XAPP777 (v1.0) December 3, 2004

A Gigabit Ethernet to Aurora Bridge

Author: Phil James-Roxby

Summary

The design described in this application note utilizes the Virtex-II Pro™ RocketIO™ transceivers, the Xilinx Aurora Protocol Engine and the 1-Gigabit Ethernet MAC core to provide a bridge between Aurora and Gigabit Ethernet. Bridges are common networking devices, and are used to connect multiple LANs. Unlike routers, bridges operate at the data link layer and are independent of protocols above this layer.

The design can be used to connect a system using the Aurora protocol to legacy Gigabit Ethernet networks. In addition, it can act as a starting point for systems wishing to use either Gigabit Ethernet, or Aurora for general data transfer. One way of considering the design is as two separate designs glued together by FIFOs, one handling general Aurora traffic, and the second handling general Gigabit Ethernet traffic. Each of these designs is useful either as part of a larger system, or glued together to form the bridge. Target applications include:

- Connecting Aurora devices to legacy Gigabit Ethernet networks
- Testing Aurora devices using Gigabit Ethernet traffic
- Larger systems requiring Aurora or Gigabit Ethernet interfaces

Introduction

Bridges are common networking devices used to connect multiple LANs. Unlike routers, bridges operate at the data link layer and are independent of protocols above this layer. Routers, on the other hand, operate at the network layer, and thus a pure IP router can only handle IP packets.

Tanenbaum [Ref 1] highlights six reasons why bridges might be required:

- Combine legacy networks
- Overcome geographical constraints
- Split a single LAN into multiple LANs to accommodate the load
- Reduce the round-trip delay
- Increase reliability
- Increase security

Bridges can be used to link together LANs using different data-link technologies.

Aurora is a scalable, lightweight, link-layer protocol that can be implemented in any silicon device/technology. It is used to move data across point-to-point serial links at a baud rate of up to 3.125 Gb/s per lane. Aurora provides a transparent interface to the physical serial links. This allows upper layers of proprietary or industry-standard protocols, such as Ethernet or TCP/IP, to easily use high-speed serial links, leading to higher connectivity performance while preserving software infrastructure investment.

This reference design describes a bridge that allows connecting a system with an Aurora interface to a Gigabit Ethernet LAN. The bridge is designed as a store-and-forward system, and is designed to have zero latency, where latency is defined as the delay between receiving the last bit of a packet and beginning the onward transmission of that packet. The bridge has a single Aurora port and a single Gigabit Ethernet port, and has been tested both by simulation and on hardware.

Design Elements

Figure 1 shows a block diagram of the reference design. The sub-sections that follow describe each of the design elements.

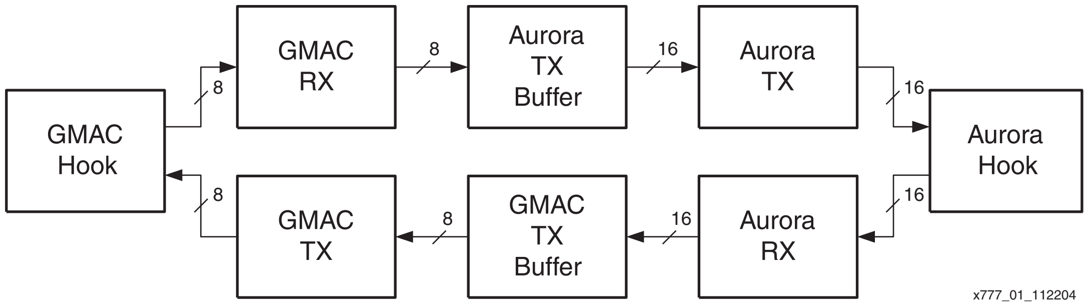


Figure 1: **System Block Diagram**

The GMAC hook and Aurora hook both contain a Xilinx LogiCORE™ module with additional logic to allow them to be easily interfaced to the rest of the system.

Top-Level Signals

Table 1: **Serial I/O Signal Descriptions**

Signal Name	I/O	Description
Aurora_TXP, Aurora_TXN	O	Differential transmitter outputs to the Aurora host.
Aurora_RXP, Aurora_RXN	I	Differential receiver inputs from the Aurora host.
Ethernet_TXP, Ethernet_TXN	O	Differential transmitter outputs to the Ethernet LAN.
Ethernet_RXP, Ethernet_RXN	I	Differential receiver inputs from the Ethernet LAN.

Table 2: **Clock and Reset Signal Descriptions**

Signal Name	I/O	Description
SysClkP	I	Positive edge-triggered LVDS input clock for RocketIO transceiver (BREFCLK).
SysClkN	I	Negative edge-triggered LVDS input clock for RocketIO transceiver (BREFCLK).
reset	I	Active High push-button reset input.

GMAC Hook

The GMAC hook contains the 1-Gigabit Ethernet MAC (GMAC) core with PCS/PMA sublayers (1000BASE-X). This core is a purchasable LogiCORE™ module delivered through the CORE Generator™ software. Xilinx provides two ways to evaluate the GMAC core:

- **Simulation-Only Evaluation** allows customization of the core through a CORE Generator GUI and generation of a Simprim-based, gate-level model for functional simulation.
- A **Full-System Hardware Evaluation** version of the GMAC core allows the same functionality as the fully licensed IP core, including configuration, place and route, simulation, timing estimation, and programming a Xilinx FPGA device. Because the

evaluation version of the core contains a time-out mechanism, however, it cannot be used in final customer systems.

On power-up, the core is initially isolated from the Gigabit transceivers. The GMAC hook contains a component that modifies the control register in the PCS sublayer, allowing normal operation.

GMAC RX Thread

The GMAC RX thread handles the incoming packet data from the GMAC hook, sending it to the Aurora TX buffer. The thread has two possible modes of operation, controlled by the PASS_MAC_HEADER generic:

Table 3: **GMAC RX Generics**

Name	Description
PASS MAC_HEADER	0 = The source and destination MAC addresses are suppressed, and the L/T field is passed to memory 1 = The whole MAC header is passed to memory

The GMAC RX thread counts the incoming bytes and sets the length of the packet in the Aurora TX buffer. This thread is activated when valid input data is detected by the GMAC hook. The GMAC RX thread also monitors the status signals, defining the successful or unsuccessful reception of the current packet. For example, if the packet is marked as successfully received, the GMAC RX thread commits the current packet writes to the memory, which then affects the write pointer as described later. If the CRC is incorrect, the packets are marked as unsuccessfully received. An unsuccessful reception means the GMAC RX thread does *not* commit the current packet writes—and because the write pointer is unmodified, the next packet received overwrites the contents of the packet memory.

GMAC TX Thread

The GMAC TX thread handles the outgoing packet data from the GMAC TX buffer, sending it to the GMAC hook. The thread has two possible modes of operation, controlled by the PASS_MAC_HEADER generic:

Table 4: **GMAC TX Generics**

Name	Description
PASS MAC_HEADER	0 = The source and destination MAC addresses are set to default values by the thread, rather than being passed from memory 1 = The whole MAC header is passed from memory

The GMAC TX thread uses the length of the packet stored in the memory to determine packet boundaries, and is activated when an entire packet is stored in the GMAC TX buffer.

Aurora Hook

The Aurora hook contains the Aurora protocol engine, delivered through the CORE Generator software. This is a free core after a registration process. The protocol engine is a serial backplane standard that is tailored for maximum efficiency and ease of use in high-performance, point-to-point data transmission systems. The protocol engine includes the Virtex-II Pro RocketIO Multi-Gigabit Transceiver (MGT). The protocol engine provides a LocalLink user interface that is used to encapsulate data packets. For additional information on the Aurora Protocol specification or the protocol engine, go to the Xilinx web site at www.xilinx.com.

The Aurora hook also contains a clock compensation module generated as part of the protocol engine.

The Aurora hook does not check the various error signals generated by the Aurora core, such as the framing error signal. If desired, this reference design could be easily modified to add support for these signals in a manner similar to the GMAC mechanisms described previously.

Aurora RX Thread

The Aurora RX thread is responsible for dealing with incoming data from the Aurora hook, which it does through the LocalLink interface. The thread uses the *RX_SOF_N* and *RX_EOF_N* signals from the protocol engine to determine packet boundaries. The bridge does not perform any error checking on the incoming data stream.

Aurora TX Thread

The Aurora TX thread takes data from the Aurora TX buffer and passes it unmodified to the Aurora hook through the LocalLink interface. The packet boundaries are determined using the stored packet length in the buffer, and the thread is activated when an entire packet has been stored in the Aurora TX buffer.

Memories

The two memories in the bridge are packet-level FIFOs with random access to packet fields. This means that packets are stored first-in first-out, but the actual contents of those packets can be read and written to in any order. The packet data is preceded by a 16-bit length field which gives the length of the packet in bytes. The read and write widths are different to reflect the 8-bit interface of the GMAC and the 16 bit interface of the Aurora core in use.

The memories support a commit mechanism that allows the fields in a packet to be written as many times are required, before being finalized by a commit action. This also supports the dropping of a packet that has been written in the memory, if required. Unlike a word-level FIFO, writing a word in these memories does not affect the write pointer—rather, all writes are done *in relation to* the write pointer. A symmetrical mechanism exists for reads.

Finally, the memories also support the generation of flow control signals, explained later in this application note. (See section "Flow Control," page 12.)

Design Implementation

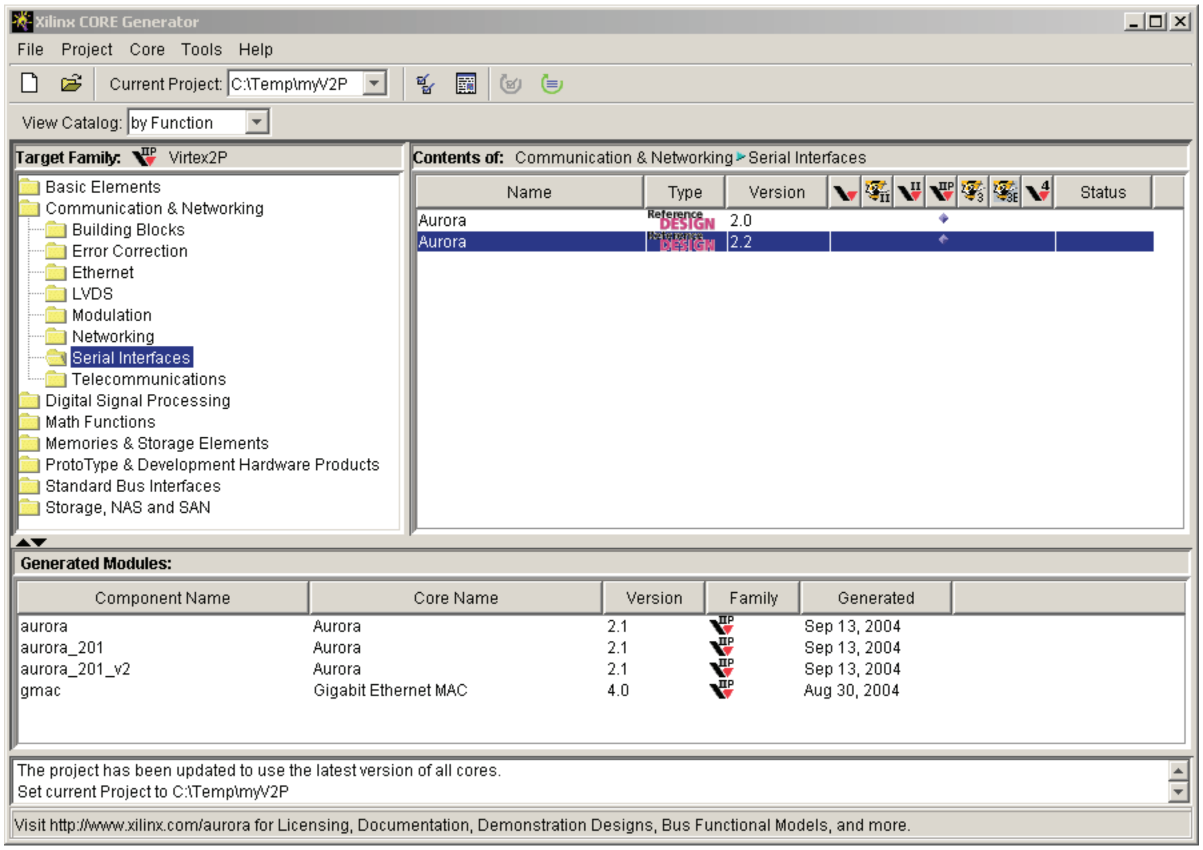
Generating the IP Cores

The IP cores are not released with this reference design. Instead, you must obtain licenses in each case, and use the CORE Generator software to produce the required files. The next steps explain in detail which options should be selected for each core.

Generating the Aurora Module

First, register for the Aurora Solutions Suite at www.xilinx.com/Aurora, and install the Aurora license.

Next, start the CORE Generator application in stand-alone mode, and select the Aurora Reference Design Version 2.2. (See Figure 2.)


Xilinx CORE Generator
File Project Core Tools Help
Current Project: C:\Temp\myV2P
View Catalog: by Function
Target Family: Virtex2P
Basic Elements
Communication & Networking
Building Blocks
Error Correction
Ethernet
LVDS
Modulation
Networking
Serial Interfaces
Telecommunications
Digital Signal Processing
Math Functions
Memories & Storage Elements
ProtoType & Development Hardware Products
Standard Bus Interfaces
Storage, NAS and SAN
Contents of: Communication & Networking ▸ Serial Interfaces

Name	Type	Version								Status
Aurora	Reference DESIGN	2.0				◆				
Aurora	Reference DESIGN	2.2				◆				

Generated Modules:

Component Name	Core Name	Version	Family	Generated
aurora	Aurora	2.1		Sep 13, 2004
aurora_201	Aurora	2.1		Sep 13, 2004
aurora_201_v2	Aurora	2.1		Sep 13, 2004
gmac	Gigabit Ethernet MAC	4.0		Aug 30, 2004

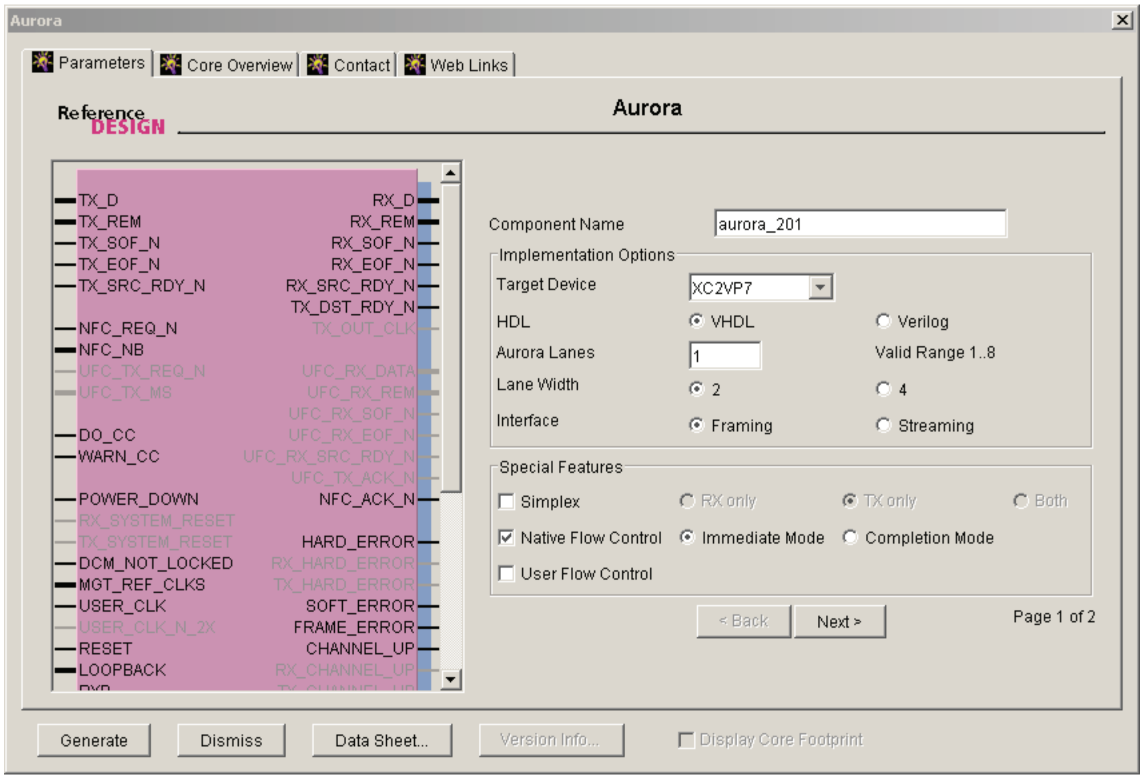
```
The project has been updated to use the latest version of all cores.
Set current Project to C:\Temp\myV2P
```

Visit http://www.xilinx.com/aurora for Licensing, Documentation, Demonstration Designs, Bus Functional Models, and more.

x777_02_112404

Figure 2: **Step 1 in Generating the Aurora Module**

Name the component, select **VHDL** as the HDL, and de-select **User Flow Control**. All other default settings are OK. (See Figure 3.)



Aurora

Component Name: aurora_201

Implementation Options

Target Device: XC2VP7

HDL: [✓ VHDL] Verilog

Aurora Lanes: 1 Valid Range 1..8

Lane Width: [✓ 2] 4

Interface: [✓ Framing] Streaming

Special Features

Simplex: RX only [✓ TX only] Both

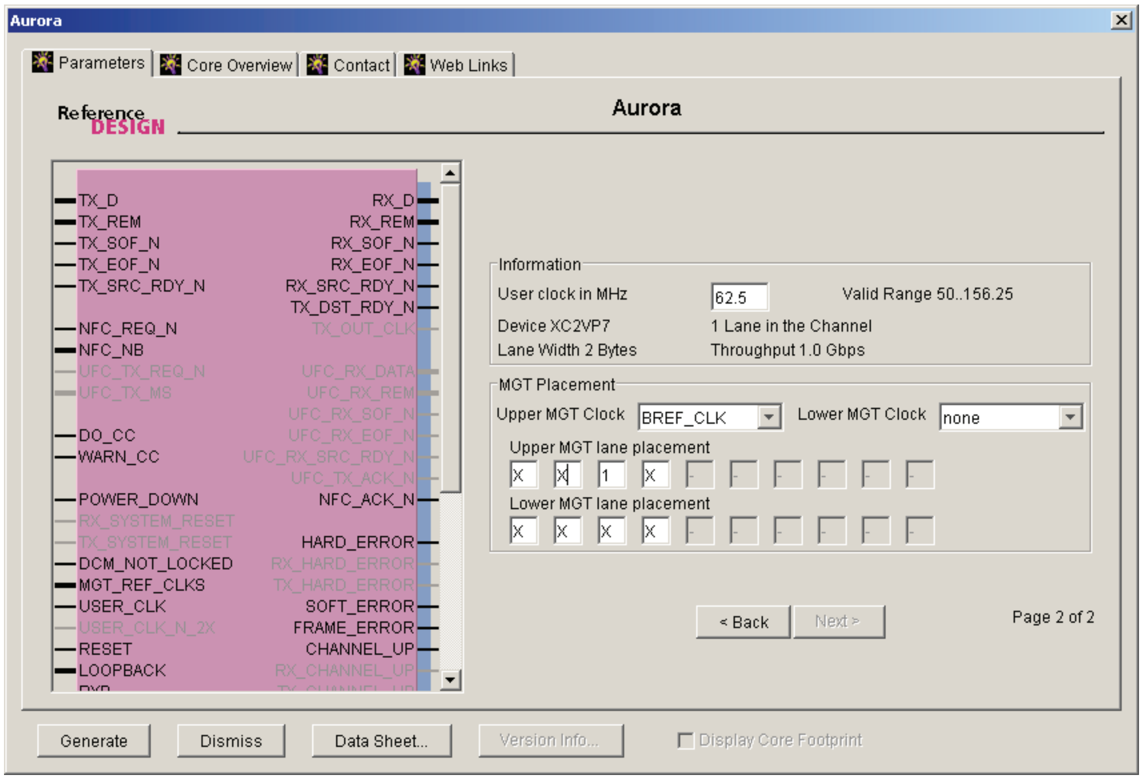
[✓ Native Flow Control] [✓ Immediate Mode] Completion Mode

User Flow Control

Page 1 of 2

Figure 3: **Step 2 in Generating the Aurora Module**

Finally, for *User clock in MHz*, select **62.5**; for *Upper MGT Clock*, select **BREF_CLK**; and for *Upper MGT lane placement*, select the third lane by placing a "1" in the third box. Finally, click **Generate**. (See Figure 4.)



Information

User clock in MHz 62.5 Valid Range 50..156.25

Device XC2VP7 1 Lane in the Channel

Lane Width 2 Bytes Throughput 1.0 Gbps

MGT Placement

Upper MGT Clock BREF_CLK Lower MGT Clock none

Upper MGT lane placement: X X 1 X - - - - - -

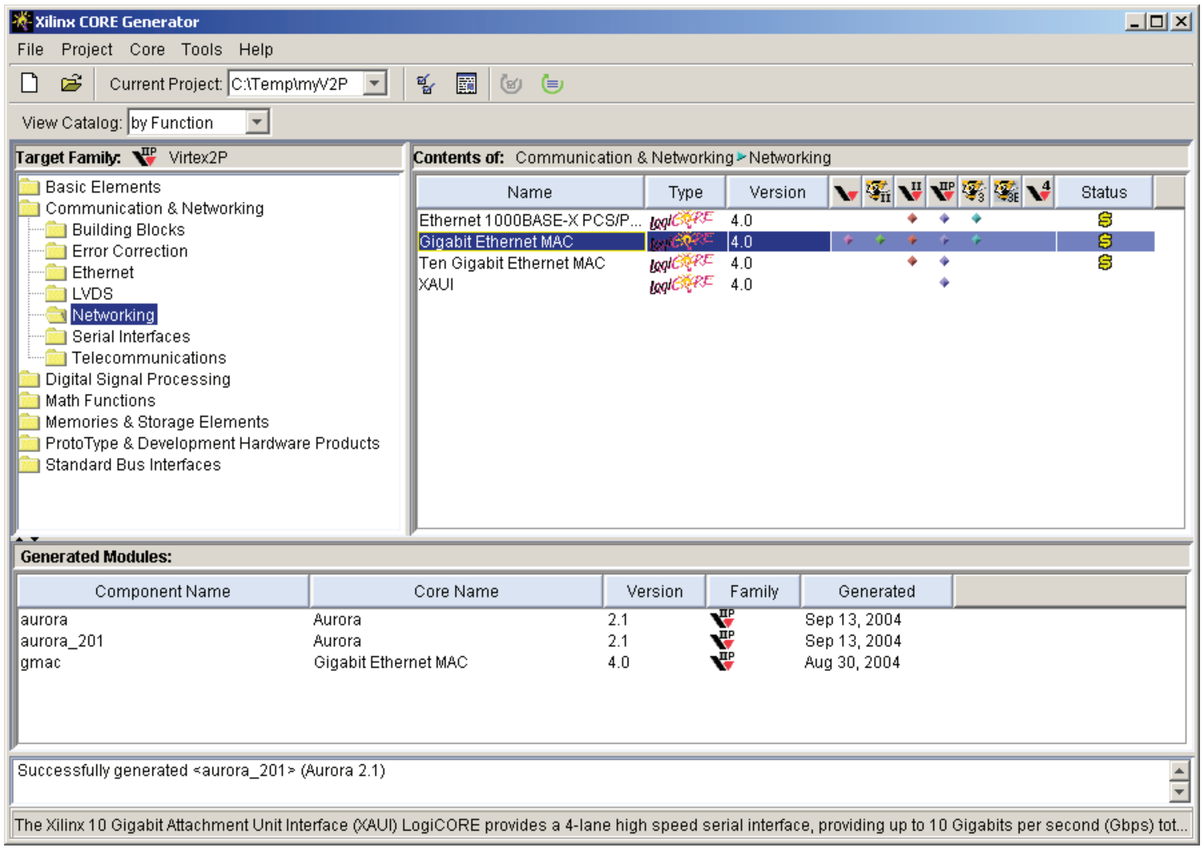
Lower MGT lane placement: X X X X - - - - - -

x777_04_112404

Figure 4: **Step 3 in Generating the Aurora Module**

Gigabit Ethernet MAC

The Gigabit Ethernet MAC is also generated through the CORE Generator software. Select **Gigabit Ethernet MAC** from the *Networking* group. (See Figure 5.)


Xilinx CORE Generator
File Project Core Tools Help
Current Project: C:\Temp\myV2P
View Catalog: by Function
Target Family: Virtex2P
Contents of: Communication & Networking ▸ Networking

- Basic Elements
- Communication & Networking
  - Building Blocks
  - Error Correction
  - Ethernet
  - LVDS
  - Networking
  - Serial Interfaces
  - Telecommunications
- Digital Signal Processing
- Math Functions
- Memories & Storage Elements
- ProtoType & Development Hardware Products
- Standard Bus Interfaces

Name	Type	Version
Ethernet 1000BASE-X PCS/P...	LogiCORE	4.0
Gigabit Ethernet MAC	LogiCORE	4.0
Ten Gigabit Ethernet MAC	LogiCORE	4.0
XAUI	LogiCORE	4.0

Generated Modules:

Component Name	Core Name	Version	Family	Generated
aurora	Aurora	2.1		Sep 13, 2004
aurora_201	Aurora	2.1		Sep 13, 2004
gmac	Gigabit Ethernet MAC	4.0		Aug 30, 2004

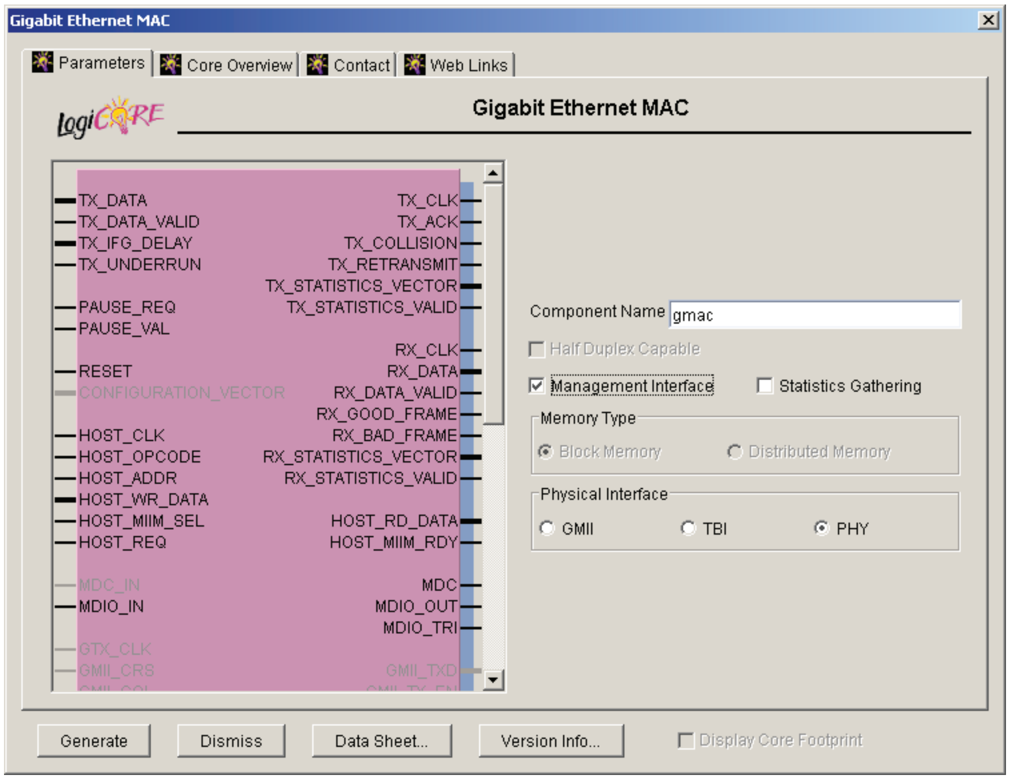
Successfully generated <aurora_201> (Aurora 2.1)

The Xilinx 10 Gigabit Attachment Unit Interface (XAUI) LogiCORE provides a 4-lane high speed serial interface, providing up to 10 Gigabits per second (Gbps) tot...

x777_05_091504

Figure 5: **Step 1 in Generating the GMAC Module**

Then name the component, deselect **Statistics Gathering**, and select **PHY** as the *Physical Interface*. Finally, click **Generate**. (See Figure 6.)



Component Name: gmac

[ ] Half Duplex Capable

[✓] Management Interface   [ ] Statistics Gathering

Memory Type: [✓ Block Memory] Distributed Memory

Physical Interface: GMII TBI [✓ PHY]

[ ] Display Core Footprint

Figure 6: **Step 2 in Generating the GMAC Module**

Next, run the `implement` script that is generated in the `implement` directory by the CORE Generator software. This synthesises the core, and generates a gate-level simulation model in the `test/vhdl` directory. It is then necessary to copy the two files `routed.sdf` and `routed.vhd` into the `sim` directory of the reference design.

Running the Simulation

The following simulation data is provided:

- A functional simulation directory
- VHDL functional simulation using stimulus objects
- Simulation scripts for ModelSim

SWIFT models are used for RocketIO transceiver simulation. A SWIFT compliant simulator must be used with this reference design, and the SWIFT models must be correctly set up. See Xilinx Answer Record 14019 for details on how to do this.

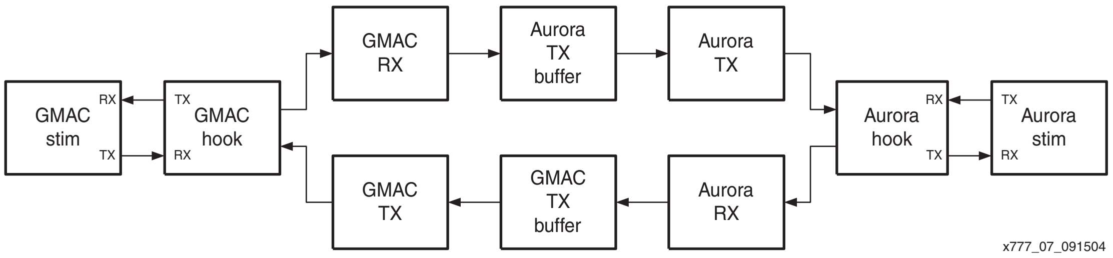


Figure 7: **Simulation Environment**

The two stimulus objects, GMACstim and AuroraStim, together with the simulation scripts are designed to show the end-to-end operation of the bridge. These components produce a series of test packets, designed to show the handling of both legal and illegal packets.

AuroraStim produces a sequence of three packets: a short even-sized packet, a short odd-sized packet, and a longer even-sized packet. GMACstim produces a sequence of four frames, one of which is received as a bad frame, and is not passed through the bridge. In each case, the files are designed to be easily read and modified.

To run the simulation, sample `sim.do` and `compile.do` scripts are provided. The `sim.do` script calls `compile.do` and works without modification. The `compile.do` script needs modifying to reflect the various locations of the CORE Generator cores and the reference design at the deployed site. This is a straightforward task that must be done only once.

The simulation takes around 10 minutes to run to 34 µs.

Implementation

The bridge can be implemented on a Xilinx ML300 board using the constraint files provided. To retarget the design to a different board, the `top.ucf` file in the `implement` directory would need modifying to reflect the new board setup.

It is necessary to copy two files generated by the `implement` script of the GMAC core into the `implement` directory: `gmac.ngo` and `gmac_gmac_gen1.ngc`. The implementation scripts assume the Aurora core was generated in a directory called `cores` at the same level of the file hierarchy as the `implement` directory. If this is not the case, either copy the generated Aurora core into a new directory called `cores`, or modify the `top.prj` file to indicate the current location of this core.

After copying these files, run the following two commands in the `implement` directory:

```
xst -ifn top_xst.scr
```

```
xflow -p xc2vp7ff672-6 -implement high-effort -config bitgen top.ngc
```

This produces a configuration bitstream called `top.bit`.

To configure the ML300 board, run:

```
impact -batch impact.cmd
```

The Gigabit Ethernet port is marked **GigE 0** on the underside of the ML300 board, and the Aurora port uses **GigE 1**. A simple way to test the design is to connect the Gigabit Ethernet port to an optical Ethernet card in a PC using an LC-SC cable. For the Aurora side, make a loopback cable from an SC-SC cable, and plug this into the Aurora port. (See Figure 8.) Then all packets (such as pings) sent from the PC are reflected back to the PC. Depending on setup, the headers might be different if the GMACTX and GMACRX threads are set up not to pass the MAC headers to the Aurora side.

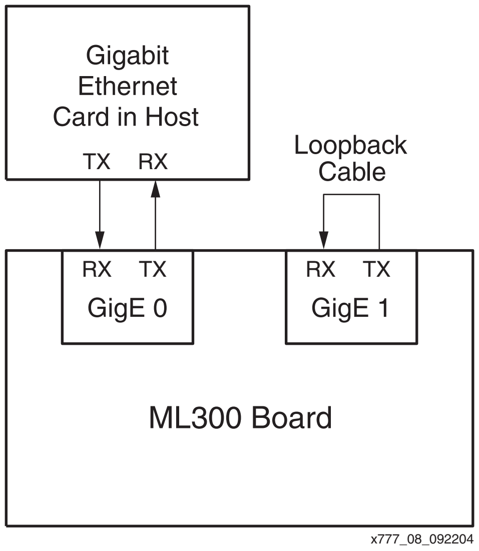


Figure 8: **Test Setup**

Testing Using ChipScope

This section is only applicable for ChipScope™ customers.

The bridge can be tested using ChipScope, where packets can be seen arriving and leaving on both interfaces on the Xilinx ML300 board. Figure 9 contains a block diagram of the ChipScope setup.

It is necessary to copy two files generated by the `implement` script of the GMAC core into the `debug` directory: `gmac.ngo` and `gmac_gmac_gen1.ngc`. After copying these files, run the following two commands in the `debug` directory:

```
xst -ifn top_xst.scr
```

```
xflow -p xc2vp7ff672-6 -implement high-effort -config bitgen top.ngc
```

This produces a configuration bitstream called `top.bit`.

Start the ChipScopePro Analyzer tool, and load the `bridge.cpj` project. Configure the Virtex-II Pro FPGA with the configuration bitstream generated in the `debug` directory.

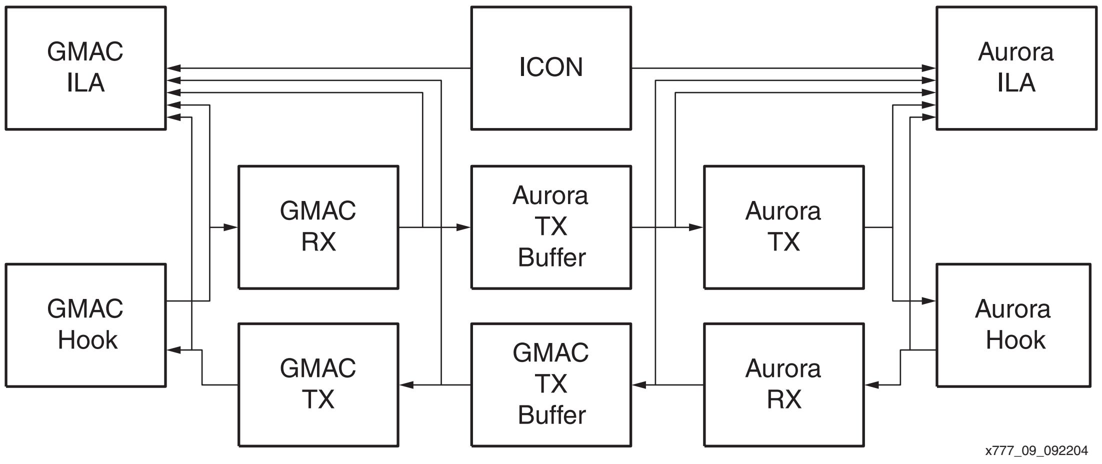


Figure 9: **ChipScope Setup**

To trigger on the start of the reception of a GMAC packet, set the GMAC ILA trigger condition to be `XX1X_XXXX`. To trigger on the start of the transmission of a GMAC packet, set the GMAC

ILA trigger condition to be `1XXX_XXXX`. To trigger on the start of the reception of an Aurora packet, set the Aurora ILA trigger condition to be `XXXX_0XXX`. To trigger on the start of the transmission of an Aurora packet, set the Aurora ILA trigger condition to be `XX0X_XXXX`.

The trigger signals are shown in Table 5.

Table 5: **ILA Trigger Signals**

	ILA Trigger:	
	Aurora	**GMAC**
Bit 0	AuroraTXready	0
Bit 1	RX_SRC_RDY_N	DCM_LOCKED
Bit 2	RX_EOF_N	Reset
Bit 3	RX_SOF_N	CommitGMACpacketWrite
Bit 4	TX_DST_RDY_N	GMACTXready
Bit 5	TX_SOF_N	GMACRXdataValid
Bit 6	AuroraError	GMACTXack
Bit 7	Reset	GMACTXdataValid

Flow Control

General

For maximum flexibility in the memories, a hysterisis-based scheme is used. A low-tide and high-tide mark can be defined on the memories, defining the point at which the incoming flow should be stopped, and the point at which it can be started again. This is shown in Figure 10.

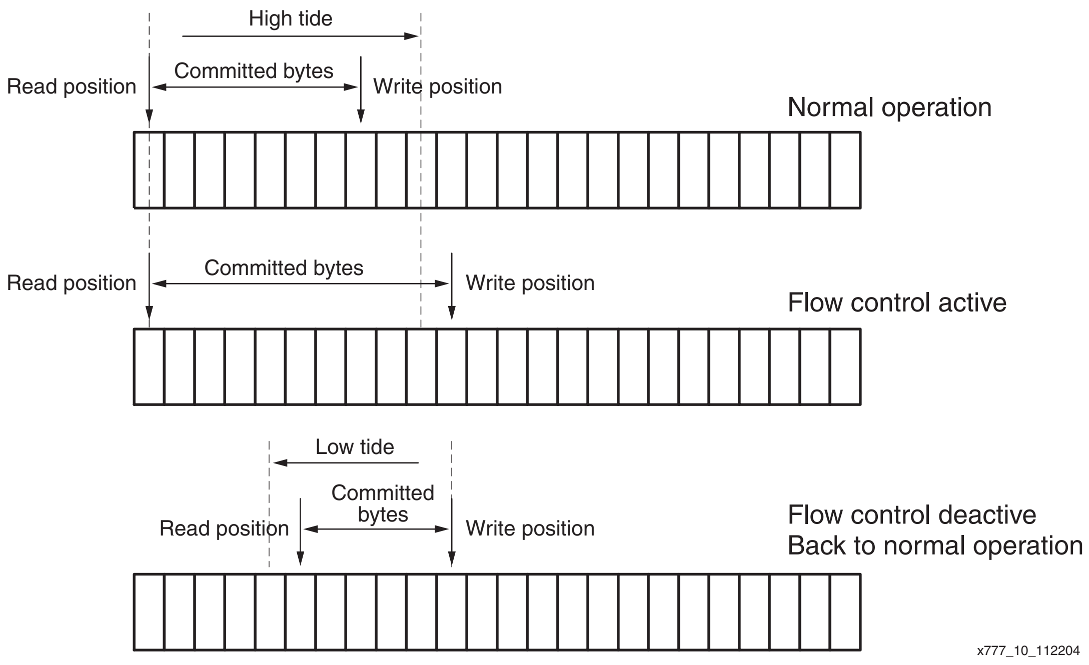


Figure 10: **Hysteresis-Based Flow Control**

The tide marks are defined in terms of bytes committed to the memory. While flow control is useful to prevent buffer overflow, there is a penalty in terms of reduced throughput when flow control is activated. Some experimentation by the designer is necessary to determine appropriate low- and high-tide mark values based on their particular traffic patterns.

To guarantee no overflow of the buffers, the high-tide mark should be set at the size of the buffer minus the size of the largest legal packet minus 1. The low-tide mark does not affect whether the buffers will overflow.

Flow Control in Aurora

Flow control in Aurora can be achieved by two methods, each of which is fully documented in the Aurora module documentation. For the bridge, the Native Flow Control (NFC) mechanism was used. NFC allows the receiver of an Aurora message to signal to the transmitter to slow down or even stop completely. This is done by requesting the number of NFC words that should be interspersed with the data words. NFC also supports either immediate mode or completion mode. In completion mode, the current transfer is completed before NFC words are sent. In immediate mode, NFC words are sent while the current transfer is taking place.

The Aurora reception needs to be slowed down when the buffer feeding the GMAC transmitter becomes nearly full, as shown in Figure 11.

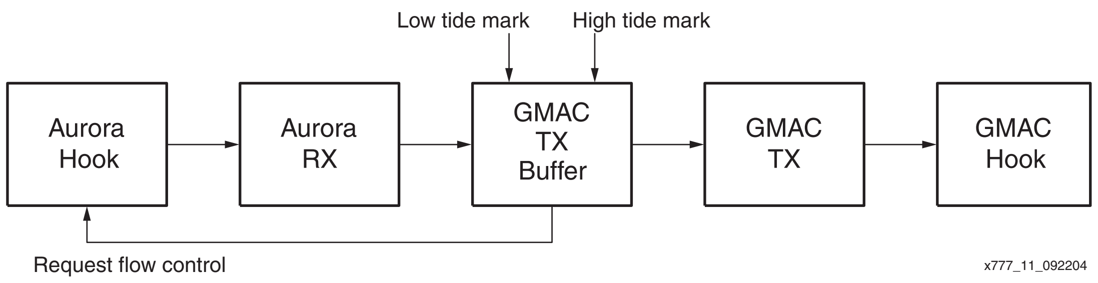


Figure 11: **Aurora Flow Control**

When the amount of bytes committed for transmission in the GMAC TX buffer is over the high-tide mark, the buffer requests that the Aurora hook send an NFC message telling the sender to stop sending indefinitely. The GMAC TX thread continues to send messages, which reduces the number of bytes committed for transmission in the GMAC TX buffer. Once the number of bytes passes below the low-tide mark, the GMAC TX buffer requests that the Aurora hook send an NFC message telling the sender to begin sending again (indefinitely). This process repeats if the number of bytes committed passes the high-tide mark.

Flow Control in Gigabit Ethernet

Flow control in Gigabit Ethernet is performed by means of pause frames. These pause frames take precedence over normal data frames, but do not interfere with sending the current frame if its transmission is already underway when a pause is requested. On reception, these pause frames are flagged as bad frames, and therefore do not interfere with normal operation, apart from throttling the flow of user data.

The Gigabit Ethernet reception needs to be slowed down when the buffer feeding the Aurora transmitter becomes nearly full, as shown in Figure 12.

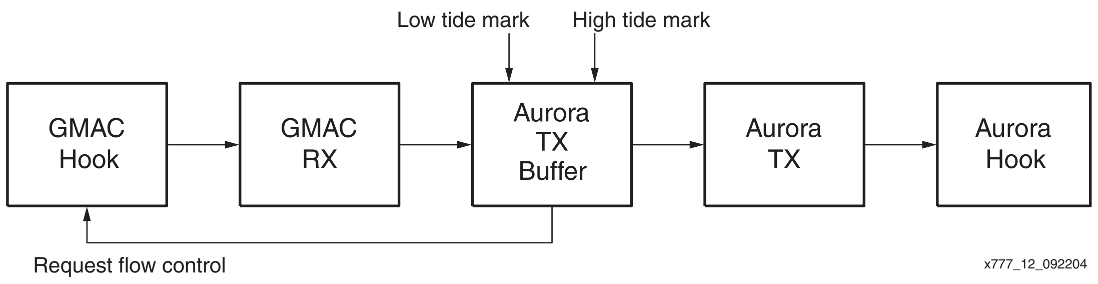


Figure 12: **GMAC Flow Control**

When the amount of bytes committed for transmission in the Aurora TX buffer is over the high-tide mark, the buffer requests that the GMAC hook begin sending pause frames. The Aurora TX thread continues to send messages, which reduces the number of bytes committed for transmission in the Aurora TX buffer. Once the number of bytes passes below the low-tide mark, the Aurora TX buffer ceases requesting the transmission of pause frames. This process repeats if the number of bytes committed passes the high-tide mark.

Device Utilization

The device utilization for an XC2VP7 Virtex-II Pro FPGA is shown in Table 6.

Table 6: **Device Utilization**

Resource Type	Number Used	Percent of XC2VP7 Resources
SLICEs	1612	32%
RAMBs	16	36%
DCMs	1	25%
GTs	2	25%
IOBs	1	1%

Modifying the Design

The design is useful as a starting point for designs using either Aurora or Gigabit Ethernet in store-and-forward systems. Considering initially Ethernet, a useful sub-system is shown in Figure 13.

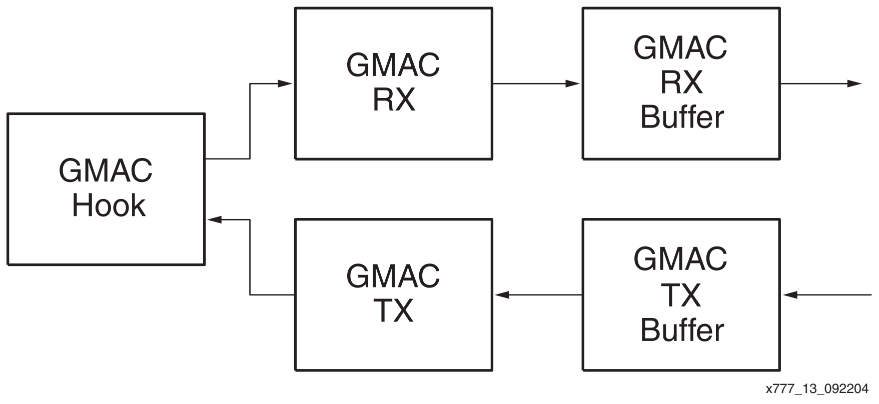


Figure 13: **Ethernet Sub-System**

The main interfacing signals on this system are shown in Table 7.

Table 7: **Ethernet Sub-system Interfacing Signals**

Signal	Description
GMAC_RX_BUFFER	
NonEmpty	GMAC_RX_BUFFER contains data
GetClk	Clock signal for the get interface
GetOffset	Offset of the current read data item within the packet
GetCommit	Commit the read of the current packet
GetData	16 bit read data item within the packet

Table 7: **Ethernet Sub-system Interfacing Signals *(Continued)***

Signal	Description
GMAC_TX_BUFFER	
PutClk	Clock signal for the put interface
PutOffset	Offset of the current write data item within the packet
PutWE	Write Enable for current data item
PutCommit	Commit the write of the current packet
PutData	16 bit write data for current item within the packet
GMAC_hook	
RXP, RXN	Differential receive pins on transceiver
TXP, TXN	Differential transmit pins on transceiver

This is a sub-system that could easily be connected to other logic. The FIFO logic is reasonably generic, though the commit mechanism is unique.

Refering to Figure 14, reads are performed by putting the offset of the current read data item on the *getOffset* port. On the next rising edge of the *getClk* signal, the data item at that offset within the current packet appears on the *getData* port.

Refering to Figure 15, writes are similarly performed by putting the offset of the current data item to write on the *putOffset* port, asserting the *putWE* signal. The data item on the *putData* port is stored on the next rising edge of *putClk*. *PutClk* and *getClk* are independent.

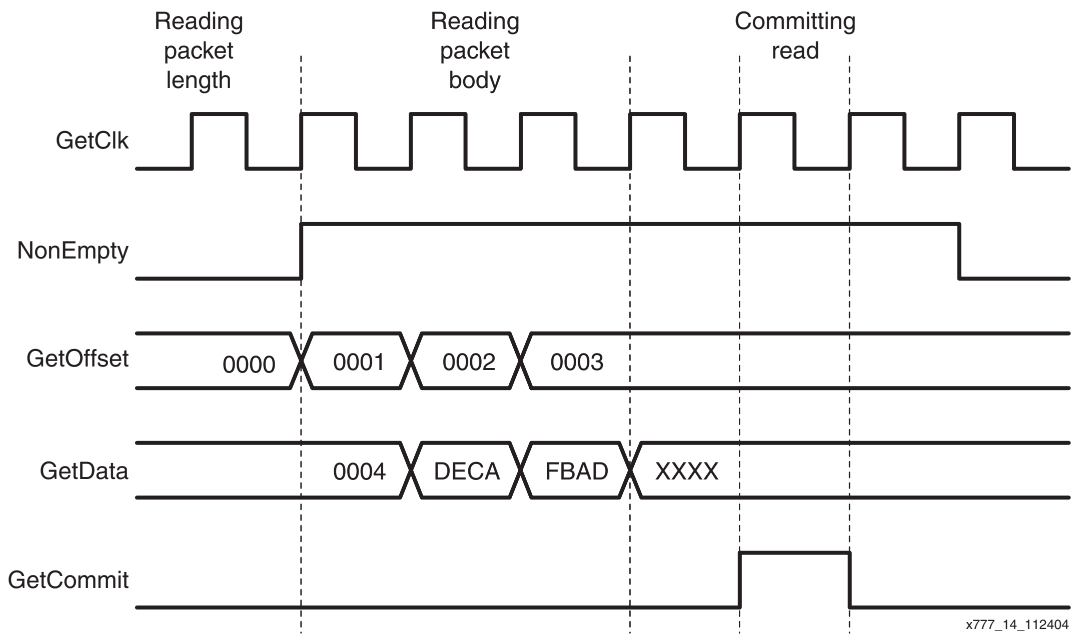


Figure 14: **Ethernet Read Mechanism**

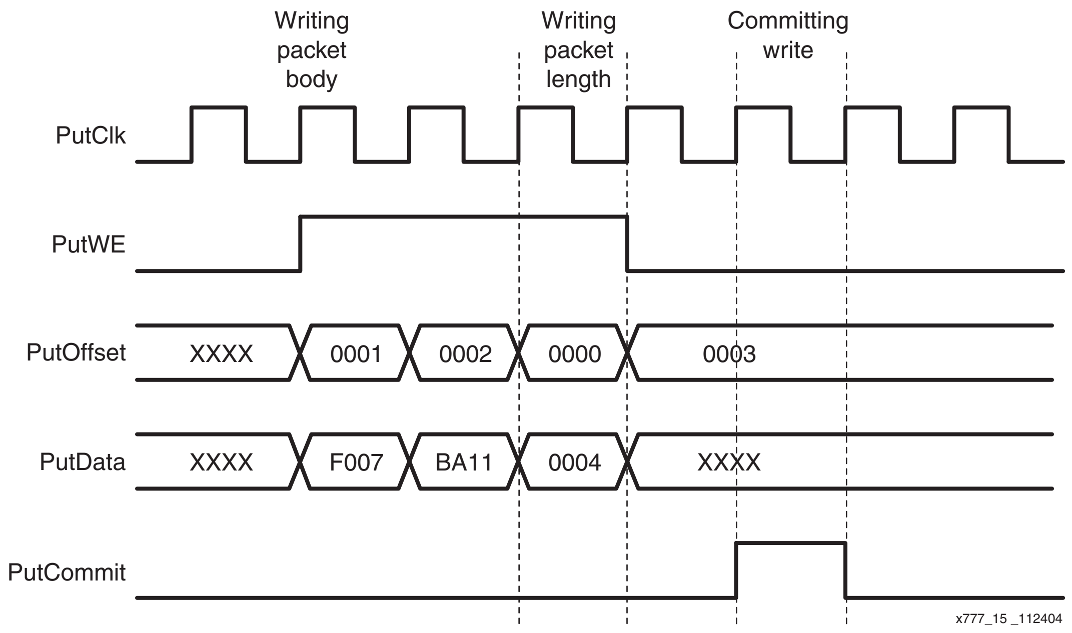


Figure 15: **Ethernet Write Mechanism**

A similar system could be put together for Aurora, as shown in Figure 16:

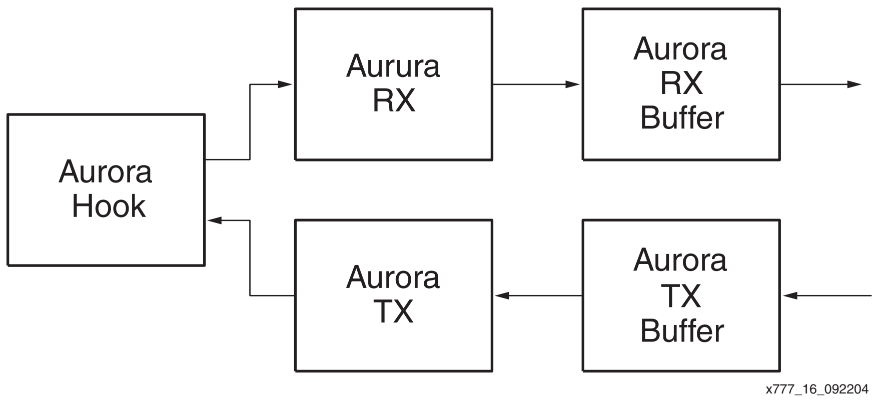


Figure 16: **Aurora Sub-System**

The main interfacing signals on this system are shown in Table 8.

Table 8: **Aurora Sub-system Interfacing Signals**

Signal	Description
Aurora_RX_BUFFER	
NonEmpty	GMAC_RX_BUFFER contains data
NearlyFull	GMAC_RX_BUFFER is nearly full (configurable depth)
GetClk	Clock signal for the get interface
GetOffset	Offset of the current read data item within the packet
GetCommit	Commit the read of the current packet
GetData	8 bit read data item within the packet

Table 8: **Aurora Sub-system Interfacing Signals *(Continued)***

Signal	Description
Aurora_TX_BUFFER	
PutClk	Clock signal for the put interface
PutOffset	Offset of the current write data item within the packet
PutWE	Write Enable for current data item
PutCommit	Commit the write of the current packet
PutData	8 bit write data for current item within the packet
Aurora_hook	
RXP, RXN	Differential receive pins on transceiver
TXP, TXN	Differential transmit pins on transceiver

Again, the FIFO logic is reasonably generic, notwithstanding the special packet commit mechanism, and could easily be incorporated in a larger system. The protocol for reading and writing data items is the same as that shown in Figure 14, apart from the data widths.

Using either of these two sub-systems, a designer could rapidly build either a total Aurora-based solution, or a total Gigabit Ethernet-based solution.

References

1. Tanenbaum, Andrew S: "Computer Networks 3rd Ed.", Prentice Hall, 1996.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/03/04	1.0	Initial Xilinx release.