Nutaq® INNOVATION TODAY FOR TOMORROW®

AURORA

Programmer's Reference Guide

February 2017

nutaq .com

Revision history

Revision	Date	Comments
0.9	September 2013	First draft
1.0	October 2013	Formatting Up to date for Software Tools Release 6.6
2.0	July 2015	Update to new Aurora core implementation. Up to date for Board & System release 7.0
2.1	October 2015	New glossary
2.2	May 2016	Formatting
2.3	February 2017	Modifications: - Formatting - Removed unwanted page break - Removed CLI section - Clarified data rates with a table

Version 2.2

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 Aurora 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. 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.
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 Aurora FPGA core allows high speed data transfers between two platforms over a point-to-point connection. The Aurora core is implemented to use a group of 4 gigabit transceivers to transmit and receive raw data stream.

2.1 Outstanding Features

Transfer speed of 3.125 Gbps or 5 Gbps on 4 lanes

The Aurora core will transfer data using a 8b/10b encoding on 4 gigabit transceiver lanes. The following table shows the maximum effective transfer rates.

Aurora Data Rate (GHz)	Effective Data Rate Per Lane (Gbps)	Effective Data Rate Per Core of 4 Data Lanes (Gbps)
3.125	2.5	10.0
5.0	4.0	16.0

Table 2 Aurora data rates

Multiple cores available simultaneously

On the Perseus 601X platform, up to three cores can coexist as long as the ports are not used by another core in the FPGA design. The connections between two Perseus 601X using the Aurora core to transfer data are point-to-point through the AMC backplane. The three available interfaces are:

- AMC Fat Pipe 4-7
- AMC Fat Pipe 8-11
- AMC Fat Pipe 17-20

On the Perseus 611X platform, up to 8 cores can coexist as long as the ports are not used by another core in the FPGA design. The Perseus 611X connections using the Aurora core to transfer data are one point-to-point connection through the AMC backplane and seven point-point connections through the Rear Transition Modules (RTM). The available interfaces are :

- AMC Fat Pipe 4-7
- RTM #1 0-3
- RTM #1 4-7
- RTM #1 8-11
- RTM #2 0-3
- RTM #2 4-7
- RTM #2 8-11
- RTM #2 12-15

The RTM #1 12-15 is not accessible since its FPGA gigabit transceivers are used for AMC Fat Pipe 4-7 instead.

Full-duplex

The Aurora core allows data transfers in both directions at the same time. When the Aurora core is started, data is continuously sent.

Simple FIFO interface

The user interface of the Aurora core is a simple FIFO with control signals. On the Aurora physical link side, the FIFO is 128 bits wide (4 bytes on each of the 4 lanes), but the user side is configurable with different element sizes (16, 32, 64, 128).

Individual resets

The user can reset the RX and TX FIFO independently from each other. The Aurora transceiver core can also be reset but when the link is not up between the FIFO in both directly are reset.

3 AURORA FPGA Core Description

The Aurora core uses a FIFO-based interface on the user's side.

3.1 Aurora Parameters

Parameter	Type	Values Allowed	Default Value	Description
AURORA_SPEED [(1)]	Integer	0,1,2,3	0	Speed and clock source selection 0 = 3G125_125MHz 1 = 3G125_156.25MHz 2 = 5G_125MHz 3 = 5G_156.25MHz The first value specifies the desired communication rate in Gbps. The second value specifies clock source frequency used by the gigabit transceiver.
USER_DATA_WIDTH [(2)]	Integer	16, 32, 64, 128	16	User data width FIFO interface
REFCLK_FREQ_MHZ	INTEGER	-	100	Frequency of the reference clock specified in MHz.

Table 3 Aurora generic parameters

NOTES:

(1) User should select a source clock that is not used by another core in the system (Ethernet, PCIe, etc) and after specify that, specify its frequency.

(2) Aurora side data width is always 128 bits, so user must know that the core will wait until 128 bits is in the FIFO before sending one sample. For example, with a user data width of 32 bits, the user will write 4 samples in the FIFO before the Aurora core outputs the data on the AMC backplane.

3.2 Core Ports

Port	Range	Direction	Description
User FIFO ports			
i_UserClk_p	-	In	User clock that will clock all the port in this section
o_RxReady_p	-	Out	'1' when the channel is up and new data are available in the receive FIFO.
i_RxReadReq_p	-	In	Request a read transaction from the FIFO. A new data sample will be read from the FIFO if the FIFO is ready.
ov_RxData_p	[USER_DATA_WIDTH-1:0]	Out	Received data sample.
o_RxDataValid_p	-	Out	Indicates that a new data sample is present on the ov_RxData_p port. This data is valid 1 clock cycle after a read request on a ready FIFO is performed.
o_TxReady_p	-	Out	'1' when the channel is up and space is available in the transmit FIFO.
i_TxWriteReq_p	-	In	When set to '1', data at the iv_TxData_p port will be written inside the FIFO if it is ready.
iv_TxData_p	[USER_DATA_WIDTH-1:0]	In	Data sample to send.
Gigabit transceiver ports			
iv4_Aurora_GTX_RX_p	[3:0]	In	Aurora gigabit transceiver RX Lanes.
iv4_Aurora_GTX_RX_n	[3:0]	In	
ov4_Aurora_GTX_TX_p	[3:0]	Out	Aurora gigabit transceiver TX Lanes.
ov4_Aurora_GTX_TX_n	[3:0]	Out	
i_Aurora_GTX_CLK_p	-	-	Aurora gigabit transceiver clock. This clock frequency must be specified by the AURORA_SPEED define.
i_Aurora_GTX_CLK_n	-	-	
System and control ports			
i_Rst_p	–	In	Resets the core internal logic and registers.
i_RefClk_p	–	In	Reference clock for the data rate counter. Its frequency must be specified by the generic parameter REFCLK_FREQ_MHZ.
i_RegWrEn_p	-	In	Interface to access the internal core registers. These ports must be connected to the Aurora ports of the platform register core. This will allow the internal register to be read and written from the processor.
iv16_RegIdx_p	[15:0]	In	
iv32_RegData_p	[31:0]	In	
ov32_RegData_p	[31:0]	Out	

Table 4 Aurora core ports

3.3 Core Registers (Indirect Addressing)

The core registers are accessible through indirect addressing. The indirect access register ports must be connected to the related platform register core ports.

To write a value to an internal register, set the index register and the write register with proper values, set the write enable high (1) and set it back to low (0).

To read the value of an internal register, set the index register and then read the read register.

Register name	Index	Direction	Description
COREID	0x0	R	Core ID and version
CONTROL_STATUS	0x1	R/W	Control and status of the Aurora core.
RECEIVE_DATA_COUNT_MSW	0x2	R	Receive data count most significant word.
RECEIVE_DATA_COUNT_LSW	0x3	R	Receive data count least significant word.
TRANSMIT_DATA_COUNT_MSW	0x4	R	Transmit data count most significant word.
TRANSMIT_DATA_COUNT_LSW	0x5	R	Transmit data count least significant word.
RECEIVE_DATA_RATE	0x6	R	Receive data rate
TRANSMIT_DATA_RATE	0x7	R	Transmit data rate
GTX_PARAMETER	0x8	R/W	GTX parameters in transmission
GTX_DFEEYEDACMON	0x9	R	GTX parameters in reception

Table 5 RTDEx channel registers

Offset 0x0 — COREID

This register shows the Aurora IP core identification and version number.

Bit	31 to 16	15 to 0
Name	core_id	core_version
Direction	R	R
Reset value	0xAAAA	0x0200

Table 6 COREID register

Name	Description	Configuration
core_id	This is the Aurora IP core unique ID number.	0xAAAA
core_version	This is the Aurora IP core version number.	0x0200 = Version 2.0

Table 7 COREID register fields description

Offset 0x1 — CONTROL_STATUS

This register is use to control the core and get status

Bit	31 to 26	25	24	23	22
Name	RESERVED	Latch_Data_Count	RESERVED	Aurora_Reset	Aurora_Hard_Error
Direction	R	R/W	R	R/W	R
Reset value	0	0	0	0	0
Bit	**21**	**20**	**19 to 16**	**15**	**14 to 12**
Name	Aurora_Soft_Error	Aurora_Channel_Up	Aurora_Lanes_Up	TX_Fifo_Reset	RESERVED
Direction	R	R	R	R/W	R
Reset value	0	0	0	0	0
Bit	**11**	**10 to 8**	**7**	**6 to 1**	**0**
Name	TX_Fifo_Full	Aurora_Channel_Up	RX_Fifo_Reset	RESERVED	RX_Fifo_Empty
Direction	R	R	R	R/W	R
Reset value	0	0	0	0	0

Table 8 CONTROL_STATUS register

Name	Description	Configuration
RX_Fifo_Empty	FIFO Empty Flag	1: The RX FIFO is empty 0: The RX FIFO is not empty
RX_Fifo_Reset	Reset the FIFO interface and content	1: Reset the RX FIFO 0: Clear the RX FIFO reset
TX_Fifo_Full	FIFO Full Flag	1: The TX FIFO is full 0: The TX FIFO is not full
TX_Fifo_Reset	Reset the FIFO interface and content	1: Reset the TX FIFO 0: Clear the TX FIFO reset
Aurora_Lanes_Up	Indicate which lane is up in the Aurora Channel	1: The lane corresponding to this bit location is up 0: The lane corresponding to this bit location is up
Aurora_Channel_Up	Indicate that the Aurora channel is up. The status is 1 when a valid Aurora link is etablished between two carriers. The status is 0 when there is no physical connection between the Aurora core or if the signal integrity is not good enough over the physical link. When the channel is down, it automatically reset the RX and TX FIFO.	1: The channel is up. Data can be transferred between the platform. 0: The channel is down
Aurora_Soft_Error	Aurora Soft Error Flag. See Xilinx Aurora documentation.	1: Soft error 0: No error
Aurora_Hard_Error	Aurora Hard Error Flag. See Xilinx Aurora documentation.	1: Hard error 0: No error.
Aurora_Reset	Force reset of the Xilinx Aurora core. See Xilinx Aurora documentation.	1: Reset Aurora core 0: Clear reset.
Latch_Data_Count	On the rising edge of this bit, the RX and TX data rate counter value are latched to be stable signals when the processor will read their values.	1: Latched the values on the rising-edge 0: Set this bit back to 0 for the next latch.

Table 9 CONTROL_STATUS register fields description

Offset 0x2 — RECEIVE_DATA_COUNT_MSW

Most significant word (bits 63 down to 32) of the total number of bytes received by the Aurora core since the last RX FIFO reset.

Bit	31 to 0
Name	RECEIVE_DATA_COUNT_MSW
Direction	R/W
Reset value	0

Table 10 RECEIVE_DATA_COUNT_MSW register

Offset 0x3 — RECEIVE_DATA_COUNT_LSW

Least significant word (bits 31 down to 0) of the total number of bytes received by the Aurora core since the last RX FIFO reset.

Bit	31 to 0
Name	RECEIVE_DATA_COUNT_LSW
Direction	R/W
Reset value	0

Table 11 RECEIVE_DATA_COUNT_LSW register

Offset 0x4 — TRANSMIT_DATA_COUNT_MSW

Most significant word (bits 63 down to 32) of the total number of bytes sent by the Aurora core since the last TX FIFO reset.

Bit	31 to 0
Name	TRANSMIT_DATA_COUNT_MSW
Direction	R/W
Reset value	0

Table 12 TRANSMIT_DATA_COUNT_MSW register

Offset 0x5 — TRANSMIT_DATA_COUNT_LSW

Least significant word (bits 31 down to 0) of the total number of bytes sent by the Aurora core since the last TX FIFO reset.

Bit	31 to 0
Name	TRANSMIT_DATA_COUNT_LSW
Direction	R/W
Reset value	0

Table 13 TRANSMIT_DATA_COUNT_LSW register

Offset 0x6 — RECEIVE_DATA_RATE

Reception data rate (bytes/second) by the Aurora core. This value is updated every second.

Bit	31 to 0
Name	RECEIVE_DATA_RATE
Direction	R/W
Reset value	0

Table 14 RECEIVE_DATA_RATE register

Offset 0x7 — TRANSMIT_DATA_RATE

Transmission data rate (bytes/second) by the Aurora core. This value is updated every second.

Bit	31 to 0
Name	**TRANSMIT_DATA_RATE**
Direction	R/W
Reset value	0

Table 15 TRANSMIT_DATA_RATE register

Offset 0x8 — GTX_PARAMETER

This register is use to set the gigabit transceiver parameters.

See Xilinx UG366, table “TX Configurable Driver Ports” and table “RX Equalizer Ports” for more information.

Bit	31 to 21	20 to 18	17 to 13	12 to 9	8 to 5	4 to 0
Name	RESERVED	rxeqmix	dfetap1	txdiffctrl	txpreemphasis	txpostemphasis
Direction	R	R/W	R/W	R/W	R/W	R/W
Reset value	0	6	0	10	0	0

Table 16 GTX_PARAMETER register

Name	Description	Configuration
txpostemphasis	Transmitter Post-Cursor TX Pre-Emphasis Control.	Value from 0 to 31
txpreemphasis	Transmitter Pre-Cursor TX Pre-Emphasis Control.	Value from 0 to 15
txdiffctrl	Driver Swing Control	Value from 0 to 15
dfetap1	DFE tap 1 weight value control	Value from 0 to 31
rxeqmix	Receiver Equalization Control	Value from 0 to 7

Table 17 GTX_PARAMETER register fields description

Offset 0x9 — GTX_DFEEYEDACMON

Averaged Vertical Eye Height (voltage domain) used by the DFE as an optimization criterion.

See Xilinx UG366, table RX Equalizer Ports for more information.

Bit	31 to 20	19 to 15	14 to 10	9 to 5	4 to 0
Name	RESERVED	lane3	lane2	lane1	lane0
Direction	R	R	R	R	R
Reset value	0	0	0	0	0

Table 18 GTX_DFEEYEDACMON register

Name	Description	Configuration
lane0	Averaged Vertical Eye Height of the first lane.	Value from 0 to 31
lane1	Averaged Vertical Eye Height of the second lane.	Value from 0 to 31
lane2	Averaged Vertical Eye Height of the third lane.	Value from 0 to 31
lane3	Averaged Vertical Eye Height of the fourth lane.	Value from 0 to 31

Table 19 GTX_DFEEYEDACMON register fields description

3.4 Interfacing the Aurora Core to the User Logic

The Aurora core can be easily inserted in an XPS project. The control of the core is done with Nutaq platform register core. The Aurora control ports must be connected to the platform register Aurora ports. One can refer to its carrier Programmer’s Reference Guide for using this core.

The user side of the Aurora core is a very simple FIFO interface as described in the Core Ports section. The RX and TX FIFO user interface share the same user clock.

4 Low-Level and Host Programming

This chapter provides the functions of the AURORA driver, the EAPI library, the CLI commands as well as the procedure to build a host application.

4.1 Aurora Driver

The table below presents the functions and use of the Aurora driver on the embedded processor.

Function	Use
Aurora_Open	Create a handle for use in the Aurora API.
Aurora_GetVersion	Retrieve the FPGA core version of the opened Aurora instance.
Aurora_ResetCore	Reset the Xilinx Aurora FPGA core.
Aurora_GetChannelStatus	Get the channel status. The status is 1 when a valid Aurora link is etablished between two carriers. The status is 0 when there is no physical connection between the Aurora core or if the signal integrity is not good enough over the physical link. When the channel is down, it automatically reset the RX and TX FIFO.
Aurora_ResetRxFifo	Reset the reception FIFO memory. It also reset the RX data and data rate count.
Aurora_ResetTxFifo	Reset the transmission FIFO memory. It also reset the TX data and data rate count.
Aurora_GetRxDataCount	Retrieve the total number of bytes received by the Aurora core since the last RX FIFO reset.
Aurora_GetTxDataCount	Retrieve the total number of bytes received by the Aurora core since the last TX FIFO reset.
Aurora_GetRxDataRate	Retrieve the reception data rate (bytes/second) by the Aurora core. This value is updated every second.
Aurora_GetTxDataRate	Retrieve the transmission data rate (bytes/second) by the Aurora core. This value is updated every second.
Aurora_SetGtxTxParam	Set the GTX parameters in transmission.
Aurora_GetGtxTxParam	Get the GTX parameters in transmission.
Aurora_SetGtxRxParam	Set the GTX parameters in reception.
Aurora_GetGtxRxParam	Get the GTX parameters in reception.
Aurora_GetDfeEyeDacMon	Averaged Vertical Eye Height (voltage domain) used by the DFE as an optimization criterion.
Aurora_Close	Close the Aurora core handle

Table 20 Aurora driver functions

4.2 Aurora EAPI Library

The table below presents the functions and use of the Aurora EAPI library.

Function	Use
Aurora_GetVersion_send	Retrieve the FPGA core version of the opened Aurora instance.
Aurora_ResetCore_send	Reset the Xilinx Aurora FPGA core.
Aurora_GetChannelStatus_send	Get the channel status. The status is 1 when a valid Aurora link is etablished between two carriers. The status is 0 when there is no physical connection between the Aurora core or if the signal integrity is not good enough over the physical link. When the channel is down, it automatically reset the RX and TX FIFO.
Aurora_ResetRxFifo_send	Reset the reception FIFO memory. It also reset the RX data and data rate count.
Aurora_ResetTxFifo_send	Reset the transmission FIFO memory. It also reset the TX data and data rate count.
Aurora_GetRxDataCount_send	Retrieve the total number of bytes received by the Aurora core since the last RX FIFO reset.
Aurora_GetTxDataCount_send	Retrieve the total number of bytes received by the Aurora core since the last TX FIFO reset.
Aurora_GetRxDataRate_send	Retrieve the reception data rate (bytes/second) by the Aurora core. This value is updated every second.
Aurora_GetTxDataRate_send	Retrieve the transmission data rate (bytes/second) by the Aurora core. This value is updated every second.
Aurora_SetGtxTxParam_send	Set the GTX parameters in transmission.
Aurora_GetGtxTxParam_send	Get the GTX parameters in transmission.
Aurora_SetGtxRxParam_send	Set the GTX parameters in reception.
Aurora_GetGtxRxParam_send	Get the GTX parameters in reception.
Aurora_GetDfeEyeDacMon_send	Averaged Vertical Eye Height (voltage domain) used by the DFE as an optimization criterion.

Table 21 Aurora EAPI library functions

5 Host Applications Using the Aurora EAPI Library

5.1 Building Your Own Host Application Using Aurora EAPI Library

The following diagram illustrates the typical flow of an Aurora application.

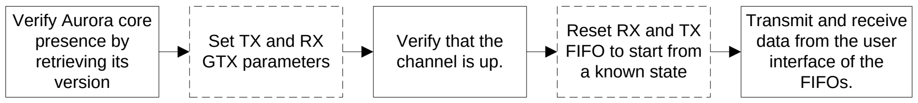
Verify Aurora core presence by retrieving its version
Set TX and RX GTX parameters
Verify that the channel is up.
Reset RX and TX FIFO to start from a known state
Transmit and receive data from the user interface of the FIFOs.

Figure 5-1 Aurora application creation flow

The first step taken by a programmer building an Aurora application should be to verify that the Aurora core he wants to use is present in its current FPGA bitstream.

Once it is done, the GTX parameters can be changed as required. The default values should satisfy most of the hardware configurations. If the link is not up or you detect error in the data transfer, change the GTX parameter values. See Xilinx UG366, table “TX Configurable Driver Ports” and table “RX Equalizer Ports” for more information.

When the channel is up, the TX and RX FIFO can be reset to start from a known state. This step is not required since the TX and RX FIFO are automatically reset when the Aurora channel is down.

In the FPGA design, writes and reads at the user FIFO interface can be performed to exchange data with the other platform.

The BAS software suite provides two example utility applications that the user can modify. Those are explained in section 5.2.

5.2 Using and Modifying the Aurora Example Applications

Two example utility applications are provided with your BAS software suite. They are command line applications that are usable on the spot to configure, use and monitor your Aurora channels. Pre-compiled executables are available in your *bas\tools\bin* directory as *AuroraInitChannel* and *AuroraMonitor,* with project folders available in your *bas\tools\apps\core* directory.

An applicative example showcases how those applications are used and is available in your *bas\examples\aurora\host\scripts* directory. Please refer to the Aurora Examples Guide for details on how to run this applicative example.

5.2.1 AuroraInitChannel Description

Usage: *AuroraInitChannel <ip address> <channel> (RX | TX | BOTH)*

- Parameter 1 is the carrier IP address, ex.: 192.168.0.101
- Parameter 2 is the Aurora channel, ex.: 0. This refers to the Aurora core instance number in the FPGA design.
- Parameter 3 is the Aurora direction, ex.: {RX, TX, BOTH}

This application initializes an Aurora channel.

1. It configures the related GTX to a known to be working configuration, by calling `Aurora_SetGtxRxParam_send()`.
2. It resets the Rx and/or Tx FIFO(s) to ensure the channel starts from a known state, by calling `Aurora_ResetRxFifo_send()`.

5.2.2 AuroraMonitor Description

Usage: *AuroraMonitor <ip address>*

- Parameter 1 is the Perseus IP address, ex.: 192.168.0.101

This application monitors all Aurora channels that are instantiated on the Perseus. Every second, the application prints statistics regarding the number of bytes received and sent, the data rate, and the channel status. It exits when the user hits any key.

5.2.3 Exploring the Applications Source Code for Modification/Rebuild

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

Windows

A folder named *prj_win* which contains the Visual Studio 2012 solution associated with the application is present in the folders listed above. If rebuilding is needed, follow these steps, where <*application to rebuild*> is either *AuroraInitChannel* or *AuroraMonitor*:

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**.

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. If rebuilding is needed, follow these steps, where <*application to rebuild*> is either *AuroraInitChannel* or *AuroraMonitor*:

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