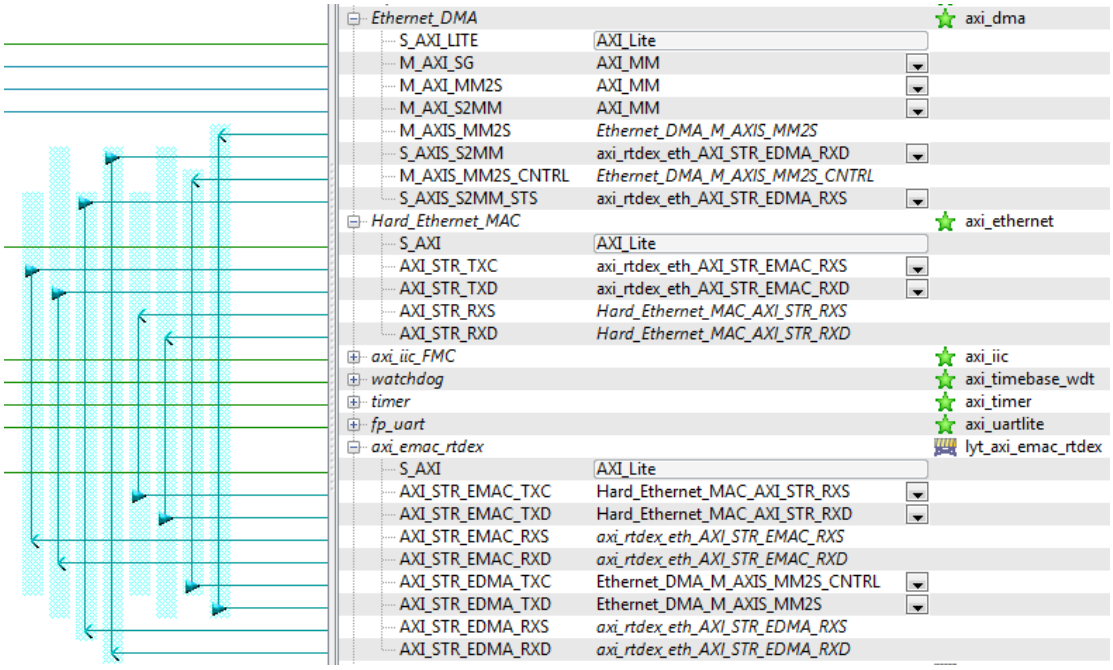

Figure 5-1 How to insert the RTDEx core between the AXI Ethernet and the AXI DMA

5.2 Interfacing the RTDEx Core to the User Logic

The user side of the RTDEx is a very simple FIFO interface to each channel. For RX channels, the user has a “FIFO not empty” (RxReady), a “Read strobe”, a “Data out”, and a “Data valid” signal. Signals of the unused channels can be left unconnected. All the FIFOs of the RX channels share the same user read clock.

For TX channels, the user has a “FIFO not full” (TxReady), a “Write strobe”, and a “Data input” signal. Signals of the unused channels can be left unconnected. All the FIFOs of the TX channels share the same user write clock.

Each data transfer direction can have its own clock for reading or writing the data. The data bus width is configured using generic parameter C_TX_WIDTH_CH# or C_RX_WIDTH_CH#. In Gigabit Ethernet and PCIe implementations, 8 channels are available and the host application can select the required one when opening the RTDEx pipe with the FPGA.

To transmit data, the TxReady flag must be high. This indicates that the FIFO will correctly handle a write operation. If this is the case, TxWe can be set high and the current data on TxData will be written to the transmit FIFO. If TxReady is low, this means that the FIFO is full or almost full and write operations must be avoided. This can occur when the user application does not read the data fast enough or the physical link cannot sustain the required throughput.

For receiving data in the FPGA, RxReady indicates if there is data available in the FIFO. If so, RxRe can be set high to perform a read operation. If the read operation is successful, RxDataValid will be high for the next clock cycle of RxUserClk and the data present on RxData will be valid.

The following figures describe timing of the RTDEx Tx and Rx ports.

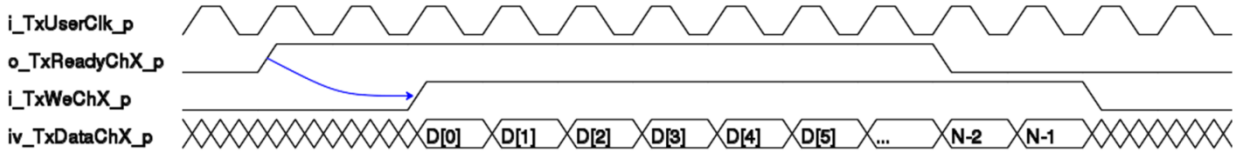

Figure 5-2 RTDEx TX port timing

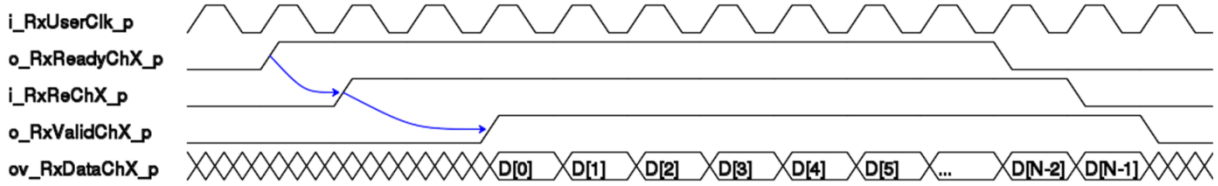

Figure 5-3 RTDEx RX port timing

5.3 Interfacing the RTDEx Core to the Record/Playback Core

The Record/Playback core can be connected to any RTDEx channel. It has dedicated RX and TX ports to connect to an RTDEx channel.

6 Host Programming

6.1 RTDEx Library

The RTDEx library is built to allow complete control over the transfers and give direct access to the received or transmitted data. Most functions use an RTDEx channel handle, created by the open function. One handle per channel per direction is necessary to operate a channel. It handles the PCI Express or the Gigabit Ethernet medium.

The following is a list of the main functions of the RTDEx library. For complete details refer to the Nutaq's API document.

Function	Description
RTDExResetCoreEth	This function resets the Ethernet RTDEx core and all its registers and FIFOs.
RTDExResetCorePcie	This function resets the PCIe RTDEx core and all its registers and FIFOs
RTDExOpenHostAndFpgaEth	This function opens a RTDEx channel handle for host to FPGA transfers on Ethernet.
RTDExOpenFpga2FpgaEth	This function opens a RTDEx channel handle for FPGA to FPGA transfers on Ethernet.
RTDExOpenHostAndFpgaPcie	This function opens a RTDEx channel handle for host to FPGA transfers on PCIe.
RTDExOpenFpga2FpgaPcie	This function opens a RTDEx channel handle for FPGA to FPGA transfers on PCIe.
RTDExSetTxFrameGapValEth	This function sets the frame gap value for transmitted packets in the ***from FPGA*** mode on Ethernet.
RTDExStart	The start function enables an RTDEx transfer on the FPGA side. In the ***from FPGA*** mode, the core starts to transmit data to the peer. In the ***to FPGA*** mode, the core is ready to receive data. The Ethernet packet size and the transfer size are determined by this call.
RTDExReceive	The receive function extracts data packets from the Ethernet interface and stores them in the defined receive buffer.
RTDExSend	The send function builds data packets from the defined transmit data buffer and sends it to the Ethernet interface for transmission.
RTDExCloseEth	Closes the Ethernet channel handle.
RTDExClosePcie	Closes the PCIe channel handle.

Table 81 RTDEx library function list

6.2 Control Connection between a Host and a Perseus AMC for PCIe RTDEx

A host computer writes and reads RTDEx registers (Ethernet version) through a TCP/IP connection with the CCE engine, an application running on the µBlaze CPU of the Perseus. This control connection uses the EMAC Ethernet interface of the Perseus for both implementation of the RTDEx; Gigabit Ethernet or PCIe.

In the PCIe RTDEx implementation, the control connection is not used to acces the RTDEx registers, but mandatory to correctly identify and associate the RTDEx core of a Perseus card with the Host Kernel Driver Instance.

When using the PCIe RTDEx implementation, the control connection can use either the EMAC Ethernet Interface or an Emulated Ethernet Interface built-in on the Kernel Host Driver. If using the second option, control data will use the PCIe bus to communicate with the CCE engine. A Kernel Driver must be installed and correctly configured on the Perseus to support this mode (named “Mailbox”). On the Host side, the installation script of the Kernel Driver must be properly configured to enable communication via the “Mailbox” of the Perseus. For the Host application to be able to control a Perseus in either mode, EMAC Ethernet or PCIe Mailbox, it only needs the IP address of the target Perseus.

6.3 Resource Limits when Using PCIe RTDEx

The PCIe RTDEx core transfers data between two FPGAs or an FPGA and a Host by using DMA engines. These DMA engines need Translation Windows (TWs) to map memory between the FPGA PCIe interface and the Host memory or the FPGA PCIe Interface of a second Perseus. There are 6 TWs per core. Only one TW is needed to support 8 reception channels (Host to FPGA) simultaneously, but one TW is needed for each transmission channel (FPGA to Host). The PCIe RTDEx core supports a maximum of 8 reception channels plus 5 transmission channels simultaneously. If no reception channels are needed, 6 transmission channels can be used. For Perseus to Perseus communication, one TW (on the source Perseus) is needed for each Perseus pair. The Kernel Host driver automatically allocates and counts TWs. It will return error code if the user tries to use more TWs than are available.

Example 1: 1 Perseus and 1 Host

Case 1:

Channels needed:

- 6 TX channels and 0 RX channel

Translation Windows usage:

- 6

Case 2:

Channels needed:

- 1 to 5 TX channels + 1 to 8 RX channels

Translation Windows usage:

- 2 to 6

Example 2: 2 Perseus and 1 Host

Case 1:

Channels needed:

- 8 channels PerseusA to PerseusB
- 8 channels PerseusB to PerseusA

Translation Windows usage:

- 1 on each Perseus

Case 2:

Channels needed:

- 1 to 8 channels (Host to PerseusA)
- 0 to 5 channels (PerseusA to Host)

Translation Windows usage:

- 1 to 6 on PerseusA

Case 3:

Channels needed:

- 1 to 8 channels (Host to PerseusA)
- 0 to 5 channels (PerseusA to PerseusB)
- 1 to 8 channels (Host to PerseusB)
- 0 to 5 channels (PerseusB to PerseusA)

Translation Windows usage:

- 1 to 6 on PerseusA
- 1 to 6 on PerseusB

6.4 Command Line Interface (CLI) Commands

The RTDEx CLI commands use glue logic implemented on top of the RTDEx library to offer an easy approach to execute transfers. The CLI commands handle both the PCI Express and the Ethernet medium.

The following is the list of RTDEx CLI commands. For complete details, refer to the Nutaq's *Board & Systems Software Tools - Command Line Interface API* guide.

Command	Description
rtdex_get	Instantiates a transfer in the ***from FPGA*** mode and receives data of the specified size.
rtdex_put	Instantiates a transfer in the ***to FPGA*** mode and transmits data of the specified size.
rtdex_framegap	Sets the frame gap value for transmitted packets in the ***from FPGA*** mode.

Table 82 RTDEx CLI command list

6.5 Building a Host Application Using RTDEx

The first step in building a host application using RTDEx is identifying the number of channels instantiated in the FPGA firmware in use.

The programmer will need to obtain an RTDEx channel handle per channel per direction using the open functions.

The first RTDEx call should always be the core reset followed by the necessary open calls.

6.5.1 Building a Host-to-Device Application

In the case the peer is a host PC on the Ethernet (host peer), the TX frame gap should be set to match the performances of the system used. Then, the data to transmit must be formatted to be passed as an argument to the send function. The start call enables the transfer to the FPGA side and the Ethernet or PCIe interface of the host peer starts receiving data.

The user then uses receive calls to fill the reception buffer and send calls to transmit data to the FPGA. In Single Transfer mode, the transfer ends when the transfer size specified in the start call is reached. Received and transmitted sizes must be verified to make sure that the application is not put in a reception timeout condition. In Continuous Transfer mode, the transfer size is not verified as it is infinite; the transfers are only terminated by a call to the close functions.

RTDEx statistics can be used to verify that all transfers happened correctly and that all the data was received without any missed packets.

6.5.2 Using Flow Control on the Ethernet RTDEx

If the RTDEx is feeding data from the host to a synchronous data consumer in the FPGA, the flow control is necessary to slow down the RTDEx transmitter to the exact rate of data consumption by the FPGA. The RTDExEnableFlowControlEth function allows the user to set the RTDEx FIFO thresholds, in the FPGA, at which pause frames and clear frames (pause frame of 0 µs) are sent.

- FIFO full threshold: Number of 32-bit samples in RTDEx FIFO for a pause frame of maximum time to be sent.
- FIFO empty threshold: Number of 32-bit samples in RTDEx FIFO for a pause frame of time 0 to be sent to clear the pause.

The RTDEx FIFO will oscillate between the full and empty thresholds. Overruns and underruns will be avoided if the host PC can sustain the data consumption rate.

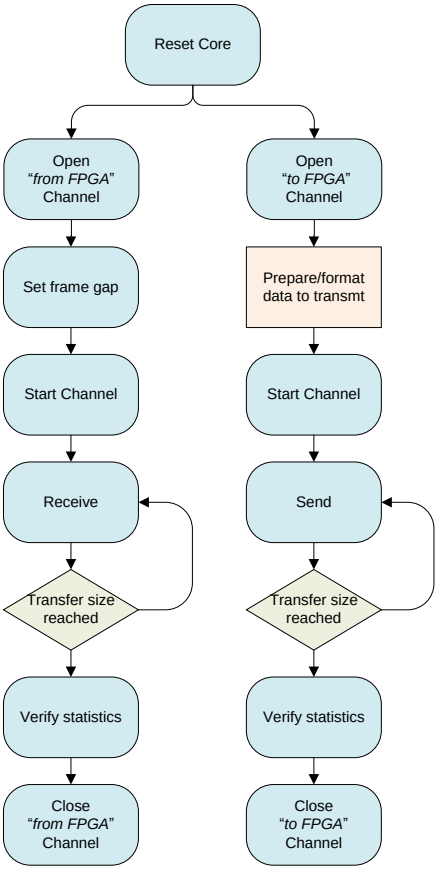

Figure 6-1 RTDEx host peer application flow chart

6.5.3 Using Flow Control on the PCIe RTDEx

The flow control on PCIe RTDEx is fully handled by the FPGA core and the PCIe driver. It is always active.

6.6 Increasing the Throughput of the PCIe RTDEx

There are two efficient ways to increase the throughput of the PCIe RTDEx.

The first way is to change the RTDEx packet size, which in the case of the PCI Express is the number of bytes transferred between two interrupts. In the case of the PCI Express, the higher it is within its limits, the less overhead the PC has to process large chunks of data. Through the characterization of the transfer speeds of the RTDEx, it was determined that 128k bytes is the optimal packet size to use.

Another factor in the way of a high RTDEx throughput on the PCI Express media is the data generation or consumption rate. The RTDEx is an asynchronous device and will only transmit data when it is available; therefore the throughput is always limited to the quantity of data available to be transmitted. On the reception side, the flow control of the PCIe will slow down the link to the data consumption speed of the FPGA.

In the case of the PCIe RTDEx example, the data generation and consumption is limited by the 100MHz system clock. Increasing the clock frequency of the data generation and consumption will increase the maximum possible transmitted or received data throughput.

When working with an MO1000 or an MI125 FMC, the user needs to make sure that the throughput is not limited by writing or reading to/from the RTDEx FIFOs at speeds that are too slow and that this speed is not limited by the system clock.

7 Host Applications Using the RTDEx EAPI Library

The BAS software suite provides two example utility applications that the user can modify. These two applications demonstrate how to stream data to and from the host using RTDEx GigE or PCIe, depending on the media that's instantiated in the FPGA.

- *RxStreaming* is used to stream data from the FPGA to the host.
- *TxStreaming* is used to stream data from the host to the FPGA.

Various examples showcase how these two applications are used and are available in your *bas**examples**directory*.

- Of those, two applicative examples directly demonstrate the RTDEx core and their utility functions. They reside in the *bas**examples**rtdex_recplay**host**scripts.*
    1. RtdexRxTest configures the RTDEx Test core to generate data. It is then streamed using RTDEx and verified on the host PC.
    2. RtdexTxTest transfers data from the host PC to the RTDEx Test core using RTDEx. The test core then verifies the data.

    Please refer to the RTDEx and Record Playback Examples Guide to execute these examples.
- Other examples demonstrating the use of Nutaq's FMCs also use the RTDEx FPGA core and the RTDEx Example Applications. Please refer to the example guide inherent to a given FMC to run a demo using the RTDEx Example Applications with this FMC.

7.1.1 RxStreaming description

Usage: *RxStreaming <configuration file name> <ip address 1> [<ip address 2>…]*

- Parameter 1 must be the Configuration file name, ex. RxStreaming.ini
- Parameter 2 must be the carrier IP address, ex. 192.168.0.101
- Parameters 3 and more are the additional carrier IP addresses

This application configures the RTDEx core to send data from the FPGA to the host. The application continuously receives the data and saves it to a file, specified in RxStreaming.ini.
The principal function of this application is function `RxStreaming()`.

1. For each carrier specified through the application's command line arguments,
    - The application calls function `connect_cce()` to connect to the carrier's CCE and get a connection state object
    - The application detects the media (GigE or PCIe using function `DetectMedia()` and, a few dozen lines of code below,
    - Resets and initializes the RTDEx core in the FPGA logic.
2. For each channel,
    - The application opens the RTDEx channel using function `RTDExOpenHostAndFpgaEth()` or `RTDExOpenHostAndFpgaPcie()`.
    - Then, it initializes two data queues. These data queues are used to transfer data between the threads that will be responsible for receiving the RTDEx stream, and writing the samples to a file. Each channel has a "Free buffer" queue, and a "Full buffer" queue.

- After initializing two mutex (in total, not per channel) and a semaphore, the application starts a `SaveTask` thread and a `RxTask` thread. `RxTask` calls `RTDExReceive()` in a loop and passes the data to the `SaveTask` thread using the buffer queues initialized before. `SaveTask` then stores the data to a file.

3. The application starts a common `DisplayTask` that displays the streaming rates for all channels.
4. The function then waits for all channel threads to end before printing final throughput stats.

7.1.2 RxStreaming configuration file

Instead of getting its parameters through command line arguments, application *RxStreaming* uses a separate configuration file to gather them. The path to this file is passed as an argument to the application. Details on how Nutaq's configuration files are built are available in the application note "*App Note - Parsing and Creating Configuration Files*" residing in the %BASROOT%*doc**app_notes* repository.

To get a feel of what the *RxStreaming* configuration file looks like, the reader is encouraged to open file *RtdexTestUplink.ini* in %BASROOT%*examples**rtdex_recplay**host**scripts* with a text editor. It is provided with the RTDEx applicative example. The file is similar to this example:

```
RTDEx_framesize=8192
RTDEx_queuesize=100
RTDEx_transfersize=204800000
RTDEx_burstsize=65536
RTDEx_framegap=0
real_time_flag=1
display_stats_flag=1

[RTDEx_1]

type=RTDEx
carrier_position=1
channel_number=1
filename=../bin/receivedramp.bin
```

Configuration files for using RxStreaming must contain all RTDEx parameters.

First, parameters common to all RTDEx channels. These parameters are not enclosed within a section:

- `RTDEx_framesize.` When the media is GigE, this is the Ethernet packet size used to transfer data between the host PC and the FPGA.
- `RTDEx_queuesize.` Depth of the data queues used in RxStreaming
- `RTDEx_transfersize.` Size of the transfer to perform before the application returns.
- `RTDEx_burstsize.` Refers to the number of bytes to receive (and wait for) each time function RTDExReceive() is called. Higher burst sizes may lead to better performance because of lower application overhead, because the function is called less often. Burst sizes too large must be avoided however to prevent timeouts.
- `RTDEx_framegap.` Number of FPGA clock cycles the FPGA logic waits for between each frame transmission.
- `real_time_flag.` Specifies whether the operating system grants this application the real-time privilege.
- `display_stats_flag.` Display streaming stats or not.

Then, parameters specific to each RTDEx channel. These parameters are contained within a section. Each section gives parameters for one RTDEx channel.

- `type` (an instance of that parameter must be equal to `RTDEx`)
- `carrier_position`. This is the carrier number this RTDEx channel will be instantiated on. This refers to the IP address list given as command line arguments to RxStreaming. For example, a value of "1" means that this RTDEx channel will run on the carrier configured with the first IP address specified in the list passed to the application.
- `channel_number`. The RTDEx channel number to use on the carrier.
- `filename`. Name of the file in which received data is to be stored.

7.1.3 TxStreaming description

Usage: *TxStreaming <configuration file name> <ip address 1> [<ip address 2>…]*

- Parameter 1 must be the Configuration file name, ex RxStreaming.ini
- Parameter 2 must be the carrier IP address, ex 192.168.0.101
- Parameters 3 and more are the additional carrier IP addresses

This application configures the RTDEx core to receive data from the host to the FPGA. The application continuously reads data from a file and sends it to the FPGA through RTDEx.

The principal function of this application is function `TxStreaming()`.

1. For each carrier specified through the application's command line arguments,
    - The application calls function `connect_cce()` to connect to the carrier's CCE and get a connection state object
    - The application detects the media (GigE or PCIe using function `DetectMedia()` and, a few dozen lines of code below,
    - Resets and initializes the RTDEx core in the FPGA logic. It also enables the flow control mechanism using function `RTDExEnableFlowControlEth()`. This will prevent the host from overflowing the RTDEx FIFO buffer in the FPGA RTDEx core. This may happen if the FPGA user logic doesn't consume the data at a fast enough rate.
2. For each channel,
    - The application opens the RTDEx channel using function `RTDExOpenHostAndFpgaEth()` or `RTDExOpenHostAndFpgaPcie()`.
    - Then, it initializes two data queues. These data queues are used to transfer data between the threads that will be responsible for reading the data from the file, and sending the RTDEx stream. Each RTDEx channel has a "Free buffer" queue, and a "Full buffer" queue.
    - After initializing two mutex (in total, not per channel) and a semaphore, the application starts a `ReadTask` thread. This thread continuously reads from the data file and puts the data into a queue. The application then starts the RTDEx channel, which makes the RTDEx channels ready for transmission. Finally, the application starts `TxTask` which receives the data from the ReadTask thread and calls `RTDExSend()` on the data.
3. The application starts a common `DisplayTask` that displays the streaming rates for all channels.
4. The function then waits for all channel threads to end before printing final throughput stats.

7.1.4 TxStreaming configuration file

Instead of getting its parameters through command line arguments, application *TxStreaming* uses a separate configuration file to gather them. The path to this file is passed as an argument to the

application. Details on how Nutaq's configuration files are built are available in the application note "*App Note - Parsing and Creating Configuration Files*" residing in the %BASROOT%*doc**app_notes* repository.

To get a feel of what the *TxStreaming* configuration file looks like, the reader is encouraged to open file *RtdexTestDownlink.ini* in %BASROOT%*examples**rtdex_recplay**host**scripts* with a text editor. It is provided with the RTDEx applicative example. The file is similar to this example:

```
RTDEx_framesize=8192
RTDEx_queuesize=100
RTDEx_transfersize=204800000
RTDEx_burstsize=65536
real_time_flag=1
display_stats_flag=1

[RTDEx_1]

type=RTDEx
carrier_position=1
channel_number=1
filename=../bin/generatedramp.bin
```

Configuration files for using TxStreaming must contain all RTDEx parameters.

First, parameters common to all RTDEx channels. These parameters are not enclosed within a section:

- `RTDEx_framesize.` When the media is GigE, this is the Ethernet packet size used to transfer data between the host PC and the FPGA.
- `RTDEx_queuesize.` Depth of the data queues used in TxStreaming
- `RTDEx_transfersize.` Size of the transfer to perform before the application returns.
- `RTDEx_burstsize.` Refers to the number of bytes to send each time function RTDExSend() is called. Higher burst sizes may lead to better performance because of lower application overhead, because the function is called less often. But, because TxStreaming sends data to each RTDEx channel in rotation, burst sizes too large must be avoided. This is because a channel may starve on the FPGA side if the application is busy sending a large burst to another channel.
- `real_time_flag.` Specifies whether the operating system grants this application the real-time privilege.
- `display_stats_flag.` Display streaming stats or not.

Then, parameters specific to each RTDEx channel. These parameters are contained within a section. Each section gives parameters for one RTDEx channel.

- `type` (an instance of that parameter must be equal to `RTDEx`)
- `carrier_position`. This is the carrier number this RTDEx channel will be instantiated on. This refers to the IP address list given as command line arguments to TxStreaming. For example, a value of "1" means that this RTDEx channel will run on the carrier configured with the first IP address specified in the list passed to the application.
- `channel_number`. The RTDEx channel number to use on the carrier.
- `filename`. Name of the file from which data is read.

7.1.5 Exploring the Applications Source Code and Modifying/Rebuilding it

Project folders, allowing the user to modify the source code of the applications are also available. They reside in *bas**tools**apps**core*.

If rebuilding is needed, follow the steps described in the section appropriate to the operating system you are using, where <*application to rebuild*> is either

- *RxStreaming,*
- *TxStreaming,*

7.1.5.1 Windows

A folder named *prj_win* which contains the Visual Studio 2012 solution associated with the application is present in the folders listed above.

1. Start Microsoft Visual Studio.
2. On the **File** menu, point to **Open** and click **Project/Solution**.
3. Browse to the *bas**tools**apps**core*\<*application to rebuild*> folder and select the <*application to rebuild*>*.sln* solution
4. Select the build configuration Release x64.
5. On the **Build** menu, click **Build Solution**.

7.1.5.2 Linux

A folder named *prj_linux* which contains a shell script and a makefile associated with the application is present in the folders listed above. Source files are available in the *inc* and *src* folders

1. Open a terminal and use the *cd* command to browse to the *bas**tools**apps**core*\<*application to rebuild* > repository in the installation.
2. To build the application, run the following command in the Linux terminal.

   sudo ./build_demo.sh

8 MBDK Programming and Interfacing

8.1 FPGA MBDK

You will find the documentation describing the RTDEx FPGA MBDK block in the following document:

BASROOT\doc\html\mbdk_fpga_rtdex.htm

9 Specifications and Benchmarking

This section presents the measured performances of the RTDEx core under different scenarios.

9.1 Gigabit Ethernet

Note:
The size of the transfers in jumbo frame mode was set to 8960 in the Windows environment and to 2496 in the Linux environment.

Host Peer Scenario			Windows 7, 64 Bits	Linux, 64 Bits
Host to Perseus on 1 channel	Normal frames (1472 bytes)	Number of packets	102400	102400
		Transfer size (in MB)	143.75	143.75
		Throughput	12.6	114.0
	Jumbo frames (8960 bytes)	Number of packets	16384	16384
		Transfer size (in MB)	140	140
		Throughput	54.7	116.7
Perseus to host on 1 channel	Normal frames (1472 bytes)	Number of packets	102400	102400
		Transfer size (in MB)	143.75	143.75
		Frame gap used	10000	0
		Throughput	13.5	114.0
	Jumbo frames (8960 bytes)	Number of packets	16384	16384
		Transfer size (in MB)	140	140
		Frame gap used	10000	0
		Throughput	70.6	116.5
Host to Perseus on 8 channels	Normal frames (1472 bytes)	Number of packets	102400	102400
		Transfer size (in MB)	143.75	143.75
		Throughput total	12.7	113,2
	Jumbo frames (8960 bytes)	Number of packets	16384	16384
		Transfer size (in MB)	140	140
		Throughput total	54.2	116.8
Perseus to host on 8 channels	Normal frames (1472 bytes)	Number of packets	102400	102400
		Transfer size (in MB)	143.75	143.75
		Frame gap used	10000	10000
		Throughput total	13.5	13.5
	Jumbo frames (8960 bytes)	Number of packets	16384	16384
		Transfer size (in MB)	140	140
		Frame gap used	10000	0

Host Peer Scenario			Windows 7, 64 Bits	Linux, 64 Bits
		Throughput total	69,7	116.7
Full-duplex on 8 channels	Normal frames (1472 bytes)	Number of packets	102400	102400
		Transfer size (in MB)	143.75	143.75
		Frame gap used	10000	10000
		Throughput total RX	14.3	111.8
		Throughput total TX	13.6	13.3
	Jumbo frames (8960 bytes)	Number of packets	16384	16384
		Transfer size (in MB)	140	140
		Frame gap used	10000	10000
		Throughput total RX	64.2	116.0
		Throughput total TX	70,0	22,4

Table 83 RTDEx host peer performances

FPGA Peer Scenario			Any Host
Perseus to Perseus on 1 channel	Normal frames (1472 bytes)	Number of packets	102400
		Transfer size (in MB)	143.75
		Throughput	114.7
	Jumbo frames (8960 bytes)	Number of packets	16384
		Transfer size (in MB)	140
		Throughput	118.4
Perseus to Perseus on 8 channels	Normal frames (1472 bytes)	Number of packets	102400
		Transfer size (in MB)	143,75
		Throughput per channel	
		Throughput total	113.8
	Jumbo frames (8960 bytes)	Number of packets	16384
		Transfer size (in MB)	140
		Throughput per channel	
		Throughput total	117.4
Perseus to Perseus, full-duplex on 8 channels	Normal frames (1472 bytes)	Number of packets	102400
		Transfer size (in MB)	143.75
		Throughput per channel RX	
		Throughput per channel TX	
		Throughput total RX	113.8
		Throughput total TX	113.8
	Jumbo frames (8960 bytes)	Number of packets	16384
		Transfer size (in MB)	140
		Throughput per channel RX	
		Throughput per channel TX	
		Throughput total RX	117.5
		Throughput total TX	117.5

Table 84 RTDEx FPGA peer performances

9.2 PCI Express

This section presents the measured performances of the PCIe-based RTDEx core under different scenarios.

9.2.1 Hardware Description

This is a description of the hardware used in the computing of the performances for the PCIe-based RTDEx core.

µTCA Chassis

Nutaq Pico box

Host Computer

HP Compaq Elite 8300 desktop computer based on Core i7 3770

Clock: 3.4GHz

Operating system: Fedora 17, 64-bit

Perseus

Perseus AMC with a LX240 FPGA

9.2.2 Performance

In this section, the performance numbers measured for different scenarios are presented. The description of the hardware used to perform these tests is presented in section 9.2.1.

Scenario 1

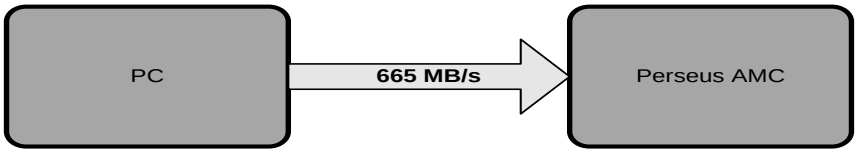

Figure 9-1 PCIe performance for scenario 1

Scenario 2

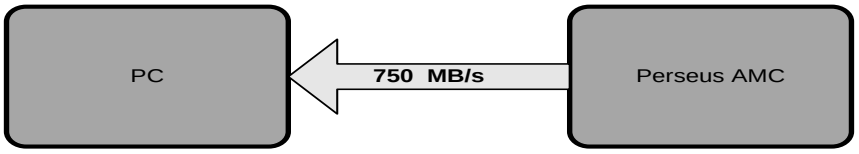

Figure 9-2 PCIe performance for scenario 2

Scenario 3

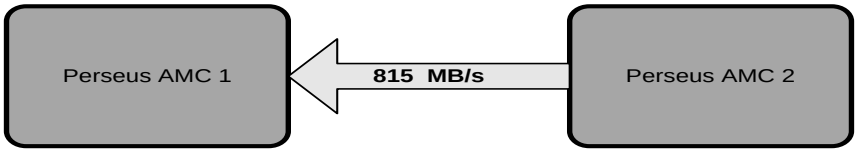

Figure 9-3 PCIe performance for scenario 3