



XAPP541 (v1.0) April 24, 2006

An Ethernet-to-MFRD Traffic Groomer

Author: Jack Lo

Summary

This application note describes the implementation of a traffic groomer that bridges the system space between a network line port (in this case, Gigabit Ethernet frame traffic) and the Mesh Fabric Reference Design (MFRD) [Ref 1]. The ingress side of the traffic groomer receives network frames from the Ethernet MAC, segments the frames into cells and prepends an MFRD-compatible header, schedules them on a per-port basis, and sends them into the MFRD. On egress, the traffic groomer receives cells from the MFRD, sorts them based on source fabric port, reassembles the cells into frames, and schedules them onto each line port.

Introduction

The traffic groomer is intended to be implemented on each line card in a network switch application (see Figure 1 and Figure 2). The traffic groomer consists of an Ingress and an Egress block (see Figure 3 and Figure 4).

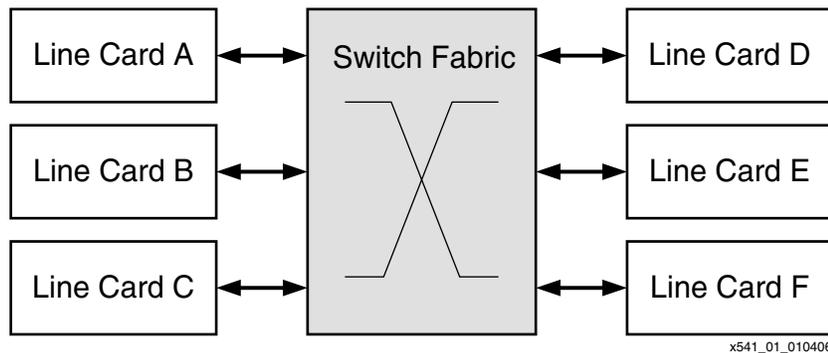


Figure 1: Overview of Network Topology

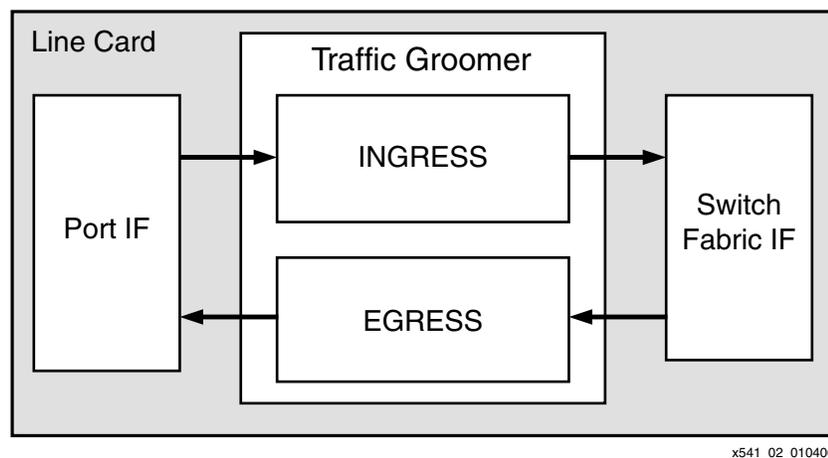
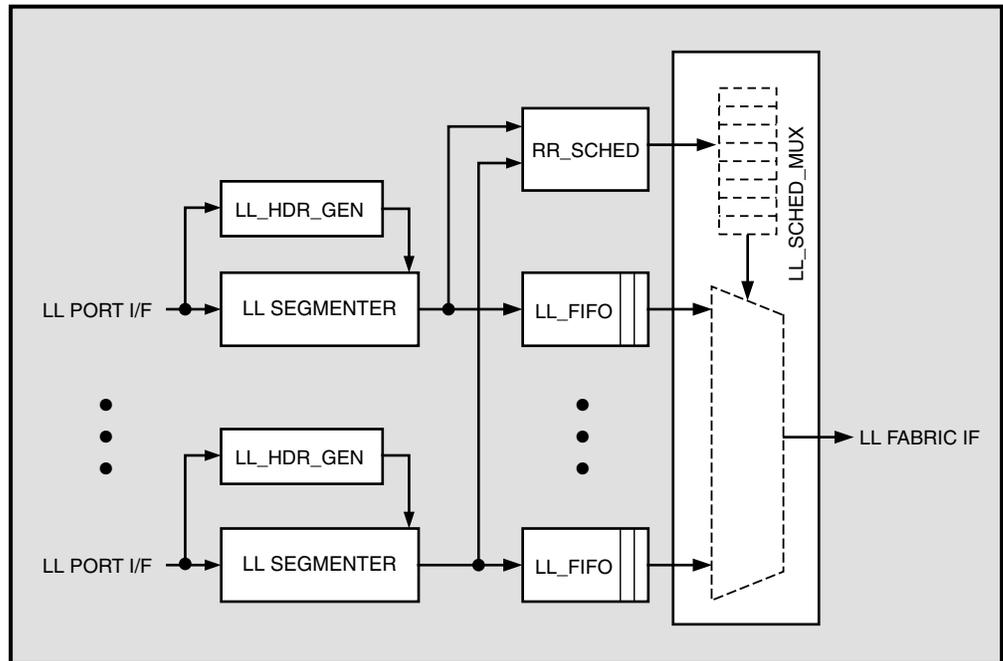


Figure 2: Overview of Traffic Groomer in Line Card

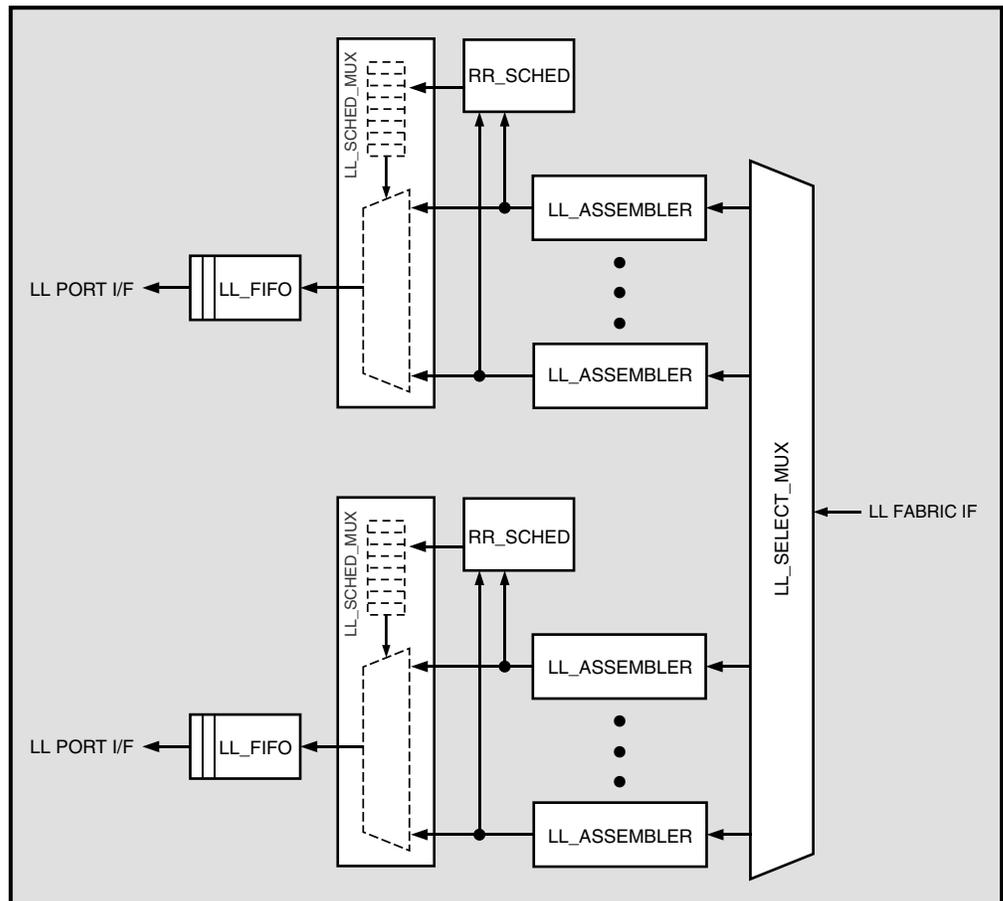
© 2006 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.



x541_03_011606

Figure 3: Ingress Block Diagram

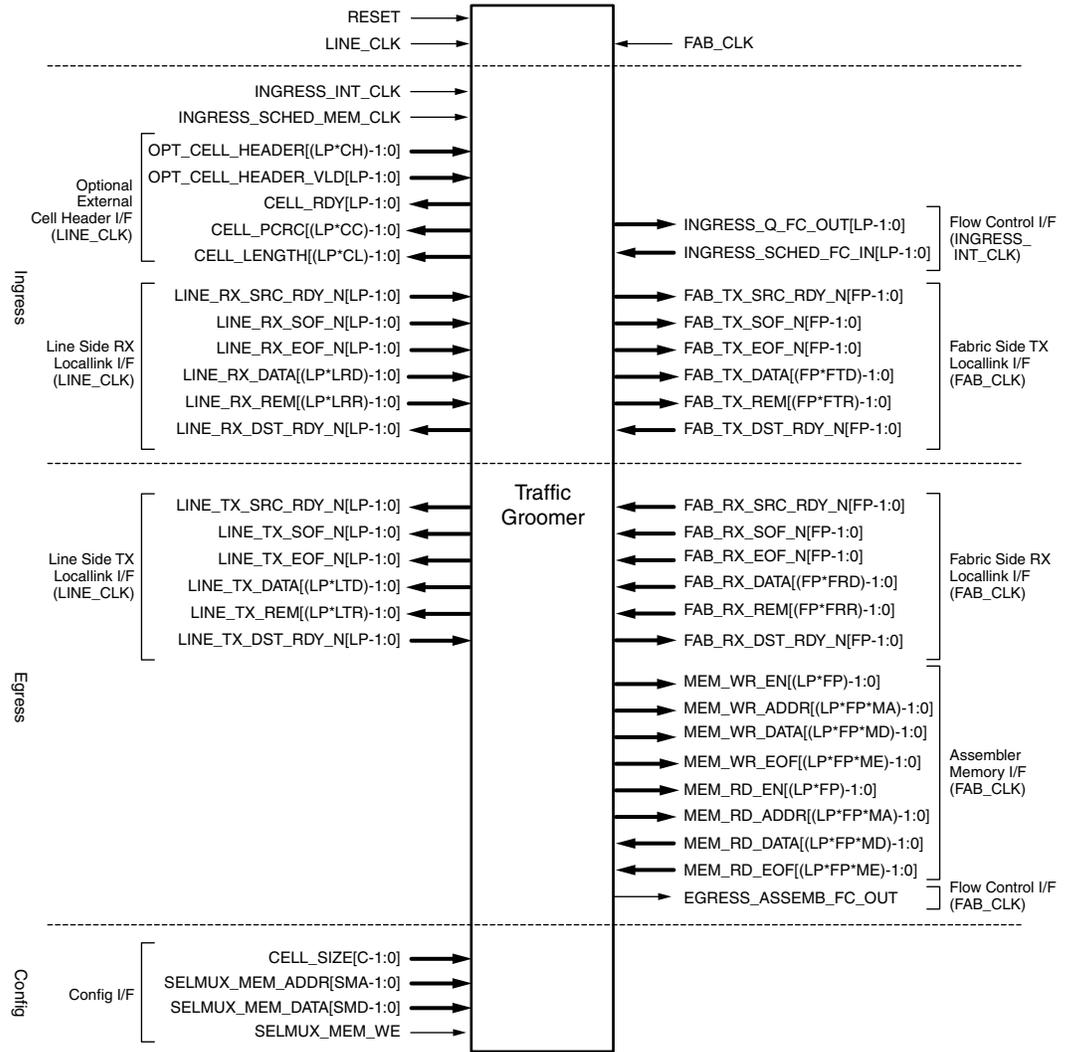


x541_04_011606

Figure 4: Egress Block Diagram

Interface

The input and output signals associated with the top-level traffic groomer block are illustrated in Figure 5. These signals are defined in Table 1.



x541_05_011606

Figure 5: TG_TOP, Top-Level Traffic Groomer Block

Table 1: TG_TOP Interface Signals

Signal Name	Width	I/O	Description
Global Interface Signals:			
RESET	1	Input	Asynchronous reset.
INGRESS_INT_CLK	1	Input	Clock for internal Ingress blocks: LocalLink Segmenter output to Ingress Queue input. Generally tied to FAB_CLK.
INGRESS_SCHED_MEM_CLK	1	Input	Clock for internal Ingress blocks: memory interface signals between LocalLink Scheduling MUX and Round-robin Scheduler. Generally tied to FAB_CLK.
Line Port Side Interface Signals:			Line port-side signals are indexed per line port, with signals in an array being independent of one another. For signals that are LP wide, bit[0] maps to line port 0, where LP is the number of line ports. For signals that are LP*N wide, bits [(N-1):0] map to line port 0, where N is the width of the signal vector.
LINE_CLK	1	Input	Clock for line port side interface signals
OPT_CELL_HEADER	LP*CH	Input	External cell header that is CH bits wide, where CH = 32 or 64. [optional]
OPT_CELL_HEADER_VLD	LP	Input	External cell header valid signal that indicates optional external cell header data is valid. [optional]
CELL_RDY	LP	Output	Indicates a full cell has been received or EOF is encountered. [optional]
CELL_PCRC	LP*CC	Output	Indicates the payload CRC for the given cell. This includes the segmentation and reassembly (SAR) header as part of the payload. The width of CC is generally 8 bits. This is valid when CELL_RDY is asserted. [optional]
CELL_LENGTH	CL	Output	Indicates the length of the current cell including the SAR header but not the cell header. The value is 0-based, meaning 0 is 1 byte and 63 is 64 bytes. This is valid when CELL_RDY is asserted. CL is the maximum width and depends on the maximum cell size. [optional]
LINE_RX_DST_RDY_N	LP	Output	Indicates to the line port RX side that data can be accepted on the subrange of LINE_RX_DATA bus in the current cycle. (Active Low)
LINE_RX_SRC_RDY_N	LP	Input	Indicates the subrange of LINE_RX_DATA is valid during the current cycle. (Active Low)
LINE_RX_SOF_N	LP	Input	Indicates the beginning of a frame transfer on the subrange of LINE_RX_DATA bus. (Active Low)
LINE_RX_EOF_N	LP	Input	Indicates the end of a frame transfer on the subrange of LINE_RX_DATA bus. (Active Low)
LINE_RX_DATA	LP*LRD	Input	Data input from the line port RX side with width LRD per line port. LRD can be 8, 16, or 32.
LINE_RX_REM	LP*LRR	Input	Remainder output from the line port RX side. Indicates the number of valid bytes on the subrange of the LINE_RX_DATA bus when LINE_RX_EOF_N is asserted. LINE_RX_REM is binary-encoded with width LRR. LRR can be 1 or 2.
LINE_TX_DST_RDY_N	LP	Input	Indicates to the line port TX side that data can be accepted on the subrange of LINE_TX_DATA bus in the current cycle. (Active Low)
LINE_TX_SRC_RDY_N	LP	Output	Indicates the subrange of LINE_TX_DATA is valid during the current cycle. (Active Low)
LINE_TX_SOF_N	LP	Output	Indicates the beginning of a frame transfer on the subrange of LINE_TX_DATA bus. (Active Low)
LINE_TX_EOF_N	LP	Output	Indicates the end of a frame transfer on the subrange of LINE_TX_DATA bus. (Active Low)
LINE_TX_DATA	LP*LTD	Output	Data input from the line port TX side with width LTD per line port. LTD can be 8, 16, or 32.

Table 1: TG_TOP Interface Signals (Continued)

Signal Name	Width	I/O	Description
LINE_TX_REM	LP*LTR	Output	Remainder output from the line port TX side. Indicates the number of valid bytes on the subrange of the LINE_TX_DATA bus when LINE_TX_EOF_N is asserted. LINE_TX_REM is binary-encoded with width LTR. LTR can be 1 or 2.
INGRESS_Q_FC_OUT	LP	Output	Status signal indicating that the Ingress Queue is full. LocalLink Segmenter automatically throttles Line port side RX when Ingress Queue becomes full.
INGRESS_SCHED_FC_IN	LP	Input	Indicates to the Round-Robin Scheduler to finish scheduling the current cell and then stop scheduling until signal is deasserted.
Fabric Port Side Interface Signals:			Fabric port-side signals are indexed per fabric port, with signals in an array being independent of one another. For signals that are FP wide, bit[0] maps to fabric port 0, where FP is the number of fabric ports. For signals that are FP*N wide, bits [(N-1):0] map to fabric port 0, where N is the width of the signal vector.
FAB_CLK	1	Input	Clock for fabric port side interface signals.
FAB_TX_DST_RDY_N	FP	Input	Indicates to the fabric port TX side that data can be accepted on the subrange of FAB_TX_DATA bus in the current cycle. (Active Low)
FAB_TX_SRC_RDY_N	FP	Output	Indicates the subrange of FAB_TX_DATA is valid during the current cycle. (Active Low)
FAB_TX_SOF_N	FP	Output	Indicates the beginning of a frame transfer on the subrange of FAB_TX_DATA bus. (Active Low)
FAB_TX_EOF_N	FP	Output	Indicates the end of a frame transfer on the subrange of FAB_TX_DATA bus. (Active Low)
FAB_TX_DATA	FP*FTD	Output	Data input from the fabric port TX side with width FTD per fabric port. FTD can be 8, 16, or 32.
FAB_TX_REM	FP*FTR	Output	Remainder output from the fabric port TX side. Indicates the number of valid bytes on the subrange of the FAB_TX_DATA bus when FAB_TX_EOF_N is asserted. FAB_TX_REM is binary encoded with width FTR. FTR can be 1 or 2.
FAB_RX_DST_RDY_N	FP	Output	Indicates to the fabric port RX side that data can be accepted on the subrange of FAB_RX_DATA bus in the current cycle. (Active Low)
FAB_RX_SRC_RDY_N	FP	Input	Indicates the subrange of FAB_RX_DATA is valid during the current cycle. (Active Low)
FAB_RX_SOF_N	FP	Input	Indicates the beginning of a frame transfer on the subrange of FAB_RX_DATA bus. (Active Low)
FAB_RX_EOF_N	FP	Input	Indicates the end of a frame transfer on the subrange of FAB_RX_DATA bus. (Active Low)
FAB_RX_DATA	FP*FRD	Input	Data input from the fabric port RX side with width FRD per fabric port. FRD can be 8, 16, or 32.
FAB_RX_REM	FP*FRR	Input	Remainder output from the fabric port RX side. Indicates the number of valid bytes on the subrange of the FAB_RX_DATA bus when FAB_RX_EOF_N is asserted. FAB_RX_REM is binary encoded with width FRR. FRR can be 1 or 2.
EGRESS_ASSEMB_FC_OUT	LP	Output	Status signal indicating that the Egress queue in the LocalLink Assembler is full.
Assembler Memory Interface Signals:			Assembler memory signals are indexed per assembler queue, which is equal to the number of line ports X the number of fabric ports. Signals in the array are independent of one another. For signals that are LP*FP wide, bit[0] maps to assembly queue 0, where LP is the number of line ports and FP is the number of fabric ports. For signals that are LP*FP*N wide, bits [(N-1):0] map to assembly queue 0, where N is the width of the signal vector.
MEM_WR_EN	LP*FP	Output	Assembler memory write enable signal.
MEM_WR_ADDR	LP*FP*MA	Output	Assembler memory write address signal with width MA per assembler queue.

Table 1: TG_TOP Interface Signals (Continued)

Signal Name	Width	I/O	Description
MEM_WR_DATA	LP*FP*MD	Output	Assembler memory write data signal with width MD per assembler queue.
MEM_WR_EOF	LP*FP*ME	Output	Assembler memory write EOF marker with width ME. ME is MEM_WR_DATA/8.
MEM_RD_EN	LP*FP	Output	Assembler memory read enable signal.
MEM_RD_ADDR	LP*FP*MA	Output	Assembler memory read address signal with width MA per assembler queue.
MEM_RD_DATA	LP*FP*MD	Input	Assembler memory read data signal with width MD per assembler queue.
MEM_RD_EOF	LP*FP*ME	Input	Assembler memory read EOF marker with width ME. ME is MEM_RD_DATA/8.
Configuration Interface Signals:			
CELL_SIZE	C	Input	Segmented cell size. Cell size is 1-based, so a 64-byte cell has a CELL_SIZE = 64.
SELMUX_MEM_ADDR	SMA	Input	LocalLink Select MUX Address signals. Used to reconfigure Select MUX CAM designed around SRL16s. See CAM CORE Generator™ (CoreGen) documentation.
SELMUX_MEM_DATA	SMD	Input	LocalLink Select MUX Data signals. Used to reconfigure Select MUX CAM designed around SRL16s. See CAM CoreGen documentation.
SELMUX_MEM_WE	1	Input	LocalLink Select MUX write enable signal. Used to reconfigure Select MUX CAM designed around SRL16s. See CAM CoreGen documentation.

Hardware Implementation

Pattern Generator

The Pattern Generator block is used in simulation and testing. It consists of a Block RAM wrapper that allows access to user-initialized Block RAMs via a LocalLink interface. Based on the state of a set of test input switches, the Pattern Generator will either be enabled or disabled. When enabled, the Pattern Generator will output the pre-initialized packet data via LocalLink to the target core, reading the entire set of Block RAMs in a continuous loop. Standard LocalLink control can be used to throttle the packet data. The user can customize the packet data stored in the Block RAMs to facilitate various test profiles. Data is read from the Block RAM one clock at a time, which allows for fine granularity control to test such things as interframe GAP and truncated packets.

LocalLink Segmenter

The basic operation of the Segmenter block is to parse the incoming variable-size data frame into fixed-size cells. A read counter keeps track of the number of bytes read during each clock cycle. When the counter reaches the maximum cell size, an *end of cell* is signaled and a *start of cell* is issued on the next clock cycle. When an *end of frame* is encountered, the read counter also stops and signals an *end of cell*. The Segmenter leverages the built-in data delineation of LocalLink by using a LL_FIFO to store the data cells. Cell boundaries are indicated by SOF/EOF LocalLink signals to the LL_FIFO.

At the same time, cell headers and SAR headers are read and stored in the cell and SAR header FIFOs respectively. These FIFOs store the full cell and SAR headers as passed in by a Header Generation block. Please see “[LocalLink Header Generation Block](#)” functional description. A unique cell header and SAR header is stored for every cell in a frame. Finally, the number of data bytes in the cell are stored in an RX byte FIFO so that final cell size and TX clock counts can be derived and used to control the output MUX.

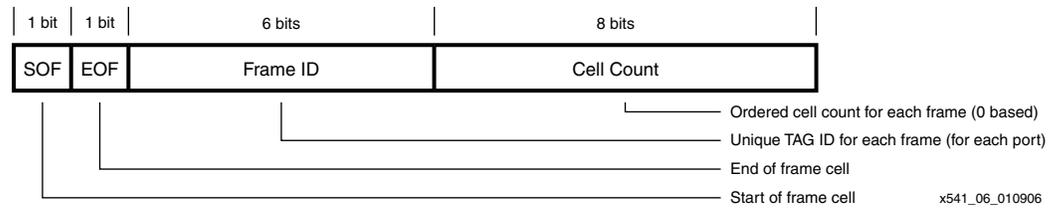


Figure 6: LocalLink Segmenter SAR Header Example

LocalLink Header Generation Block

The LocalLink Header Generation block is responsible for generating the cell and SAR header for each outgoing cell and outputting that data to the LocalLink segmenter block. The basic functions involve counting the number of incoming bytes until either a full cell is counted or an EOF occurs. The cell header and SAR header are then sent to the segmenter block for insertion. This block supports CRC calculation and cell padding (zeroes out the remainder of the cell). Cell length is also calculated and can be inserted into the cell header as required.

An optional external cell header mode allows for customization beyond the built-in crossbar/MFRD multicast cell header mode. When the configuration parameter EXT_CELL_HDR is set to TRUE, this block responds by passing through the optional CELL_HEADER and CELL_HEADER_VLD instead of generating one internally. This block provides a cell ready signal, along with the cell length and cell PCRC value if the user wishes to insert these into their customized cell header.

Round-Robin Scheduler

The round-robin scheduler receives cell records with a specified cell length, cell EOF, and cell error from the input cell queues (either LL_SEGMENTER or LL_ASSEMBLER) and stores them. The scheduler keeps track of this data and schedules out entire frames per port, one at a time. A frame can consist of one or more cells. If no cells are available, either nothing is written to the Scheduling MUX memory, or a null value is written, depending on compilation parameters.

LocalLink Scheduling MUX

This block writes data directly to the scheduler MUX memory. When the memory is read out, the values are decoded to control DST_RDY signals for each input LocalLink port. Input LocalLink signals are then MUXed to a single output LocalLink port. By default, the current design supports up to 2048 time slots for 128 ports. This is a function of the instantiated memory width and depth, and the maximum number of ports is $2^{(\text{memory width}-1)}$. The maximum number of time slots is essentially the memory depth of the input ports.

It is the responsibility of the scheduler to keep the scheduler MUX memory filled with valid data and not overwrite the entries until the MUX has read it. This mainly applies if the write clock runs faster than the read clock and involves responding to the MEM_READY signal. Overwrite is prevented by the CoreGen FIFO block. It stops reading data if no new data is written and the FIFO is empty. When the FIFO is completely empty, the MUX selects none of the input ports. A valid use model is for the scheduler to write null time slices (time slices where none of the input ports is selected), but the design is capable of not selecting any ports when no data is read.

PROG_FULL is set to assert when FIFO can only write one more cell. It deasserts when the FIFO can write two cells. This can be modified in the sched_mux_fifo.vhd files if necessary.

LocalLink Queue

This block is basically a wrapper for a LocalLink FIFO with support for flow control.

LocalLink Select MUX

This block reads the incoming cells (from the fabric) and sorts them by their cell headers. It first stores the LocalLink signals in a shift register while registering data bits in predefined cell header positions. These registers are passed to a lookup CAM, which returns the designated output queue or queues for the given cell. The sort operation can factor in up to two sorting bit ranges, with an optional MCAST bit. For example, in the MFRD case, sorting can be done on the 8-bit source header, the 12-bit destination header, and the MCAST bit. All output LocalLink signals are shared by all the queues with a unique SRC_RDY signal for each queue to enable actual data to be passed.

When MCAST_ENABLED is FALSE, the sort is based on both the lookup value, a concatenation of the sort-1 and sort-2 bits. The user can opt to use only sort-1 or only sort-2 by setting the width of the unused sort to 0. When MCAST_ENABLED is TRUE, the presence of the MCAST bit is checked. When the MCAST bit is asserted, sort-2 bits are masked in the CAM lookup operation. As an example of this, source ports could be used in sort-1 and destination ports in sort-2. When MCAST is asserted, the destination is masked, and only the source port is matched. All destination ports are considered a match in this case.

The assignment of the output ports to the matching sort-1 and sort-2 values can be customized in the CAM initialization table (.mif file) and updated during run-time through the configuration ports.

LocalLink Assembler

The Assembler basically takes incoming cells, checks for continuity by examining the SAR headers, and assembles them in a continuous stream into local memory (local memory accessed through the memory interface). This is done by shifting the incoming data to match previous cells until all cells in a frame are assembled. Errors cause the write address to roll back to the last good value. Reads begin when the read counter and the valid write counter have a difference greater than 2 (to accommodate timing). The valid write counter increments, whenever valid cells are received for an entire frame.

Ethernet to LocalLink Shim

This block converts ethernet signals to LocalLink signals. Very similar to converting between `data_valid` and `sof/eof` formats. The only additional feature is limited response to DST_RDY. Here, the TX ethernet side deasserts the RX LocalLink DST_RDY when waiting for TX ethernet acknowledge. If a response to TX LOCALLINK DST_RDY is required, then RX Ethernet Pause is asserted when TX LOCALLINK DST_RDY is asserted.

DCR Interface Block

This block interfaces with a DCR bus to write and read to local configuration registers. The processor can reconfigure the CELL_SIZE configuration values through this DCR compatible interface.

Compilation Parameters

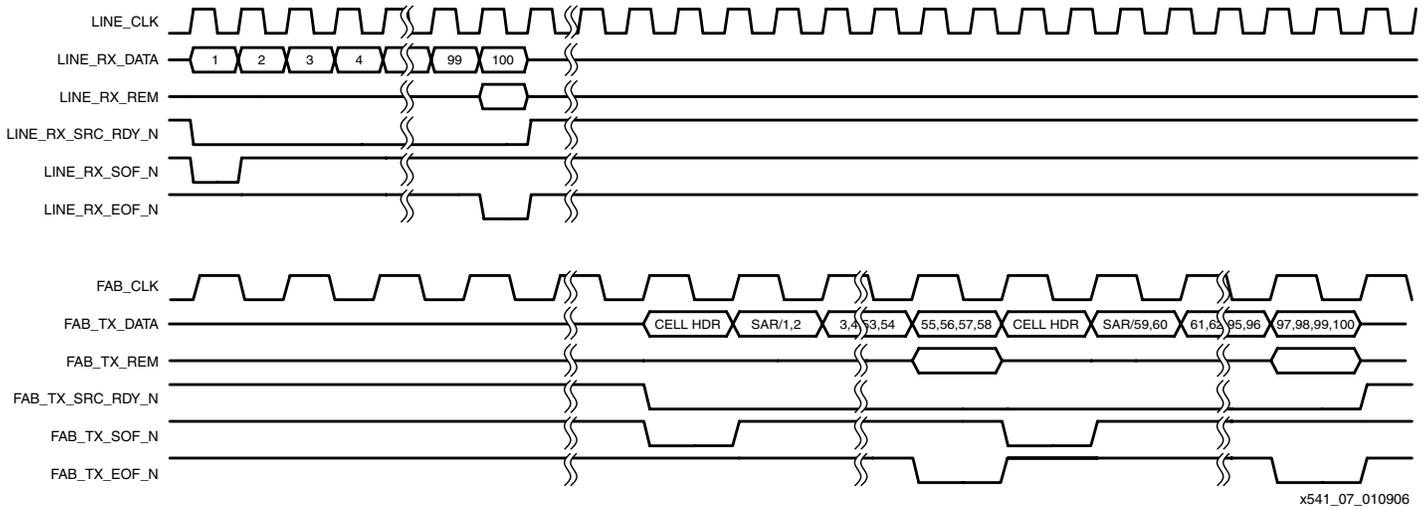
[Table 2](#) lists the traffic groomer parameterization values and their descriptions. These value are adjusted in the `groomer_pack.vhd` file and must be coupled with appropriate change in the local `switch_scale_pack_single.vhd` file in order to connect to the MFRD correctly.

Table 2: Compilation Parameters

Name	Values	Default Values	Description
PATGEN_ENABLED	Boolean: TRUE, FALSE	TRUE	Enable pattern generator block and MUX.
TOP_MGTCLKGEN_ENABLED	Boolean: TRUE, FALSE	TRUE	Enable MGT clock generator on the top of the device.
BOT_MGTCLKGEN_ENABLED	Boolean: TRUE, FALSE	TRUE	Enable MGT clock generator on the bottom of the device.
NUM_FAB_PORTS	Integer: 2 to 256	4	Number of fabric ports including this one. Number of MGTs is NUM_FAB_PORTS – 1.
NUM_LINE_PORTS	Integer: 1 to 256	4	Number of line ports (e.g., Ethernet) per line card.
SAR_HDR_WIDTH	Integer: 8, 16, 24, or 32	16	Width of SAR header.
SAR_FRAME_ID_WIDTH	Integer: 1 to (SAR_HDR_WIDTH – 3)	6	Width of Frame ID field in SAR header. SAR cell count width is equal to SAR_HDR_WIDTH – SAR_FRAME_ID_WIDTH – 2.

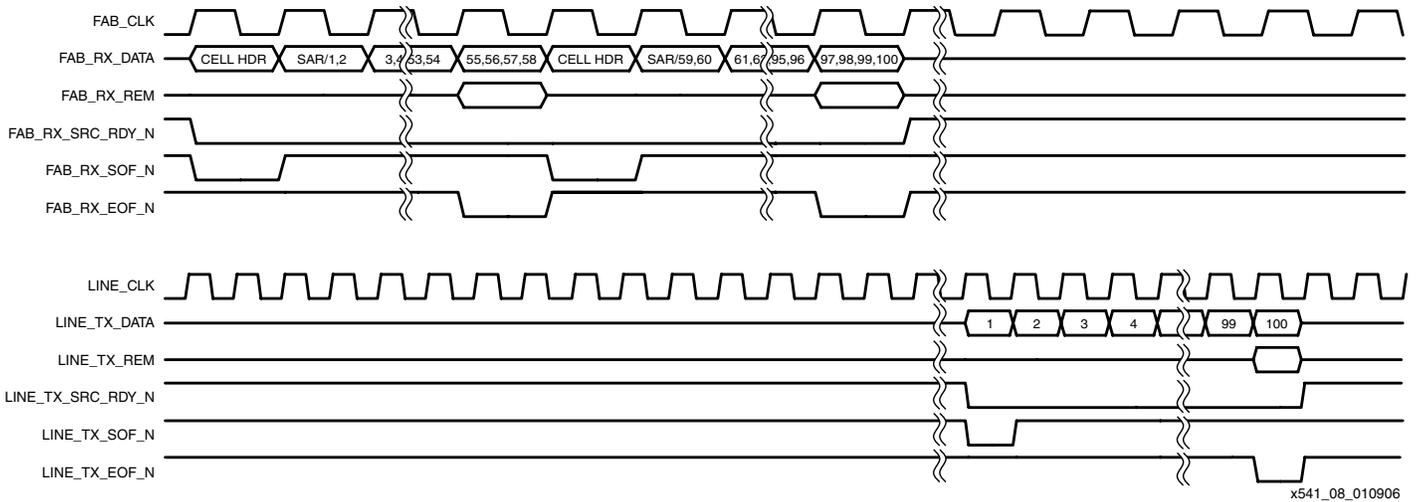
Waveforms

Top-level waveforms are shown in Figure 7 and Figure 8.



x541_07_010906

Figure 7: Ingress Waveform



x541_08_010906

Figure 8: Egress Waveform

Verification

Simulation

Figure 9 shows the simulation testbench used to verify the traffic groomer, Ethernet MAC, pattern generator, and MFRD.

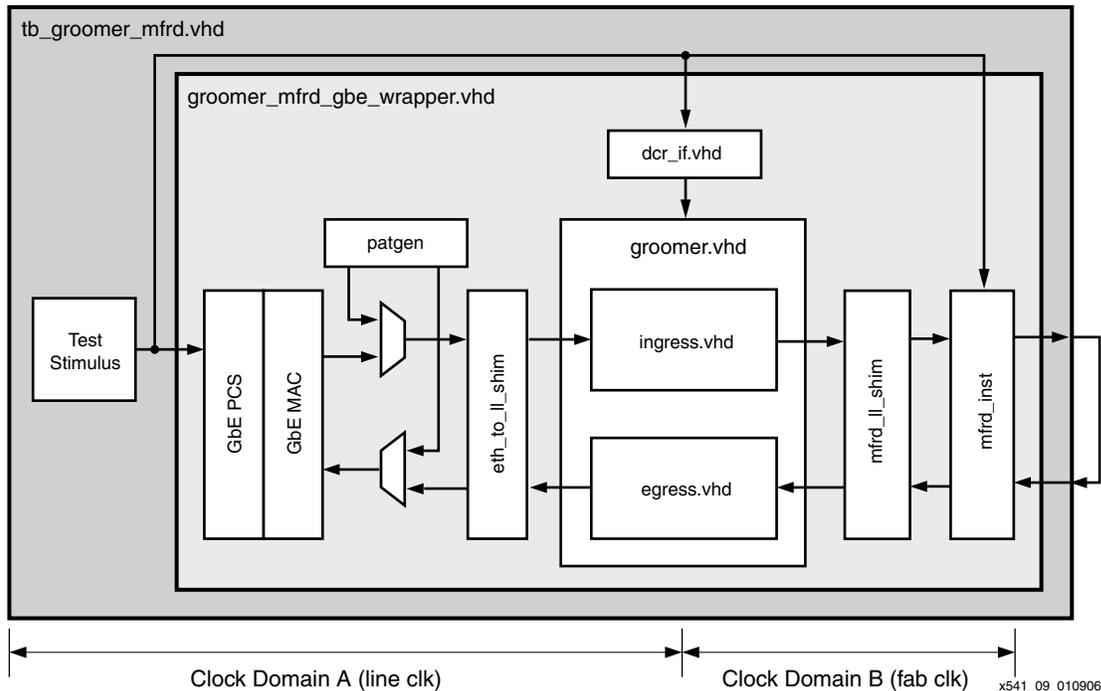


Figure 9: Simulation and Implementation Testbench

Hardware Testing

Figure 10 shows the hardware implementation used to verify the traffic groomer, Ethernet MAC, pattern generator, and MFRD. The reference design was implemented and tested on an ML321 and an ML310 board using the included Virtex™-II Pro device.

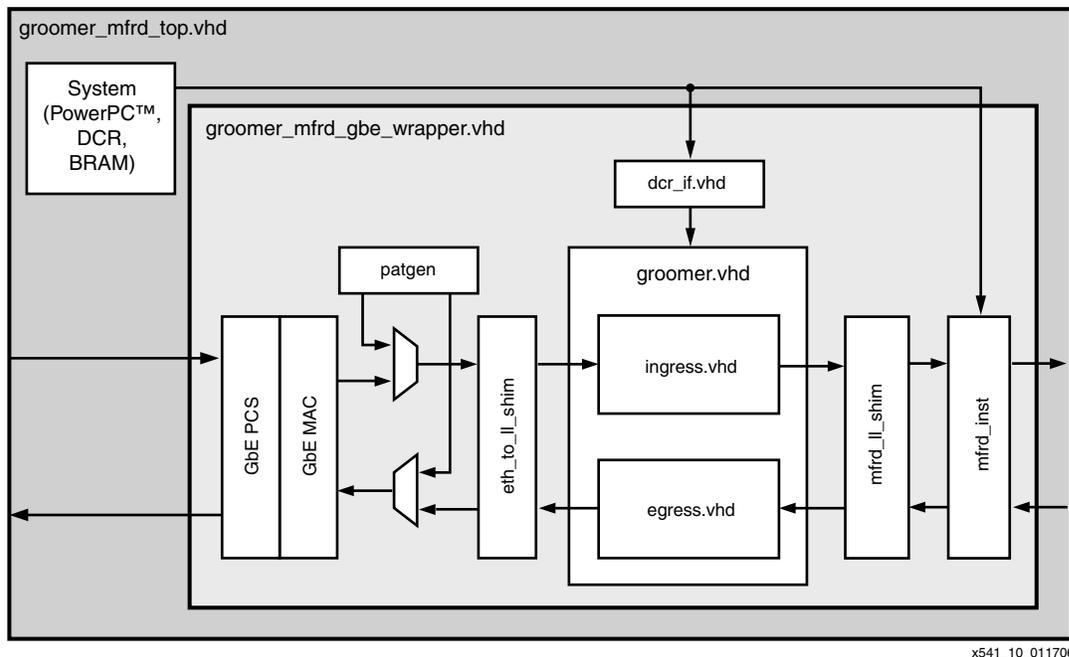


Figure 10: Top-Level Hardware Implementation

Porting to Virtex-4 Devices

Porting the design to Virtex-4 devices would require some changes:

1. All CoreGen components would need to be regenerated to target a Virtex-4 device.
2. The Gigabit Ethernet MAC soft core would still be used, because the embedded Virtex-4 hard core is not currently supported. (The design should not have a problem interfacing with the Virtex-4 hard core, but the reference design would require source code modifications to accommodate it.)

All other design elements should function properly in a Virtex-4 design.

Reference Design

The Ethernet-to-MFRD traffic groomer reference design (VHDL files) is available on the Xilinx website:

<http://www.xilinx.com/bvdocs/apnotes/xapp541.zip>

Simulation test benches and scripts are provided.

Within the traffic groomer design, the LocalLink FIFO reference design is used and does not need to be separately downloaded. More information on this design is available on the Xilinx website.

<http://www.xilinx.com/bvdocs/apnotes/xapp691.zip>

The traffic groomer source files can be found off the main installation directory. Specific design examples, such as the Ethernet-to-MFRD reference design, can be found under `<examples>`.

Table 3 lists the VHDL source file names located under the `<src/vhdl>` directory. **Table 4** lists the file names and descriptions for the simulation testbench found in the `<test/func_sim/vhdl>` and `<test/testbench/vhdl>` directories. **Table 5** lists the implementation scripts and project files found under the `<build>` directory. Specific implementation examples, such as the Ethernet-to-MFRD reference design, can be found under `<examples/*/build>`. **Table 6** lists the pattern generator source files, which are used in various example implementations to generate network traffic.

Table 3: Traffic Groomer – VHDL Source Files

Name	Description
ll_segmenter.vhd	LocalLink segmenter block. (ingress)
ll_hdr_gen.vhd	LocalLink header generation block. (ingress)
segmenter_fifos.vhd	FIFO wrappers for the ll_segmenter block. Generally wraps CoreGen FIFOs but can be customized further. (ingress)
ll_queue.vhd	LocalLink queue. (ingress/ egress)
rr_sched.vhd	Round-robin scheduler block. (ingress/ egress)
ll_sched_mux.vhd	LocalLink scheduler MUX block. (ingress/ egress)
sched_mux_fifo.vhd	FIFO wrapper for the ll_sched_mux block. Generally wraps CoreGen FIFOs but can be customized further. (ingress/ egress)
ingress.vhd	Ingress block wrapper.
ll_select_mux.vhd	LocalLink select MUX block. (egress)
lookup_cam.vhd	CAM wrapper for the ll_select_mux block. Generally wraps CoreGen CAMs but can be customized further. (egress)
ll_assembler.vhd	LocalLink assembler block. (egress)
egress.vhd	Egress block wrapper.
groomer_pack.vhd	Traffic groomer package file. Modify to change groomer design. Contains global functions as well.

Table 3: Traffic Groomer – VHDL Source Files (Continued)

Name	Description
groomer.vhd	Traffic groomer block wrapper.
<LL_FIFO>	LocalLink FIFO source files in LL_FIFO subdirectory. See LocalLink FIFO documentation for further information.
eth_ll_shim.vhd	Ethernet to LocalLink shim block. (optional)
ext_cell_hdr_gen.vhd	External cell header generation block (optional)

Table 4: Traffic Groomer – Simulation Testbench Files

Name	Description
tb_ingress.vhd	Ingress block testbench.
tb_egress.vhd	Egress block testbench.
tb_groomer.vhd	Top-level testbench to include Ethernet MACs, pattern generator, and MFRD.
debug_io.vhd	Debug I/O reference file for testbench.
types.vhd	Types reference file to support debug_io.vhd.
conversions.vhd	Conversions reference file to support debug_io.vhd.
functions.vhd	Function reference file to support debug_io.vhd.
fileread_tester.vhd	Test block that reads test vectors from a file.
readmem.vhd	File read functions to be used in fileread_tester.vhd.
*.vhd	Rest of VHD files are CoreGen wrappers.
run_tb_ingress.do	Modelsim DO script to test Ingress block.
run_tb_egress.do	ModelSim DO script to test Egress block.
run_tb_groomer.do	ModelSim DO script to test traffic groomer block.
mklibs.do	ModelSim DO script to set up libraries. Called by run scripts.
fifo_src_files.do	ModelSim DO script to compile LocalLink FIFO source files.
tb_ingress.vec	Ingress test vectors. (Ethernet frames)
tb_egress.vec	Egress test vectors. (MFRD cells)
wave_tb_ingress.do	ModelSim DO script to setup Ingress waveforms.
wave_tb_egress.do	ModelSim DO script to setup Egress waveforms.
wave_tb_groomer.do	ModelSim DO script to setup traffic groomer.
mklibs.do	ModelSim DO script to set up libraries. Called by run scripts.
fifo_src_files.do	ModelSim DO script to compile LocalLink FIFO source files.
clean_all	Unix script to clean modelsim generated files.
modelsim_sol.ini	ModelSim initialization file for UNIX environment (Solaris).
lookup_cam_20x16_srl.mif	Lookup CAM initialization file for 20 X 16 SRL CAM (MFRD).
lookup_cam_8x16_srl.mif	Lookup CAM initialization file for 8 X 16 SRL CAM (Aurora/Crossbar).

Table 5: Traffic Groomer – Scripts and Files

Name	Description
build.pl	PERL script to implement the design.
README	README file to describe necessary file changes to retarget and reimplement the design.
<ucf>	Implementation constraints files for ML310 and ML323 boards.
<syn>	Synthesis project files.
<edk>	EDK portion of reference design. Use this to build processor software, block RAM initialization, and processor submodule synthesis.
<imp>	Implementation project files.

Table 6: Traffic Groomer – Pattern Generator

Name	Description
<examples/patgen/src/vhdl>/pattern_generator.vhd	Pattern generator block.
<examples/patgen/src/vhdl>/*.vec	Pattern generator vectors source files.
<examples/patgen/src/vhdl>/gen_pattern_generator.pl	Conversion script to create pattern generator block from vector source files.

Ethernet-to-MFRD Design Example

The Ethernet-to-MFRD design example bridges the gap between an Ethernet MAC and the MFRD. The MFRD must first be downloaded and referenced for this design to work. The MFRD source (in VHDL) is available on the Xilinx website:

http://www.xilinx.com/esp/networks_telecom/optical/xlnx_net/mfrd/mfrd_source_code.zip

MFRD Simulation test benches and scripts are provided. The source files for the Ethernet-to-MFRD design example can be found under <examples/mfrd>. All subsequent table references assume this directory location as the base directory.

Table 7 lists the VHDL source file names located under the <src/vhdl> directory. **Table 8** lists the file names and descriptions for the simulation testbench found in the <test/func_sim/vhdl> and <test/testbench/vhdl> directories. **Table 9** lists the implementation scripts and projects files found under the <build> directory.

Table 7: Ethernet-to-MFRD Design Example - VHDL Source Files

Name	Description
dcr_if.vhd	DCR Interface block.
mfrd_ll_if.vhd	MFRD to LocalLink interface block.
switch_scale_pack_single.vhd	MFRD scale file to parameterize MFRD design to match traffic groomer parameterization.
mesh_switch_nods.vhd	MFRD wrapper file to redefine top-level generics for unified implementation.
groomer_pack.vhd	Traffic groomer package file. Locally edited to customize groomer for MFRD interface.
groomer_gbe_mfrd_wrapper.vhd	Top-level wrapper for groomer, MFRD modules, GbE PCS and MAC modules, and pattern generator.
groomer_mfrd_top.vhd	Top-level wrapper for groomer-MFRD and EDK subsystem design file.

Table 8: Ethernet-to-MFRD Design Example – Simulation Testbench Files

Name	Description
tb_groomer_mfrd.vhd	Top-level testbench for groomer_gbe_mfrd_wrapper.vhd.
bb_gbe_pcs_v_4_0_txbuf.vhd	Black box Ethernet PCS block.
bb_gbe_mac_v_4_0.vhd	Black box Ethernet MAC block.
bb_mesh_switch_nods.vhd	Black box MFRD block to speed up simulation.
run_tb_groomer_mfrd.do	ModelSim DO script to test traffic groomer plus MFRD.
run_tb_groomer_bb_mfrd.do	ModelSim DO script to test traffic groomer plus black-box MFRD.
mklibs.do	ModelSim DO script to set up additional Aurora specific libraries. Called by run scripts.
fifo_src_files.do	ModelSim DO script to compile LocalLink FIFO source files.
mfrd_src_files.do	ModelSim DO script to compile MFRD source files.
bb_mfrd_src_files.do	ModelSim DO script to compile black-box MFRD source files.
wave_tb_groomer_mfrd.do	ModelSim DO script to setup traffic groomer plus MFRD.
wave_tb_groomer_bb_mfrd.do	ModelSim DO script to setup traffic groomer plus black-box MFRD.
mklibs.do	ModelSim DO script to set up libraries. Called by run scripts.
clean_all	Unix script to clean ModelSim-generated files.
modelsim_sol.ini	ModelSim initialization file for UNIX environment (Solaris).
lookup_cam_8x16_srl.mif	Lookup CAM initialization file for 8 X 16 SRL CAM (Aurora/ Crossbar).

Table 9: Ethernet-to-MFRD Design Example – Scripts and Files

Name	Description
All files are similar in structure to generic traffic groomer files. See generic Scripts and Files for more details.	
<ucf>/groomer_mfrd_ml323.ucf	UCF file for implementing design on ML323 board.
<ucf>/groomer_mfrd_ml310.ucf	UCF file for implementing design on ML310 board.
<chipscope>	ChipScope™ EDN files for implementation.
<edk>	EDK portion of reference design. Used to build processor software, BRAM initialization, and processor submodule synthesis. This portion is borrowed from the MFRD but modified to allow run-time customization through the DCR bus.
<imp>/system_bd.bmm	BMM file for EDK subsystem.

Design Tools

The following versions of design tools were used to create and verify these design examples. Earlier versions might work, but it is recommended that the tool set be upgraded if possible.

- Xilinx ISE 7.1i
- Xilinx EDK 7.1
- ModelSim SE 5.8c
- Synplify Pro 7.5.1

Resource Utilization and Performance

Resource utilization depends on the number of line ports and fabric ports in the design. The following tables present formulas showing how that number is calculated.

[Table 10](#) shows an early estimation of device utilization for each module in the Ingress block.

[Table 11](#) shows an early estimation of device utilization for each module in the Egress block.

[Table 12](#) shows an early estimation of device utilization for the entire traffic groomer block.

Table 10: Ingress Resource Estimation

Modules		Virtex-II / Virtex-II Pro			Block RAM Count (1 Block RAM = 2 KB)
		LUT Count (Total)	LUTs Used as Dual-Port RAMs	Flip-Flop Count	
<i>Ingress Block (per Ethernet Port)</i>					
	LL_LINE_SEGMENTER	603	256	424	0
	LL_HDR_GEN	65		26	0
	LL_QUEUE	78		100	1
Subtotal:		746		550	1
<i>Ingress Block (per Ingress Block)</i>	<i>Number of Inputs</i>				
RR_SCHED	1 input	N/A		N/A	0
	2 inputs	275	112	111	
	4 inputs	545	224	215	
	5 inputs	1099	448	423	
LL_SCHED_MUX	1 input	N/A		N/A	1
	2 inputs	259		153	
	4 inputs	300		153	
	5 inputs	418		153	
Subtotals:⁽¹⁾					
	1 line port, M fabric ports	746		550	1
	2 line ports, M fabric ports	2026		1364	3
	4 line ports, M fabric ports	3829		2568	5

Notes:

1. Size estimates are per line card

Table 11: Egress Resource Estimation

Modules	Virtex-II / Virtex-II Pro		Block RAM Count (1 Block RAM = 2 KB)
	LUT Count	Flip-Flop Count	
Ingress Block (per Line Port)			
LL_ASSEMBLER	299N ⁽¹⁾	125N ⁽¹⁾	1N ⁽¹⁾
LL_QUEUE	87	106	2
RR_SCHED	Note (2)		Note (2)
SCHED_MUX	Note (2)		Note (2)
Subtotals:			
1 fabric port	1219	620	4
2 fabric ports	2128	974	6
4 fabric ports	3996	1682	10
Egress Block (per Egress Block)			
	Number of Outputs		
LL_SELECT_MUX	2 Outputs	N/A	N/A
	4 Outputs	414	103
	8 Outputs	468	107
	16 Outputs	577	115
Subtotals:			
1 line port, 4 fabric ports ⁽³⁾	2542	1077	6
2 line ports, 4 fabric ports ⁽³⁾	4724	2055	12
4 line ports, 4 fabric ports ⁽³⁾	9098	4011	24

Notes:

1. N = number of fabric ports (1, 2, or 4)
2. Refer to Table 10. Number of fabric ports = number of line ports
3. 4 fabric ports = 3 MGTs + loopback

Table 12: Traffic Groomer Resource Estimation

Modules	Virtex-II / Virtex-II Pro		Block RAM Count (1 Block RAM = 2 KB)
	LUT Count	Flip-Flops	
Ingress Subtotals:			
1 line port, M fabric ports ⁽¹⁾	746	550	1
2 line ports, M fabric ports ⁽¹⁾	2026	1364	3
4 line ports, M fabric ports ⁽¹⁾	3829	2568	5
Egress Subtotals:			
1 line port, 4 fabric ports ⁽²⁾	2542	1077	6
2 line ports, 4 fabric ports ⁽²⁾	4724	2055	12
4 line ports, 4 fabric ports ⁽²⁾	9098	4011	24
Totals:			
1 line port, 4 fabric ports ⁽³⁾	3288	1627	7
2 line ports, 4 fabric ports ⁽⁴⁾	6750	3419	15
4 line ports, 4 fabric ports ⁽⁵⁾	12918	6579	29

Notes:

1. Size estimate is per line card
2. 4 fabric ports = 3 MGTs + loopback
3. 4 X 4 port switch
4. 8 X 8 port switch
5. 16 X 16 port switch

Table 13 and Table 14 describe the latency through the Ingress and Egress blocks. Table 15 summarizes performance as a function of clock frequency.

Table 13: Latency through Ingress Blocks

Modules	Latency
LL_SEGMENTER	Reads entire input cell (or last bytes of input frame) before outputting data. Delay between end of cell for input data and beginning of cell for output data = 2 input clocks + 2 output clocks. <i>Example:</i> byte 1 (64 byte cell) = 69 clocks @ 125 MHz byte 64 (64 byte cell) = 25 clocks @ 125 MHz (core clock = 100 MHz)
LL_QUEUE	~ 8 clocks @ core clock.
RR_SCHED / LL_SCHED_MUX	Delay between receiving cell status to writing to scheduling MUX = 3 clocks @ core clock. Delay between reading from scheduling MUX to outputting data = 1 clock @ core clock. Average total delay (assuming synchronous clock for reading and writing scheduling MUX, and continuous writing) = 6 clocks @ core clock.
MFRD_LL_IF	None.
Total:	64-byte cell: input ethernet clock = 125 MHz, core clock = 100 MHz. Latency between last input byte and last output byte = ~ 0.8 μ s.

Table 14: Latency through Egress Blocks

Modules	Latency
MFRD_LL_IF	None.
LL_SELECT_MUX	2-3 clocks @ core clock.
LL_ASSEMBLER	Last byte input to first byte output = 7 clocks @ core clock.
RR_SCHED/ LL_SCHED_MUX	Delay between receiving cell status to writing to scheduling MUX = 3 clocks @ core clock. Delay between reading from scheduling MUX to outputting data = 1 clock @ core clock. Average total delay (assuming synchronous clock for reading and writing scheduling MUX, and continuous writing) = 6 clocks @ core clock.
LL_QUEUE	Byte 1 = ~ 2 clocks @ core clock. Last byte = Depends on length of frame, depth of queue, and frequency ratio between core clock and Ethernet clock.

Table 15: Sub-block Performance

Sub-block	Maximum Frequency (per Clock Region)	Overall Maximum Frequency
LL_HDR_GEN	347 MHz	347 MHz
LL_SEGMENTER	Rx: 345 MHz, Tx: 267 MHz	267 MHz
LL_QUEUE	Rx: 311 MHz, Tx: 386 MHz	311 MHz
LL_SCHED_MUX	Mem: 315 MHz, Data: 367 MHz	315 MHz (4 port)
RR_SCHED	Mem: 137 MHz, Data: 434 MHz	137 MHz (4 ports)
LL_SELECT_MUX	181 MHz	181 MHz (16 ports)
LL_ASSEMBLER	Rx: 235 MHz, Tx: 275 MHz	235 MHz
GbE MAC	See GbE Documentation	See GbE Documentation
MFRD	See MFRD Documentation	See MFRD Documentation

The design is capable of operation at GbE line rates, but it would need to be further optimized to meet the performance requirements of various configurations. One such example is a 2-line-port, 2-fabric-port switch. To prevent the Ingress block from dropping packets, the line side would operate at 125 MHz (125 MHz x 8 = 1 Gb/s), while the fabric side would run at 125 MHz x (number of line ports / 4). In this case, the fabric clock would run at 62.5 MHz. The input queues would also need to be able to store N full-length Ethernet frames (where N = number of line ports) for optimal performance.

The design requirements are similar for the egress side. The fabric clock would need to run at 125 MHz x (number of fabric ports / 4). Fabric egress queues would need to be able to store M full-length Ethernet frames (where M = number of fabric ports). Because the fabric clocks are shared in the traffic groomer design, it is desirable for the number of line ports and the number of fabric ports to be similar.

System Integration Considerations

This reference design allows for the rapid prototyping of an Ethernet MAC connected to the MFRD. It is a good framework for further design activity and would require additional traffic grooming and optimized scheduling functions to make it more robust. In addition, out-of-band flow control is very rudimentary; this would need to be expanded to provide adequate traffic management functionality.

Conclusion

This application note describes implementation of a traffic groomer that provides segmentation, reassembly, and basic scheduling functions between an Ethernet MAC and the MFRD. The design can be customized for a different number of line and fabric ports. It can have a custom-length SAR header.

References

The following documents provide additional information useful to this application note:

1. Xilinx Application Note [XAPP698](#), *Mesh Fabric Reference Design*.
2. Xilinx Application Note [XAPP691](#), *Parameterizable LocalLink FIFO*.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/24/06	1.0	Initial Xilinx release.