

SPANIDS Load Balancer Prototype Design and Interface Specification

Lambert Schaelicke

*Technical Report TR-2005-06
Department of Computer Science and Engineering
University of Notre Dame*

May/13/05

The SPANIDS prototype hardware distributes packets from a Gigabit Ethernet link to a set of NIDS sensors by rewriting the destination MAC address such that a commodity switch routes individual packets to different sensors. As such, the SPANIDS custom hardware makes routing decisions but leaves the task of forwarding and switching to an off-the-shelf switch. The SPANIDS load balancing hardware provides one GigE input that receives traffic off a passive tap, and forwards packets with the modified destination MAC addresses to the switch. Load balancing feedback from individual sensors arrives on the receive portion of the link to the switch.

The prototype hardware is implemented on a DN3000k10S board from the Dini Group. This board hosts one Xilinx Virtex-II 2v6000 FPGA and two GigE daughtercards from Metanetworks. Hardware development is carried out using Verilog as the hardware description language, Mentor Graphics Modelsim as a simulation tool and Xilinx ISE Foundation including the XST Synthesizer for implementation.

This document is a collection of design and interface specifications for the SPANIDS load balancer. Individual sections are written by various contributors, including Divish Ranjan, Ahmad Zakaria, Michael Heilman and Lambert Schaelicke.

Revision History

- 2/6/2004: Initial version of document
- 2/16/2004: Added table of PCI registers, updated delay pipe and PCI documents
- 2/27/2004: Added info on Verilog include file
- 3/19/2004: Added section on init/flow control frames, added top-level sectioning, reformatted table of contents
- 3/30/2004: Added section on flow control receiver, created section on loadbalancer module, added LED section
- 4/20/2004: Added preliminary performance monitor section
- 5/4/2004: Added section of alternate packet generator with PLI code
- 8/8/2004: Updated performance monitor section, added 64-bit counter and pulse generator
- 9/24/2004: Added sensor packet rates, sensor bucket counts and hot list sections.
- 3/8/2005: Updated PCI register definitions
- 3/28/2005: Updated PCI register definitions & delay pipe
- 4/22/2005: Added hash table and feedback processing description
- 4/28/2005: Finished initial version of technical report
- 5/13/2005: Added section on buffer overflow handling

Table of Contents

Section I: Overview

1. Architecture - - - - -	1
2. Logistics - - - - -	2
3. Interfaces and Data Formats - - - - -	3
4. Initialization and Flow Control Frames - - - - -	5
5. LED Definitions - - - - -	6
6. PCI Register Definitions - - - - -	6

Section II: External Interface Modules

1. Asynchronous FIFO - - - - -	11
2. Delay Pipeline and MAC Address Rewrite - - - - -	14
3. Transmit Control - - - - -	17
4. PCI Target - - - - -	19
5. Performance Monitor - - - - -	22
6. Performance Counters - - - - -	25
7. Frame Acknowledgement - - - - -	28
8. Frame Latch - - - - -	29

Section III: Load Balancer

1. Load Balancer Overview - - - - -	30
2. Packet Decoder - - - - -	33
3. Hash Functions - - - - -	34
4. Hash Table - - - - -	35
5. Sensor Feedback Processing - - - - -	40
6. Flow Control Receiver - - - - -	41
7. Sensor Packet Rate Table and Cold List - - - - -	43
8. Sensor Bucket Count Table - - - - -	45
9. Sensor Hot List - - - - -	47
10. Hash Table Feedback Processing - - - - -	49

Section IV: Test Environment

1. Packet Generator - - - - -	53
2. TCPDump Packet Generator - - - - -	55
3. Packet Dump - - - - -	57
4. Packet Monitor - - - - -	58
5. PCI Bus - - - - -	59

Section I: Overview

1 Architecture

The Spanids NIDS Hardware Load Balancer distributes packets received from a Gigabit Ethernet tap to a number of sensors by rewriting the original destination MAC addresses such that packets are appropriately forwarded to the intended sensor. The actual switching of frames is performed by a separate commodity Ethernet switch.

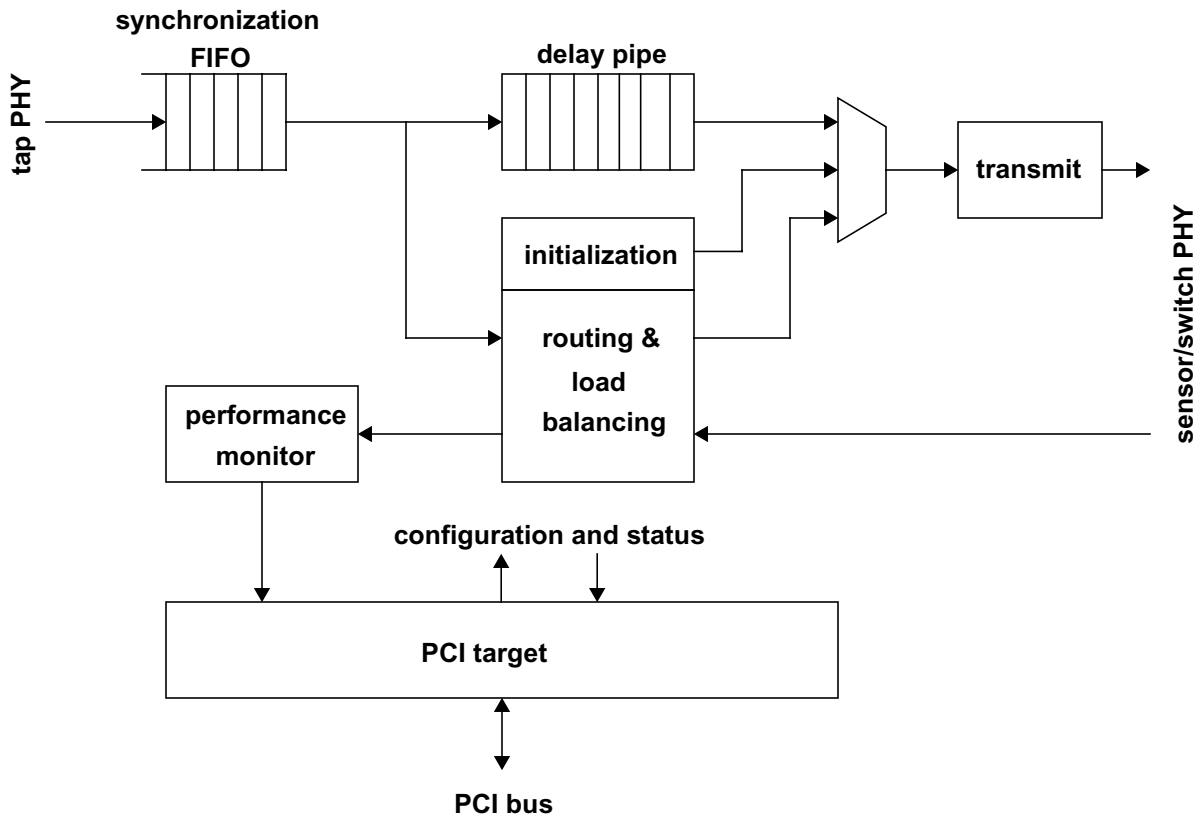


Figure 1: Spanids Loadbalancer Architecture

Gigabit Ethernet physical interfaces provide their own clock when operating in slave mode. Since the load balancer forwards frames between independent PHY interfaces, incoming frames are first synchronized to the transmit clock by an asynchronous FIFO. The majority of the hardware operates at the receive clock. Frames are forwarded to the routing hardware, and enter a delay pipeline at the same time. The pipeline delays frames by a sufficient number of cycles to let the routing hardware produce a new destination MAC address which is multiplexed into the frame as it leaves the delay pipeline. The transmit stage calculates a new checksum, appends it to the outgoing frame and forwards it to the transmit PHY. The receive portion of the same PHY serves to receive flow control and configuration information from the sensors. The initialization module disables the routing and forwarding logic, sends out an initialization request packet and waits for several seconds so that all attached sensors can reply and properly configure the load balancing

logic. The PCI target module implements a 33 Mhz 32-bit PCI target device with two memory address spaces. Internally it provides unidirectional 32-bit read and write interfaces that can be connected to any number of status or performance monitors. A dedicated performance monitoring subsystem collects data from the load balancer and exports it via the PCI interface.

2 Logistics

2.1 Version Control

All Verilog design files as well as configuration and support files are managed in a CVS repository in the Spanids project directory: */afs/nd.edu/user37/spanids/CVSROOT*. To access the CVS repository, set the *CVSROOT* environment variable to */afs/nd.edu/user37/spanids/CVSROOT*.

The usual CVS commands apply checkout, checkin and adding of files, with a few additional dependencies. When adding a Verilog file, usually one or more rules must be edited and added in the *Makefile* in the *top/* directory. In addition, if the Verilog file is part of the hardware design to be implemented on the FPGA board, it must be manually added to the ISE project in the *ISE/fpga/* directory using the Xilinx ISE project manager. This process generates a new project file with the name derived from the module name and the file extension *.jhd* which must be added to the CVS repository.

2.2 Verilog Simulation

All Verilog files and libraries required to simulate the prototype hardware are compiled with the help of a *Makefile* in the *top/* directory. This process creates a working library, converts state machine (FSM) descriptions into Verilog and compiles Verilog modules for simulation using Mentor Graphics Modelsim.

To simulate a design, a variety of environment variables must be set using the shell script provided in the Mentor Graphics installation tree: */usr/local/src/idea_en2002/mentor_en2002*. The command to start a simulation is *vsim test*, where the argument corresponds to the top-level module name. Post place and route designs are simulated in the *top_post/* directory. The provided *Makefile* compiles all required component libraries, the annotated Verilog file and the test bench. The simulation command is: *vsim -l simprimis_ver test glbls*.

2.3 Implementation

Designs are implemented using the Xilinx ISE toolset. The *ISE/* directory contains the *fpga/* project directory, which houses all implementation related files. When adding Verilog files to the design, it is necessary to manually add these files to the ISE project. This step produces a *.jdh* file that must be added to the CVS repository. The ISE design tools are run as: *ise fpga/fpga.npl*. This brings up the ISE project manager with the correct project. Common implementation steps include generating a post-place&route simulation file, and creating programming files. Always check that the synthesizer created the desired structures, especially flip flops, memory and multiplexers.

Generally, latches are undesirable and indicate a possible problem with the Verilog code, for instance incomplete case-statements. Also, check for any other warnings or errors, and verify that the timing constraints are met with a sufficient slack (at least 10 percent of cycle time).

2.4 Documentation

The documentation is composed of multiple documents combined into a FrameMaker book. When editing any part of the documentation, it is necessary to update page and heading numbers by running *make* in the *Docs/* directory. New sections are best added by first making a copy of a suitable section, then adding it to the book file in the appropriate order and adding the new section file to the CVS repository. The *intro.fm* file contains a revision history that should be updated for any major edits. The *overview.fm* file contains a table summarizing the addresses and usage of PCI registers, it should also be kept up to date.

3 Interfaces and Data Formats

3.1 Metanetworks MP1000tx Interface

While Gigabit Ethernet links operate at 125 Mhz, the PHY daughtercard converts the GMII interface into a 16-bit interface running at 62.5 Mhz. Each daughtercard provides its own clock that is derived from the network link, and this clock is used for both receive and transmit operation. Ethernet frames start with an 8-byte (4 cycle) preamble consisting of '0x55' values, followed by the Ethernet header, payload and 4-byte checksum. Frames always start 16-bit aligned, but may be of odd size and thus end with only one valid byte on the 16-bit data bus. On a 16-bit data bus, bits 15-8 carry the even byte and bits 7-0 carry the odd byte, thus the last byte of an odd-size frame appears on bits 15-8. The PHY board does not check or calculate the checksum, and does not enforce the minimum inter-packet gap, this is the responsibility of the FPGA.

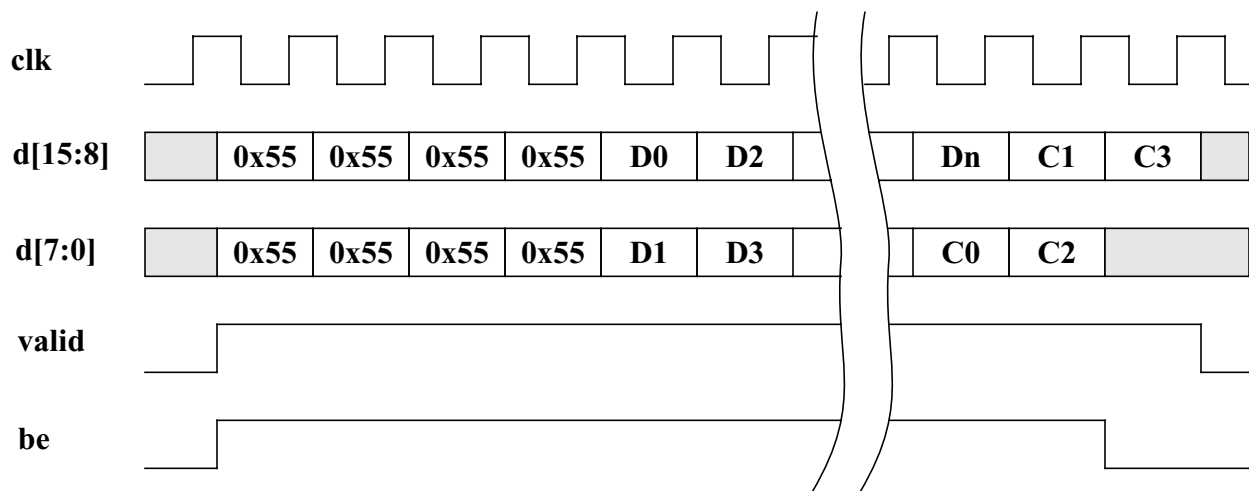


Figure 2: MP1000tx Interface Signals

Figure 2 above shows the transmission of an odd-size frame, starting with 4 cycles of preamble, transmission of the packet from D0 through Dn, followed by the 4-byte checksum C0 through C3. The ‘frame’ signal is high for the entire duration of the frame transfer, while the byte-enable signal goes low during the last cycle to indicate that only the upper byte is valid. For an even-size frame, byte-enable would remain high until the end, identical to the valid signal. The error signal provided by the PHY daughtercard will not be used, the transmit-error signal should be set to low. The interface for transmission and receive operation is identical. The pin-constraint file maps three PHY interfaces to FPGA pins. receive (incoming) ports are labeled PHYx_yy_RX_zz, with ‘x’ being port 1, 2 or 3, and ‘yy’ and ‘zz’ being the remainder of the signal name. Similarly, transmit (outgoing) frames are labeled PHYx_yy_TX_zz. Generally, connectors J23 and J24 (PHY1 and PHY2) should be used.

Frames can be transmitted at any time, on a full-duplex link there is no flow control the core FPGA. It is the responsibility of the sender to enforce minimum inter-packet gap. It is critical that all input signals are latched in registers before being used for any logic operation. This design adds one cycle of delay, but ensure greater independence from external wire delays. Similarly, all output signals are latched in registers before leaving the chip. Both types of registers should be placed in IOBs. Also note that the clock provided by the PHY board must be buffered and phase aligned through a DLL (BUFGDLL in Xilinx ISE).

3.2 Internal Frame Format

Inside the SPANIDS load balancer, packets are represented by the same 16-bit interface, but without checksum and with a slightly different control signal timing. To simplify the transmission control logic and checksum calculation, the valid-signal remains high only until the second to last data cycle, i.e. it goes low in the last cycle with valid data. The byte-enable signal is also shortened to indicate even or odd size frames in the last data cycle.

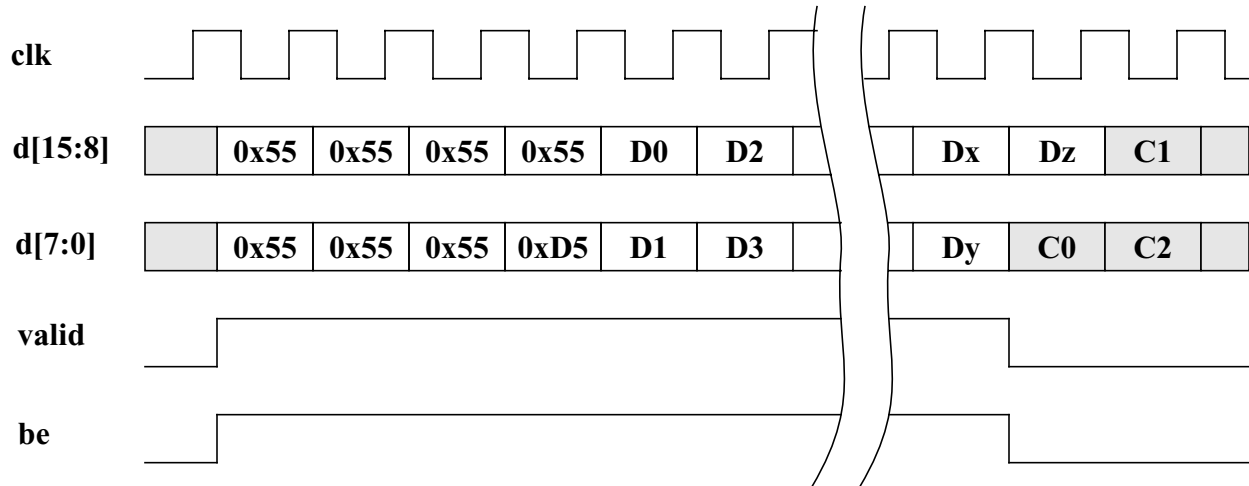


Figure 3: Internal Frame Format, Note Shortened valid and byte-enable Signals.

Figure 3 above shows the internal format of an odd-size frame. Note that the valid-signal is low while the last data byte (Dx) is transmitted. In the same cycle, byte-enable is low to indicate an odd size frame.

4 Initialization and Flow Control Frames

During initialization and to communicate flow control information, the load balancer and sensors exchange Ethernet frames of a specific format. During the initialization phase, immediately after reset or as triggered via the PCI interface, the delay pipeline module broadcasts an init-request packet to all sensors. Each sensor responds with a similarly formatted packet that contains its IP and MAC address. During normal operation, sensors may send flow control packets to provide feedback to the loadbalancer. The following table describes the format of each packet in detail.

Field	Width (bytes)	Init-Request	Init-Reply	Flow Control
MAC Destination Address	6	0xFFFFFFFF broadcast	loadbalancer MAC address	load balancer MAC address
MAC Source Address	6	loadbalancer address	sensor address	sensor address
Ethernet Type	2	0x0800 - IP	0x0800 - IP	0x0800 - IP
IP Header Word 0 type, flags, header length	4	0x45000028	0x45000028	0x45000028
IP Header Word 1 ID, fragment offset	4	0x00004000	0x00004000	0x00004000
IP Header Word 2 TTL, protocol, header checksum	4	0x4011---- udp	0x4011---- udp	0x4011---- udp
IP Header Word 3 source address	4	loadbalancer IP address	sensor IP address	sensor IP address
IP Header Word 4 destination address	4	0xFFFFFFFF broadcast	loadbalancer IP address	loadbalancer IP address
UDP Header Word 0 source/destination port	4	0x1BCD1BCD	0x1BCD1BCD	0x1BCD1BCD
UDP Header Word 1 length, checksum	4	0x0014---- 20 bytes	0x0014----	0x0014----
OpCode	2	0x0000 - request	0x0001 - reply	0x0002 - flow control
Sender MAC Address	6	loadbalancer MAC address	sensor address	sensor address
Sender IP Address	4	loadbalancer IP address	sensor address	sensor address
Padding	6	0x000000000000	0x000000000000	0x000000000000

Table 1: Communication Frame Formats

Notice that all packets are IP/UDP packets of the same length. The init-request packet is broadcast to all MAC and IP addresses, while the other two packets originate from one specific sensor node.

If any portion of the IP header of the Init-request packet changes, the IP header checksum (word 2) must be manually recomputed since it is hardwired into the loadbalancer. The other two packets originate from a proper protocol stack and have their checksum computed automatically.

The UDP port number on the sensors is fixed to 0x1BCD, the init-request packet is directed at this port and the reply and flow control packets are expected to originate from this port. For the init-request packet, the UDP checksum can be set to 0 to indicate that no checksum is computed or verified. Packets originating at the sensors have the proper UDP checksum calculated, but it is currently not verified by the loadbalancer.

Each packet contains the senders MAC and IP address in the payload, as well as an opcode that indicates the nature of the packet. The MAC and IP addresses are required so that the software daemon running on the sensors can form a proper reply packet. The opcode indicates the type of packet.

5 LED Definitions

The prototype board provides 8 LEDs that are visible through the PCI slot cover. Some of these LEDs are used for general board functions while others are routed to the FPGA and can be used to visually indicate the loadbalancer operation. The following table summarized the function of each LED.

LED	Description
0	FPGA programming success
1	SmartMedia card invalid
2	Configuration error
3	on during configuration, flashes when transmitting packet to sensors
4	on during configuration until initialization starts, flashes when receiving packet from sensors
5	on during load balancer reset, flashes during PCI transactions
6	Roboclock 1 locked
7	Roboclock 2 locked

Table 2: LED Definitions

6 PCI Register Definitions

The following table summarizes the purpose and definition of all registers accessible via the PCI interface. It is expected that these will change frequently to adapt to various debugging and testing needs, and it is important that the table will be kept up to date with the current hardware implementation. These definitions, as well as other constants, are defined in the Verilog-include file *top/defs.v*

Offset (hex)	R/W	Description
00	R	Rd: returns magic number 0xDEADBEEF
04	R/W	Initialization control R: initialization status { ~init_reset, init_start, init_done } W: start initialization (any value)
08	R/W	Loadbalancer mode: { 00 .. 00, no_frames, no_routing } R: read current mode W: set mode
0C	R/W	Performance monitor control: { 00 .. 00, snapshot, clear 2, clear1, clear0 } R: read status of current operation W: start snapshot and/or clear
10-14	R/W	Packet count (64 bits) W-upper: latch current packet count W-lower: clear counter R: read latched packet count
18-1C	R/W	Ignored non-IP packet count (64 bits) W-upper: latch current counter value W-lower: clear counter R: read latched count
20	R	Number of sensors
24	R	Initialization latency, number of cycles between transmission of init-request and first init-reply packet
28	R/W	Rate period value (6 bit, in 250 millisecond units): W: set value R: read current value
2C	R/W	Random number control: { 00..00, high, 00..00 low } (5 bits each) W: set values R: read current values
30	R/W	Promotion threshold (8 bits) in 0.25 units W: set value R: read current value
34	R/W	Number of buckets to move or promote (4 bits) W: set value R: read current value
38	R/W	Loadbalancer mode (3 bits) (move/promote/random/average/static) W: set value R: read current value
40-44	R/W	Number of hash buckets moved (64 bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
48-4C	R/W	Number of hash buckets promoted (64 bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count

Table 3: PCI Region 0 Address Map

Offset (hex)	R/W	Description
50-54	R/W	Number of hash buckets demoted (64 bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
58-5C	R/W	Number of packets routed through hash level 0 (64-bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
60-64	R/W	Number of packets routed through hash level 1 (64-bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
68-6C	R/W	Number of packets routed through hash level 2 (64-bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
70-74	R/W	Number of packets routed through hash level 3 (64-bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
78-7C	R/W	Number of packets routed through hash level 4 - round-robin (64-bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
80	R	length of most recent packet received
84	R	length of most recent packet transmitted
88	R	length of most recent flow control packet
90 - 9C	R	most recent hash level 0-3 index
A0	R	most recent operand 0 for intensity * bucket-count (intensity)
A4	R	most recent operand 1 for intensity * bucket-count (bucket-count)
A8-AC	R	most recent result for intensity * bucket-count (64 bit)
B0	R	most recent operand 0 for packet-count * threshold (packet-count)
B4	R	most recent operand 1 for packet-count * threshold (threshold)
B8-BC	R	most recent result for packet-count * threshold (64 bit)
C0-C4	R/W	number of packets lost due to buffer overflow (64-bits) W-upper: latch current counter value W-lower: clear counter R: read latched packet count
100-17C	W	init packet
6000 - 7FFC	R	most recent packet received
A000 - BFFC	R	most recent packet transmitted
E000 - FFFC	R	most recent flow control or init packet received

Table 3: PCI Region 0 Address Map

Offset (hex)	R/W	Description
10000 - 13FFC	R	hash table 0 contents { intensity, promoted, sensor }
14000 - 17FFC	R	hash table 1 contents { intensity, promoted, sensor }
18000 - 1BFFC	R	hash table 2 contents { intensity, promoted, sensor }
1C000 - 1FFFC	R	hash table 3 contents { intensity, promoted, sensor }

Table 3: PCI Region 0 Address Map

Writing any value to the init-register triggers a full re-initialization of the loadbalancer, including transmission of the init-request packet. By reading from the register the progress of the initialization can be observed. The mode register control the operation of the load balancer. Bit 0 disables routing and rewriting of Ethernet MAC addresses, instead all frames are passed through without modification. Bit 1 disables forwarding of frames completely. Initially both bits are clear, enabling full operation of the loadbalancer.

Writing to the performance monitor control register starts a snapshot or clear operation, depending on which bit is set. The three clear-bits specify which portion of each performance monitoring record is to be cleared. Bit 2 (clear2) clears the packet count, bit 1 clears the byte count and bit 0 clears the flow control count. Multiple clear-bits can be set in one operation, in which case all respective entries will be cleared. When simultaneously starting a snapshot and clear operation, the performance monitor first performs the snapshot immediately followed by the clear. Reading from the same register returns the status of the performance monitor. The respective bits remain set until the operation is complete.

Before reading the 64-bit packet counter and non-IP counter, the current counter value needs to be latched by writing to the upper 32-bit register. The value read from the upper and lower register remains stable until the next snapshot operation. Writing to the lower 32-bit register clears the counter as well as the latched value.

The rate period value controls the period at which packet rates are measured in the load balancer, it can be written as well as read for confirmation. The base unit is 250 milliseconds, with a default of 4 units or 1 second. Writing a new value takes effect at the end of the current period.

Random number generation is controlled by 5-bit upper and lower bound. Random numbers are generated within this range, including the bound values. Note that when changing a value, it may not take effect in the current cycle of the random number generator. If the upper bound is adjusted to a lower value, the random number generator may produce values that exceed even the previous upper bound until the current rotation is complete. The promotion threshold controls by what factor a hash bucket intensity needs to exceed the average bucket intensity to be promoted. It is represented as an 8-bit number in 0.25 increments. For instance, a value of 5 corresponds to a threshold of 1.25. The bucket count register controls how many hash buckets are moved or promoted for each flow-control operation. Finally, the loadbalancer heuristic can be controlled by writing a 3-bit code to the control register. Supported heuristics are: always-move (000), always-promote (001), random (010), weighted-average (011) and static (100).

Three 64-bit counters record the number of hash buckets moved, promoted or demoted. Similarly to the packet counters, these counters need to be latched by writing to the upper word (aligned at an 8-byte boundary) before reading the counter value. Writing to the lower (odd) word clears the counter.

The last packet length and packet contents registers are intended for debugging purposes. Although the length registers specify length in bytes, they round up to the next even byte count. Other debugging information includes the last hash values, the most recent incoming, outgoing and flow-control packets and the contents of the hash tables. Each hash table entry is available as a 32-bit word consisting of the promote bit (30), the sensor number (29-24) and the bucket intensity in packets since the last clear operation (23-0). Finally, the initialization packet can be changed dynamically by writing to the appropriate word locations. Note that the init-packet needs to start with an 8-byte Ethernet preamble and needs to be a complete and correct IP/UDP packet, excluding the Ethernet checksum which is calculated dynamically. Note also that changing the sender MAC or IP address and the port number may prevent systems from signing on correctly, since the flow control receiver matches these fields against static values.

Offset (hex)	R/W	Description
000 - 7FC	R	performance monitoring records
800 - FFC	R	snapshot performance monitoring records

Table 4: PCI Region 1 Address Map

PCI region 1 contains the performance monitor records, as shown in the table above.

Section II: External Interface Modules

This section describes the architecture of all modules surrounding the actual loadbalancer component. These modules deal with interfacing to the Ethernet PHY ports or the PCI bus, or provide other general support functionality.

1 Asynchronous FIFO

files: fifo/afifo.v, fifo/ASYNC_FIFO_V5_0.v, fifo/fifocontrol.fsm

An asynchronous FIFO is required to synchronize between the receive and transmit clocks on different PHY boards. In addition, the FIFO module converts the external PHY interface into the internal frame format.

1.1 FIFO Functionality

The asynchronous FIFO serves as a synchronization stage between the two clock domains, and as a buffer for incoming frames. To compensate for possible clock rate differences between the incoming and outgoing PHY, some amount of data must be buffered in the asynchronous FIFO before it is read out. Otherwise, a slower receive port may force bubbles at the output of the FIFO. The minimum Ethernet frame is 72 bytes long (64 bytes data plus 8 bytes preamble, or 36 cycles), with a maximum of 1526 bytes. Internally, frames are 4 bytes shorter since the checksum is stripped off. Delaying the shift-out operation until at least 34 words have been buffered leaves 745 cycles (1490 bytes) for the receiver to fill the FIFO while the internal control logic drains the FIFO over 762 cycles (1524 bytes). Assuming the internal control logic operates at the nominal 62.5 Mhz, the receiver can run as slow as 61.106 Mhz. Alternatively, with the receiver running at 62.5 Mhz, the internal logic can operate as fast as 63.92 Mhz. This allowable range exceeds the requirements of the IEEE 802.11 standard of 100 ppm and guarantees bubble-free synchronization of the two clock ranges.

At the same time the FIFO needs to handle situations where the incoming clock frequency exceeds that of the output port. In this case it is inevitable that frames are dropped, regardless of the FIFO size.

To hold an entire frame, the asynchronous FIFO needs to be at least 762 entries deep and needs to provide full-signals with a granularity of 32 entries. The Xilinx Core Generator provides a suitable FIFO with a depth of 1023 entries and with 5-bit binary full-vectors synchronized to the read and write ports.

1.2 Design

The asynchronous FIFO module is directly connected to the PHY receive port. Both valid and byte-enable signal must be shortened according to the internal interface specification. This is accomplished by delaying all incoming signals by three cycles, and generating new control signals from the original and the delayed signal. The *valid* signal needs to be shortened by three cycles, *be*

is shortened by two cycles, and the FIFO *shift-in* signal is derived from the original *valid* signal shortened by one cycle. Signals are shortened by ANDing the delayed signal with the same signal from a previous pipeline stage. For instance, the internal *valid* signal goes high when the first preamble word leaves the delay registers, but goes low when the last frame word enters the delay stages, three cycles before that data leaves the delay stage. All interface signals, including the derived *valid* and *be* signals, are shifted into the FIFO.

Shifting frames into the FIFO is controlled by the overflow state machine. This state machine observes the full-vector synchronized to the write port and disables the shift signal if the FIFO has fewer than 64 entries left. To avoid corrupting frames, the shift-enable signal only changes during the frame gap. For each frame that is not shifted into the FIFO, the state machine drives an overrun signal for one cycle to allow an external counter to count buffer overruns.

A finite state machine observes the FIFO read port, controls the shift-out operation and generates *valid* and *be* signals synchronous to the read port. If any of the full-vector bits are active, indicating that the FIFO has at least 32 entries filled, the state machine enables the shift-out signal to start reading data. After one cycle, the first valid data item appears on the FIFO output ports. In the next cycle, the state machine drives both *valid* and *be* to indicate the beginning of a frame. The state machine then observes the *valid* and *be* signals on the FIFO read port. When *valid* goes low, indicating the last data cycle for the current frame, the state machine enters either one of two last-cycle states to signal the end of an even or odd frame, depending on the FIFO *be* signal. Finally, the state machine inserts 5 idle cycles to enforce the inter-packet gap. Ethernet requires that consecutive frames be spaced by 96 bits, equivalent to 12 bytes or 6 cycles. The internal frame format leaves out the checksum, adding two more cycles, for a total of 8 cycles. Since it takes three cycles for the first valid word to leave the module, the state machine adds 5 idle cycles at the end of a frame.

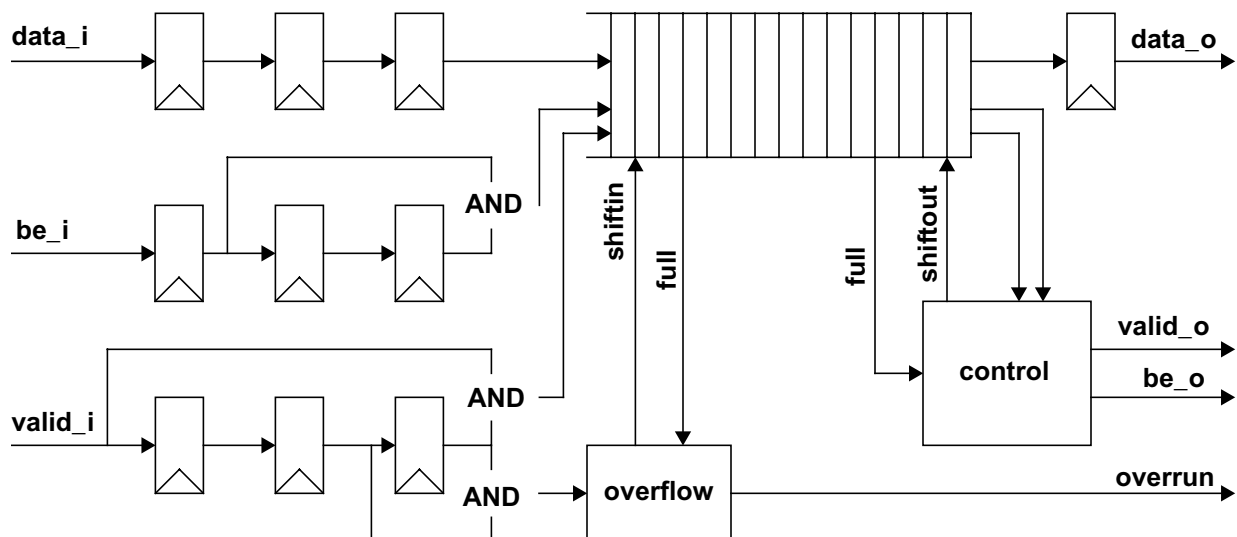


Figure 4: Asynchronous FIFO Module

Note that although the FIFO provides *valid* and *be* signals according to the internal format, the state machine recreates these signals with a one-cycle delay. Initially, as well as between frames, the FIFO outputs are undefined, but the state machine observes these signals only when they are valid. Furthermore, the shift-in logic shifts one additional word into the FIFO at the end of each frame. This is required to provide a consistent initial state at the FIFO output. FIFO output changes only when the state machine drives shift-out. When waiting for a new frame, the outputs are undefined and remain undefined until the next shift-out operation. Inserting an invalid word between frames lets the state machine always shift out and discard the first word.

1.3 Interface

- reset: asynchronous reset
- clk_i, clk_o: input (receive) and output (transmit) clock
- data_i[15:0], data_o[15:0]: input and output data, synchronized to clk_i and clk_o respectively
- valid_i, valid_o: input and output valid signals
- be_i, be_o: input and output byte-enable
- overrun: one-cycle active-high signal indicating dropped packet due to buffer overrun, synchronous to clk_i (output)

2 Delay Pipeline and MAC Address Rewrite

files: *delaypipe/delaypipe.v, delaypipe/rewrite.fsm, delaypipe/initcntl.fsm, delaypipe/delay_cntr_rd.fsm, delaypipe/delay_cntr_wr.fsm, counters/counter7.fsm, memory/ramd.v*

The delay pipeline delays internal frames by a number of cycles to let the routing and load balancing logic produce a new destination MAC address. Frames leaving the delay pipeline may have their destination MAC address rewritten, or may be completely ignored in favor of a frame transmitted from the initialization logic. In addition, this module implements the initialization control that transmits an initialization request packet and delays starting up the routing logic for several seconds.

2.1 Functionality

The delay pipeline receives frames in the internal format. The complete frame, including valid and byte-enable signal, is shifted into a delay pipeline. At any point before the original destination MAC address leaves the pipeline, a new MAC address can be loaded into an internal register.

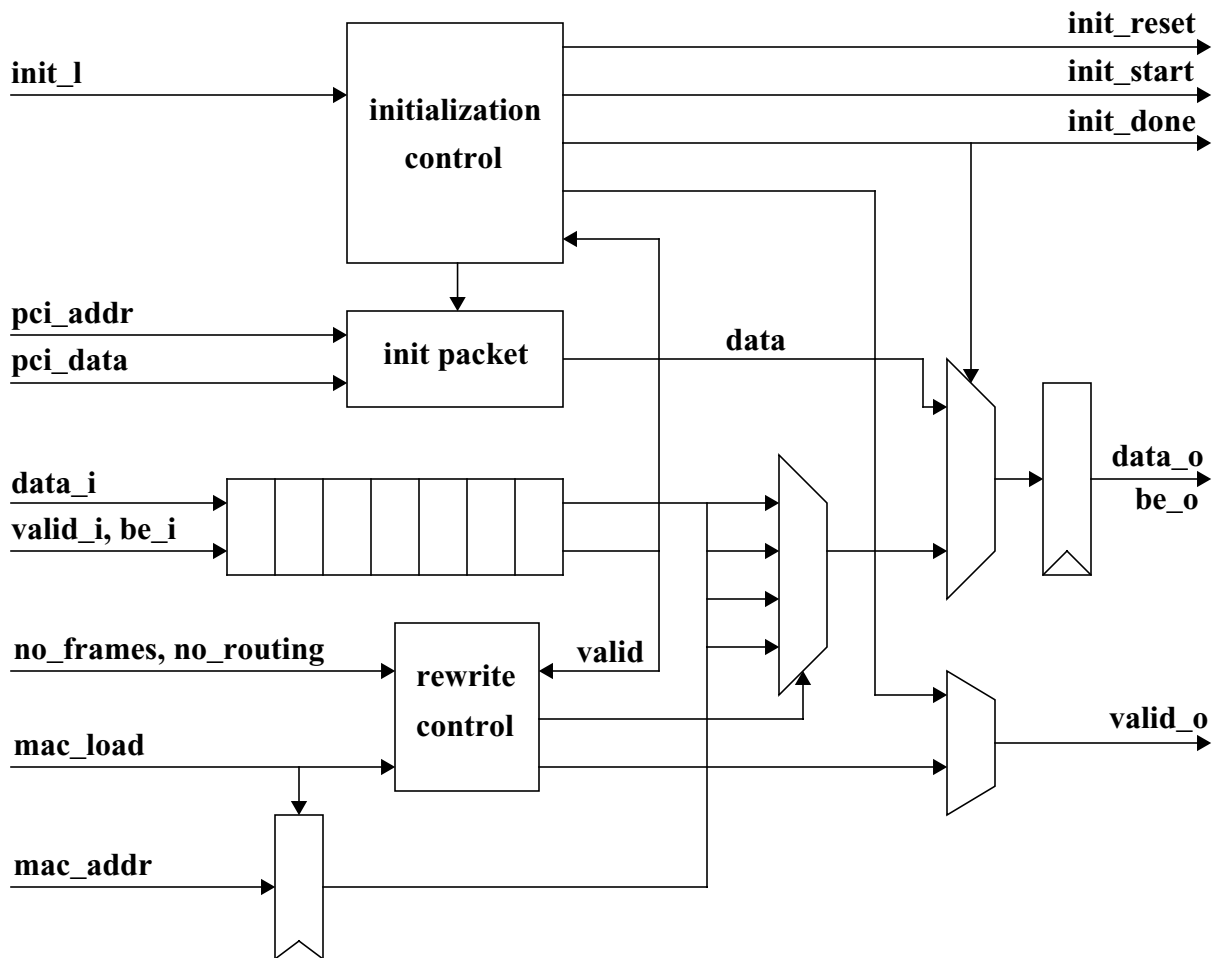


Figure 5: Delay Pipeline and MAC Address Rewrite Stage

The delay pipeline is implemented as a dual-ported RAM with free-running counters to provide read and write addresses. Addresses are offset from each other to realize the delay. The read address counter starts at address 0, while the write address counter is initialized to 48, thus producing a 49-cycle delay. Note that the delay does not need to match the exact load balancer delay, as long as the new MAC address is latched before the end of the current frame's preamble.

A state machine observes the outgoing packet stream as well as the MAC load signal and controls a multiplexer to rewrite the original destination MAC address. The state machine observes the Ethernet frame preamble and waits for the start-frame delimiter. If the load signal has been active at any point before the frame proper begins, it replaces the original destination MAC address with the new address, and replaces the source MAC address with the load balancer's address. If no MAC address is latched, the packet is ignored and not forwarded. If routing is disabled (MAC address are not to be rewritten), all packets are forwarded without rewriting of the MAC address. If packet forwarding is disabled, no packets are forwarded at all.

The initialization control logic is implemented as a 28-bit counter and a control state machine. The free-running counter of 28 bits width is constructed by chaining four 7-bit counters with three enable-inputs each. Running at 62.5 Mhz, it rolls over once every 4.295 seconds. The control state machine observes the counter roll-overs and controls the transmission of the initialization request packet. After reset, the state machine waits for the counter to roll over three times while it disables the forwarding of any incoming packets via a multiplexer in the delay pipeline stage. This delay of about 13 seconds leaves sufficient time for the outgoing PHY chip to power up and negotiate the link state, before the first packet is transmitted. During the first counter period, the state machine also drives a reset signal that initializes the load balancer logic.

When the counter rolls over for the third time, the control state machine drives the init-request packet through the multiplexer. The packet is generated by indexing into a table, its format is described in a previous section. After transmitting this packet, the control state machine waits for another two counter overflows, leaving ample time for all sensors to respond. After the first roll-over, the `init_start` signal becomes active, indicating the all initialization-response packets should have been received by that time. This signal can be used by the load balancer logic to perform the initial configuration. Upon the second overflow, the state machine may wait until the end of an incoming frame before switching the output multiplexer to its normal operation. This wait state is required to avoid driving incomplete and corrupted packets on the output link. The initialization logic controls a high-priority multiplexer that replaces both data and valid with the initialization request packet. Since initialization frames are always even, the byte-enable signal is forced high in this case.

The initialization packet is stored in an array of registers that is initialized to the packet format described in the introduction of this document. The contents of the initialization packet can be overwritten via the PCI interface. Note that the packet must start with the 8-byte Ethernet preamble. It is the responsibility of the software or user to ensure the new packet constitutes a valid IP packet, including the IP and UDP or TCP checksums. Also note that the control state machine currently assumes a fixed packet length of 64 bytes, not including the preamble.

The entire initialization sequence can be triggered externally by driving `init_l`, for instance via writing to a PCI control register. To avoid corrupting a frame currently in transit, the initialization control state machine waits for the end of the current frame before restarting initialization.

Address rewriting and frame forwarding is in addition controlled by two mode signals: `no_routing` and `no_frames`. The former signal is forwarded to the rewrite controller and masks the `mac_load` signal, thus disabling rewriting of destination MAC addresses even if an alternate MAC address was loaded. The `no_frames` signal masks the outgoing `valid` signal and thus completely disables forwarding of frames.

2.2 Interface

The delay pipeline takes as inputs in internal frame and a 48-bit MAC address with load signal and produces an internal frame.

- `clk`: clock signal (input)
- `reset`: asynchronous active-high reset signal
- `data_i[15:0]`: incoming frame data (input)
- `valid_i`, `be_i`: incoming frame valid and byte-enable signals (input)
- `mac_addr[47:0]`: new destination MAC address (input)
- `mac_load`: active-high MAC address load signal (input)
- `data_o[15:0]`: outgoing frame data (output)
- `valid_o`, `be_o`: outgoing frame valid and byte-enable (output)
- `no_routing`, `no_frames`: frame rewriting and forwarding control (input)
- `init_cntl_l`: active-low, triggers initialization sequence (input)
- `init_reset`: synchronous, active-high reset signal for loadbalancer logic (output)
- `init_start`: active-high signal, indicates end of initialization-request phase and triggers load balancer configuration (output)
- `init_done`: active-high signal, indicates end of initialization phase, resumes normal operation (output)
- `pci_clk`: PCI interface clock signal (input)
- `pci_addr<4:0>`: word-aligned address for PCI interface (input)
- `pci_data<31:0>`: PCI data (input)
- `pci_wr_l`: active-low write signal for PCI interface (input)

3 Transmit Control

files: *transmit/transmit.v, transmit/crc32_8.v, transmit/crc32_16.v, transmit/txcontrol.fsm*

This module controls the transmission of rewritten frames, including calculating and appending the checksum.

3.1 Functionality

Frames arrive at the transmit stage with the required inter-packet gap. Since there is no flow control from the PHY back to the core FPGA, frames can be sent as soon as they are available. The main responsibility of the transmit stage is to control calculation of the checksum, to insert the checksum with the correct alignment and to generate the correct *valid* and *be* signals. Frames begin with a four-cycle preamble, during which the output state machine initializes the checksum calculation logic. Four cycles after the valid-signal goes high, checksum calculation starts.

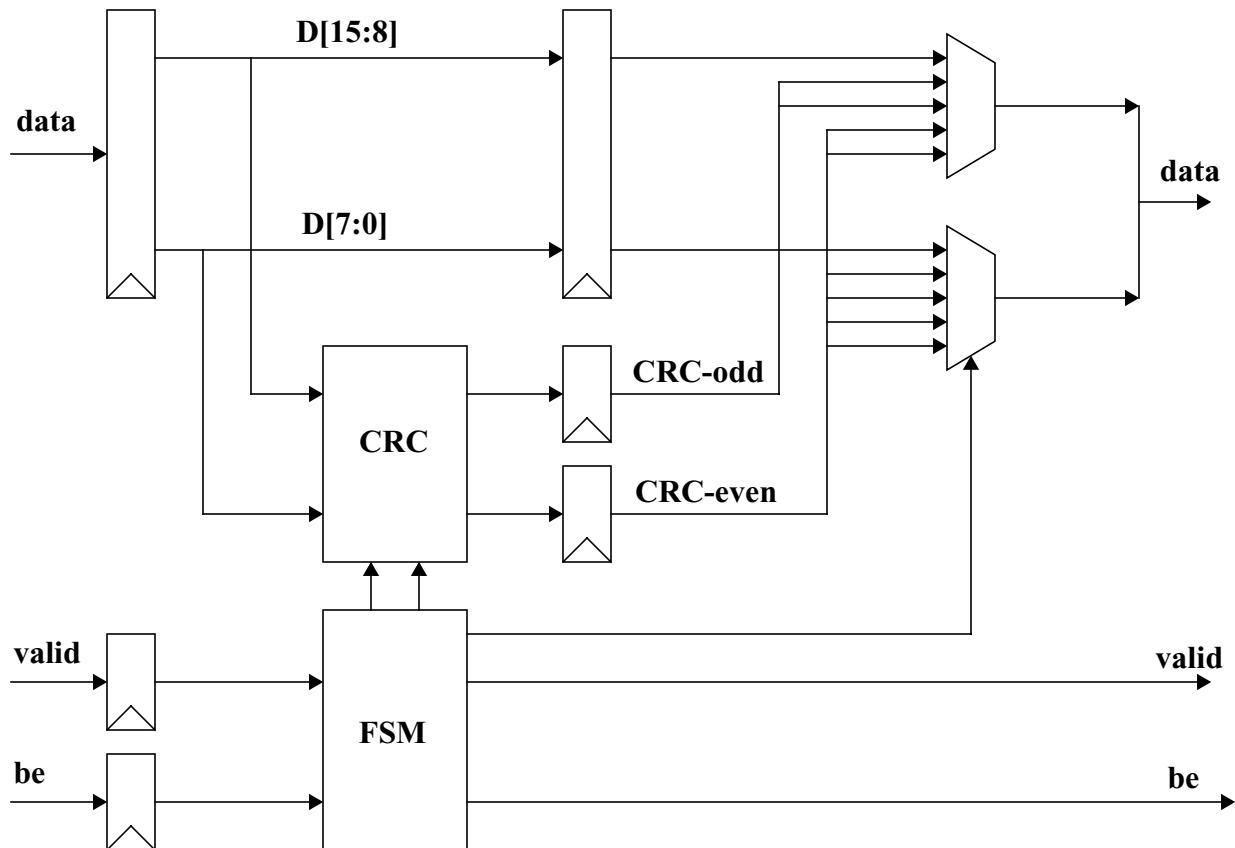


Figure 6: Transmit Control

To support odd-size frames, both the 16-bit and 8-bit checksums are calculated simultaneously. At the end of the frame, the state machine selects the correct checksum to append depending on the *be* signal. Each cycle, both checksums are inverted and bit-flipped according to the Ethernet

standard, and are stored in registers. In addition, the unmodified 16-bit checksum is fed back into the CRC calculation for the next cycle.

The incoming *valid* signal goes low one cycle before the end of data, leaving one cycle for the state machine to stop checksum calculation. In the same cycle, the *be* signal indicates an even or odd frame, letting the state machine correctly multiplex one of the calculated checksums into the data stream. In addition to the multiplexer and checksum control, the state machine also generates the *valid* and *be* signal according to the external PHY interface.

3.2 Interface

The transmission control module takes an internal frame as input and produces a modified frame according to the external PHY interface.

- *clk*: transmit clock
- *reset*: asynchronous reset signal
- *data_i*[15:0]: 16-bit frame data, not including checksum (input)
- *valid_i*, *be_i*: internal valid and byte-enable signals (input)
- *data_o*[15:0]: complete outgoing frame data (output)
- *valid_o*, *be_o*: outgoing valid and byte-enable signals (output)

4 PCI Target

files: pci/pci_target.v, pci/pci_ack.fsm

The PCI target module implements a target-only PCI device. It is derived from code provided by the Dini Group, supports two memory address spaces and single-cycle memory read and write transactions as well as configuration transactions.

4.1 Structure

The PCI target module consists of a number of synchronous circuits that implement all logic required for a 33 Mhz, 32-bit PCI target device. The device supports two non-prefetchable memory address spaces of 1 Mbyte each. The PCI interface is controlled by a state machine and a 4-bit response delay-counter. Initially, the state machine is in the *idle* state, waiting for a bus transaction. An unconfigured PCI device responds only to configuration cycles, indicated by asserting *idsel*. When observing a configuration read or write, the state machine enters the *state_cfg* state. In this state it waits until the delay counter asserts *trdy* to indicate that the device is accepting or providing data. Configuration cycles are always single-cycle transactions, hence the state machine returns to *idle*.

When decoding memory read or write transactions, the state machine enters *bus_busy* and in the next cycle *comp_addr* to decode and compare the transaction address. If the address matches any of the devices two address regions, the state machine enters *s_data*, otherwise it returns to *idle*. In *s_data* it waits until both *irdy* and *trdy* are active and then returns to *idle*.

Most PCI bus signals are controlled by asynchronous logic that is driven by the current state and other control signals. The direction of the data transfer is determined based on one bit in the transaction command encoding. For read transactions, the data source is by default the configuration register addressed by the PCI bus address. For memory read transactions, data is sourced from the backside data bus instead.

The device supports only single-cycle transactions. It drives the *stop* signal if *frame* remains asserted during the data transfer cycle (*trdy* and *irdy* active) to force termination of the transfer. The device also computes and checks parity for all valid cycles and drives *perr* if it detects a parity error. The delay between detecting a transaction and responding by asserting *irdy* is controlled by a 4-bit counter. Configuration cycles incur a delay of (configurable) delay of 10 cycles. Regular transactions incur a minimum delay of 4 cycles, which may be extended under the control of the backside agent.

4.2 Internal Interface

The backside interface of the PCI target module consists of an address bus with byte-enables, and address space identifier, read and write buses, read and write indicators and acknowledgement signals. The address space identifier distinguishes between the two address regions, it is high when region 1 is accessed and low for region 0. The address bus forwards the word address from any PCI transaction to the backside agent. Separate active-high byte-enable signals indicate which bytes are

valid. The write-data bus forwards data from the PCI bus during write transactions. Writes are signaled by asserting *wr_l*, but the PCI transaction does not conclude until the backside agent acknowledges the write by asserting (lowering) *wr_ack_l*. This flow control mechanism allows backside agents to effectively hold the write address, data and byte-enable signals until they can be processed. Similarly, read transactions are signaled by asserting *rd_l*, upon which the backside agent decodes the address, multiplexes the desired data and acknowledges the transaction by asserting *rd_ack_l*.

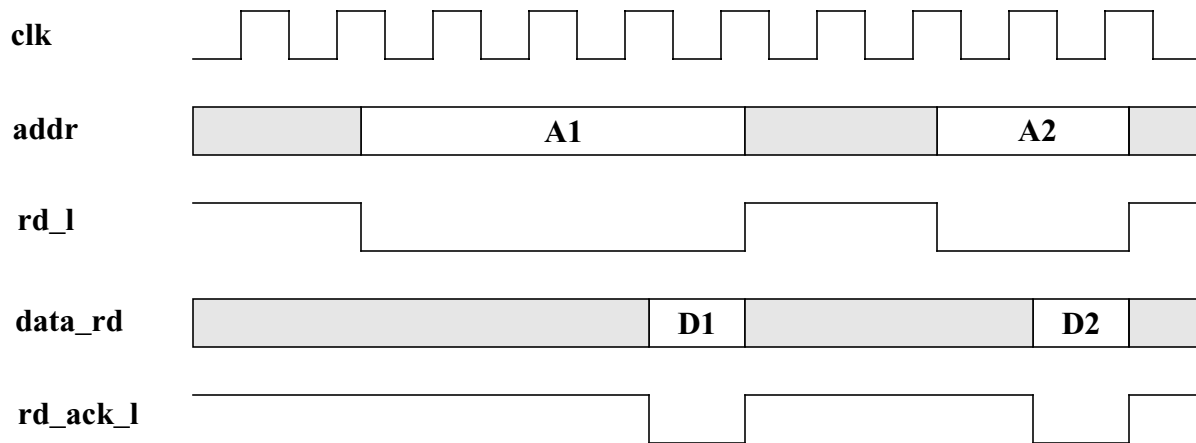


Figure 7: Two Consecutive Read Transactions with Different Acknowledgement Latencies

It is important to note that, for the correct operation of the PCI interface, the acknowledgement signals must be deasserted between transactions, and must be active for exactly one cycle. Even if no flow control is needed, these signals must remain inactive until a read or write is received, and must not be activated until at least one cycle after a read or write is signaled.

It is the responsibility of the backside agent to decode the address region and word address, properly demultiplex write-data and multiplex read-data. The acknowledgement state machine observes the internal transaction signal (read or write) and generates a single-cycle acknowledgement after a three-cycle delay. It then waits for the end of the transaction before returning to the idle-state where it waits for the next transaction. In case this simple fixed-latency acknowledgement scheme is insufficient, more complex control must be implemented by the PCI clients.

4.3 Interface

The PCI target connects to a standard 32-bit PCI bus and provides a simple 32-bit unidirectional interface for internal clients.

- **clk**: PCI clock
- **reset_l**: asynchronous active-low reset signal
- **ad[31:0]**: PCI address and data (bidirectional, tristate)
- **c_be_l**: PCI command and active-low byte-enable (input)

- par: PCI parity (bidirectional, tristate)
- perr_1, serr_1: active-low PCI error signals (output, tristate)
- frame_1, irdy_1: active-low PCI frame and initiator-ready signals (input, tristate)
- trdy_1, stop_1: active-low PCI target-ready and stop signals (output, tristate)
- devsel_1: active-low PCI device-select signal (output, tristate)
- idsel: PCI device select during configuration (input)
- addr_offset[17:0]: internal address offset for read and write transfers, 64 Mwords (output)
- addr_region: address region identifier (output)
- be[3:0]: internal byte-enable (output)
- data_wr[31:0]: internal write data (output)
- data_rd[31:0]: internal read data (input)
- wr_1, rd_1: active-low write and read control (output)
- wr_ack_1, rd_ack_1: active-low write and read acknowledgements (input)

5 Performance Monitor

files: *perf_mon/perf_monitor.v, perf_mon/perfmon_cntl.fsm, perf_mon/perfmon_cntl.v*

The performance monitor collects performance data from the load balancer module and makes it available via the PCI interface. Performance data is grouped by NIDS sensor, each record includes the sensor MAC and IP address, number of packets, number of bytes and number of flow control packets, with the following format:

Byte	Description
0-1	Number of hash buckets assigned to sensor
2-7	Sensor MAC Address
8-11	Sensor IP Address
12-15	Number of flow control packets from sensor
16-23	Number of bytes routed to sensor
24-31	Number of packets routed to sensor

Table 5: Performance Data Record Structure

The performance monitor maintains an array of 64 such sensor records and updates them as directed by the load balancer. In addition, upon a command the performance monitor copies a snapshot of all records into a separate array. Finally, another external command selectively clears the performance data of all records. A change to the three bit clear code initiates the clear operation. The 0 bit clears flow control data, the 1 bit clears the byte count, and the 2 bit clears the packet count. The performance monitor operates at PCI bus frequency to simplify communication with the PCI target interface. It receives commands from the load balancer via an asynchronous FIFO.

5.1 Functionality

The load balancer issues commands to the performance monitor to update individual records. Commands are buffered in an asynchronous FIFO to synchronize between the two clock domains, and to allow buffering of commands during snapshot or reset operations. The following table summarizes the commands and arguments.

Command <1:0>	Address <9:0>	Data <15:0> <16:31	Description
01: Write	16-bit aligned	16-bits of data	Write data into address specified in command
10: Add32	32-bit aligned addr<1:0> = 00	16 bits of data	Add data to current 32-bit value at address
11: Add64_1	128-bit aligned addr<2:0> = 000	16 bits of data	Add data to current 64-bit value at even address add one to 64-bit value at odd address

Table 6: Performance Monitoring Commands

The performance monitor processes commands from the FIFO one by one. Should one of the external control signals (snapshot, clear) be active, command processing is interrupted and the external command is executed and acknowledged.

5.2 Structure

The figure below shows a simplified block diagram of the performance monitor. Commands from the load balancer are shifted into an asynchronous FIFO. A performance monitor command consists of an opcode, a 10-bit half-word address and a 16-bit data word. The FIFO synchronizes between the two clock domains and buffers command during snapshot or clear operations.

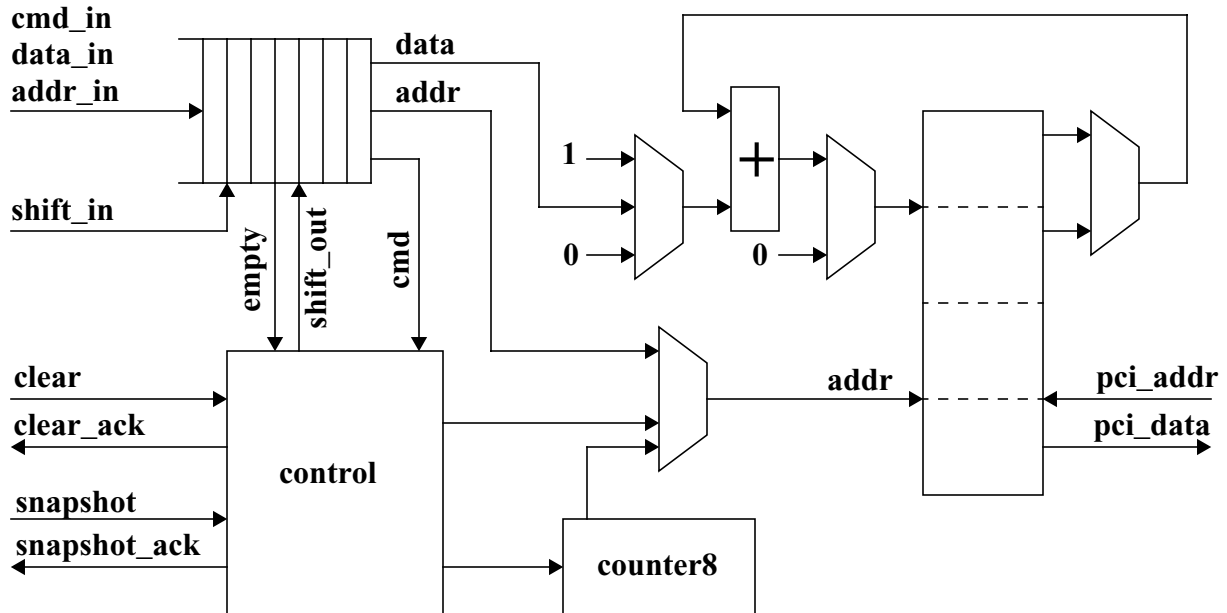


Figure 8: Performance Monitor Hardware Diagram

The performance monitor records are stored in dual-ported SRAM holding 128 sensor records. Each record consists of four 64-bit entries, thus the total memory is 4096 bytes organized as four 16-bit wide banks to support writes to individual 16-bit half-words.

After reset as well as when the clear signal is active, the performance monitor clears all records in the lower portion of the table holding the current values. In this case, the table address is provided by an 8-bit counter, and the data multiplexers are set to write zero values. During normal operation, the control state machine observes the FIFO and processes commands if it is not empty.

If the command is a 16-bit write, the lower address bits determine which write-enable signal is active while the data is broadcast to all four table banks. When writing, the 64-bit adder takes the input data and a 0 as selected by a multiplexer as operands, thus effectively passing the original data through.

For a 32-bit add command, the least-significant address bit selects both the write-enable for either two of the four banks and selects the even or odd portion of the data read from the table. The adder increments the data read from the table by one, while the input address is used as table index.

The 64-bit add command combines two operations. First, the current data is read from the even 64-bit word, the input data is added and the result is written back into the same location. Then, the adjacent odd 64-bit word is incremented by one by again reading the current value, adding one to it and writing it back.

A snapshot operation is initiated by asserting the snapshot signal. In this case, the address is provided by the 8-bit counter, while the data multiplexers are set to pass the data read from memory through to the write port. The control state machine controls the lowest address bit, alternating between 0 when reading the current value and 1 when writing it into the snapshot section. If both clear and snapshot signals are active at the same time, the state machine first performs a snapshot before clearing the lower table section. It is the responsibility of the external logic to leave either signal active until it is acknowledged.

The performance monitor selectively clears different portions of each record, specified by a three-bit clear signal. If the highest clear-bit is set, the packet count portion is cleared. Bit 1 corresponds to the byte count, and bit 0 clears the flow-control count. Any or all of these bits can be set simultaneously, the control state machine controls the write-enable signals accordingly. A clear operation starts when any of the clear-bits is one. Changing these bits while a clear operation is underway results in unspecified table contents. The second port of the table SRAM makes all performance monitoring records available to the PCI interface.

5.3 Interface

The performance monitor communicates with the PCI target and the load balancer.

- pci_clk: PCI clock
- clk: FIFO write-port clock
- pci_reset_l: asynchronous active-low reset signal, also clears all memory locations
- pci_addr[17:0]: PCI address, 32-bit aligned (input)
- pci_data[31:0]: PCI data (output)
- clear[2:0]: clear command (input)
- clear_ack: clear in progress (output)
- snapshot: snapshot command (input)
- snapshot_ack: snapshot in progress (output)
- cmd_in[1:0]: command from load balancer (input)
- addr_in[9:0]: sensor record address (input)
- data_in[15:0]: load balancer data (input)
- shift_in: load command into FIFO (input)

6 Performance Counters

Various performance counter facilities are provided to observe the behavior of the Spanids loadbalancer hardware.

6.1 32-bit Edge-sensitive Counter

*files: perf_counter/event_cntr.v, perf_counter/event_cntl.fsm,
perf_counter/event_cntr64_async.v, counters/counter32.v, counters/counter8.fsm*

The edge-sensitive performance counters are incremented for every low-high transition of the control signal. It is thus suitable to count events that are longer than one cycle, for instance entering or leaving frames. In addition to the edge-sensitive control input, the counters provide an active-high enable and a synchronous clear signal.

The 32-bit counter module consists of a synchronous 32-bit counter cascaded from four 8-bit counters. Each of the 8-bit counters is modeled as a finite state machine with 256 states, 4 *enable* inputs and a synchronous *clear* that returns the state machine to the initial zero-state. Each counter provides 8 data signals and an overflow. When cascaded, the least significant counter forwards its overflow output to the next counters enable input. All other counters receive the overflow signals of previous counter modules. In addition, each counter is controlled by the global enable and clear inputs.

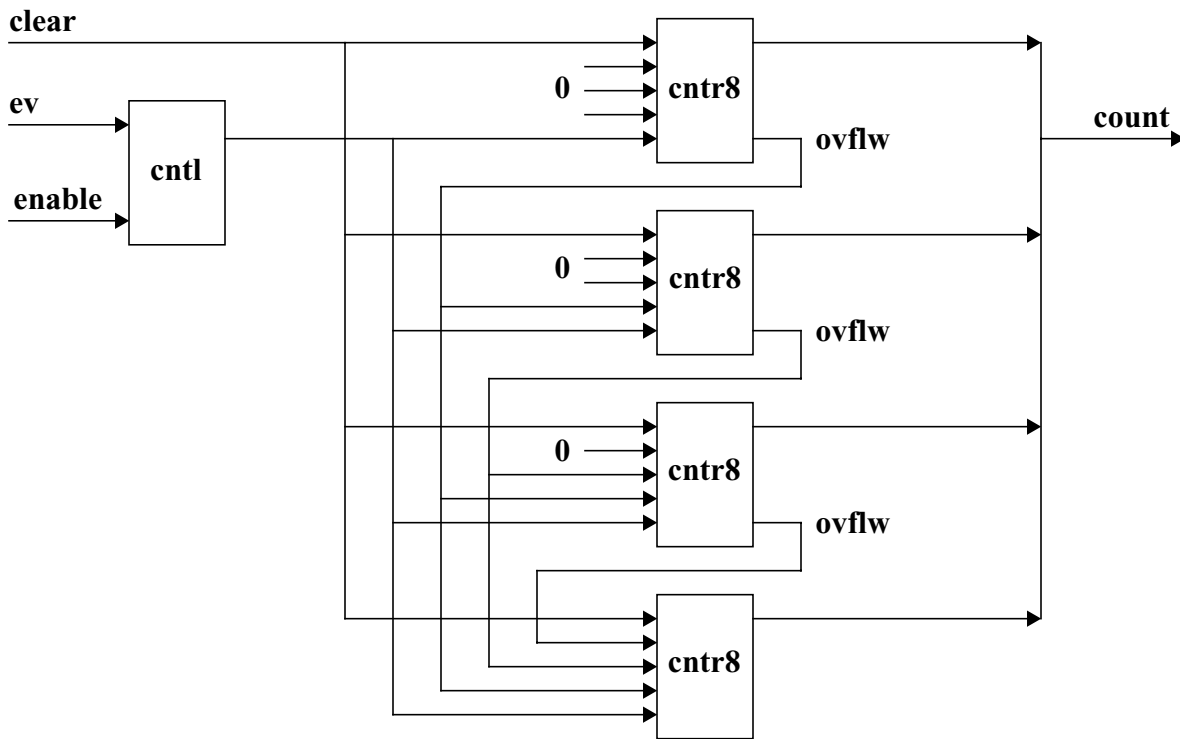


Figure 9: 32-bit Edged-triggered Counter

An event controller state machine generates a 1-cycle pulse to increment the counter if the enable input is active and the event signal transitions from low to high. It then waits for the event signal to become inactive again before responding to the next transition.

The 32-bit edged-triggered performance counter provides the following interface:

- clk: clock signal (input)
- reset: asynchronous reset, active-high (input)
- clear: synchronous active-high clear (input)
- ev: event signal, rising-edge sensitive (input)
- enable: active-high enable (input)
- count[31:0]: counter value (output)

The 64-bit asynchronous counter module consists of a 64-bit counter cascaded from two 32-bit counter modules. In addition to the larger count available, this module also has a count output register that is set to the current count by an active low latch_1 signal synchronous to a separate clk_out clock.

The 64-bit edge-triggered asynchronous performance counter provides the following interface:

- clk, clk_out: clock signals (input)
- reset: asynchronous reset, active-high (input)
- clear_1: synchronous (with clk) active-low clear (input)
- event0: event signal, rising-edge sensitive (input)
- enable_1: active-low enable (input)
- latch_1: active-low latch signal (input)
- count[63:0]: counter value (output)

6.2 Packet Byte Counter

files: perf_counter/byte_cntr.v, perf_counter/perf_cntl.fsm, counters/counter6.fsm, counters/counter7.fsm

The packet byte counter determines the size of the software-visible portion of each Ethernet frame. At a maximum frame size of 9000 bytes, excluding checksum, 14 bits are required to represent the

frame size. The internal frame format transfers 2 bytes per cycle, hence the counter is incremented by two for every cycle with a valid frame.

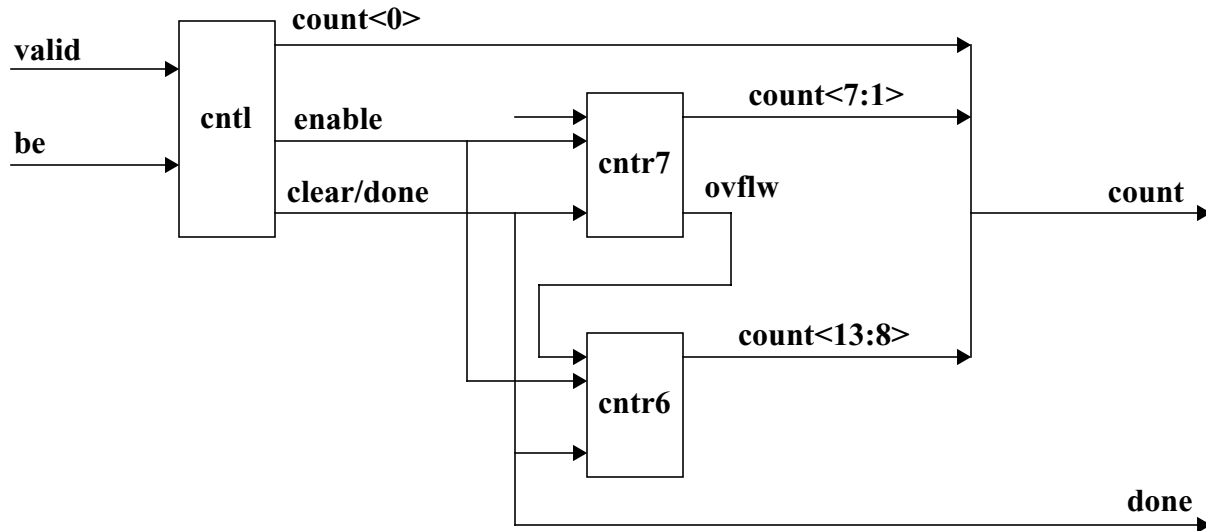


Figure 10: Packet Byte Count Module

The counter is implemented as cascaded 6-bit and 7-bit counters with enable and synchronous clear. Only bits 13 through 1 of the overall byte count are provided by the counter, the least-significant bit is controlled by a separate state machine. The same state machine observes the *valid* and *be* signals and controls the counter *enable* and *clear* signals. Four cycles after *valid* becomes active, the state machine enables the counter. If *valid* becomes inactive while *be* is high, the state machine enables the counter for one more cycle before latching the final count into an output register. If *be* is low during the last cycle, indicating an odd frame, the state machine stops counting immediately and activates the least significant bit while latching the counter value into the output register. The *done* signal also serves as a clear signal for the counter.

The packet byte counter module provides the following interface:

- clk: clock signal (input)
- reset: asynchronous, active-low reset (input)
- valid, be: active-high frame valid and byte-enable signals (input)
- count[13:0]: packet length in bytes, valid only when *done* is active (output)
- done: high for one cycle when *count* is valid (output)

7 Frame Acknowledgement

files: *frameack/frameack.v*, *counters/counter5.fsm*, *frameack/framecntl.fsm*

7.1 Introduction

The frame acknowledgement module provides a visual cue that Ethernet frames are passing through the load balancer. The module observes the valid signal and turns on an LED. Since frames on a Gigabit Ethernet link are too short to be observable by a human, the frame signal is extended to several milliseconds. While the LED is on, additional frames are ignored. Furthermore, the LED remains off for a certain amount of time before responding to the next frame to achieve a ‘blinking’ effect under load. It is important to note that in addition to visually indicating frames, the same module can be used to signal various other events such as error conditions.

7.2 Implementation

The frame acknowledgement module consists of a simple 2-state finite state machine and a 20-bit counter. The control state machine remains in the *idle* state until the *valid* signal becomes active. It then transitions into *count* state where the 20-bit counter is enabled. In this state, it observes the overflow signals from the cascaded counter. Once the counter rolls over, it returns to the idle state and is ready to signal the next event.

The 20-bit counter consists of four 5-bit counters with four enable inputs each. These counters are modeled as state machines that transition to the next state only if all enable inputs are low. Each counter produces an active-low overflow output that is used for cascading as well as to indicate rollover to the control state machine. The least significant counter is only controlled by the control state machine, the remaining enable inputs are hardwired to low (active). Each additional counter is controlled by the overflow inputs of all previous counters, with the unused enable inputs hardwired to low. In addition, all overflow signals are used to indicate a roll over to the control state machine.

The LED signal is derived from the global enable signal and the two most significant bits of the counter. The LED is on (low) only when the global enable is active (low) and when at least one of the two most significant bits are low. Consequently, the LED is on for the first three quarters of the counter’s period and off for the last quarter. For a 20-bit counter running at 62.5 Mhz, the cycle time is 16.77 milliseconds. The resulting on-time of 12.6 millisecond is easily observable.

7.3 Interface

The frameack module provides the following interface signals:

- clk: clock signal (input)
- reset: asynchronous reset signal (input)
- valid: active-high signal to start signalling an event
- led: active-low output to drive external LED

8 Frame Latch

*files: frameack/framelatch.v, counters/counter6.fsm, counters/counter7.fsm,
memories/ramd_async.v*

8.1 Description

The frame latch module captures a complete frame in memory and makes it available for retrieval via the PCI interface. It is intended to aid in debugging the load balancer operation. The frame storage is composed of two 16-bit wide and 4096-entry deep memory blocks that are written with the even and odd frame word. A 13-bit counter, composed of cascaded 6-bit and 7-bit counter modules, is incremented for every cycle the input data is valid. The least-significant bit controls writing to the even and odd memory block, while the remaining bits index into the memory blocks. The counter is cleared when the valid signal is inactive.

Both memory blocks can be accessed via a secondary address to read 32-bit words. In addition, the last counter value is latched into a length register. To correctly latch the last counter value, the valid signal is delayed by one cycle to control the length latch. Note that this implementation ignores odd frame sizes and always rounds up to the next even byte count.

8.2 Interface

- clk: clock signal for frame data (input)
- reset: asynchronous reset (input)
- data: 16-bit frame data (input)
- valid: frame valid signal (input)
- pci_clk: clock signal for secondary (PCI) interface to memory (input)
- pci_addr: 12-bit secondary word read address (input)
- pci_data: 32-bit secondary data (output)
- pci_length: 32-bit frame length in bytes (output)

Section III: Load Balancer

1 Load Balancer Overview

files: *loadbalancer/loadbalancer.v*

The load balancing and routing stage is at the core of the Spanids prototype. It receives all incoming frames in the internal format (without checksum) and produces a new destination MAC address to control subsequent routing of frames. Currently, only a small subset of the router functionality is implemented, namely the frame decoder and latch control.

1.1 Structure

The overall architecture of the load balancing stage currently assumes a packet header decode and latch module, a flow-control receiver module and various not yet specified modules to perform initialization, load balancing and routing. Each module will be described in more detail in separate sections.

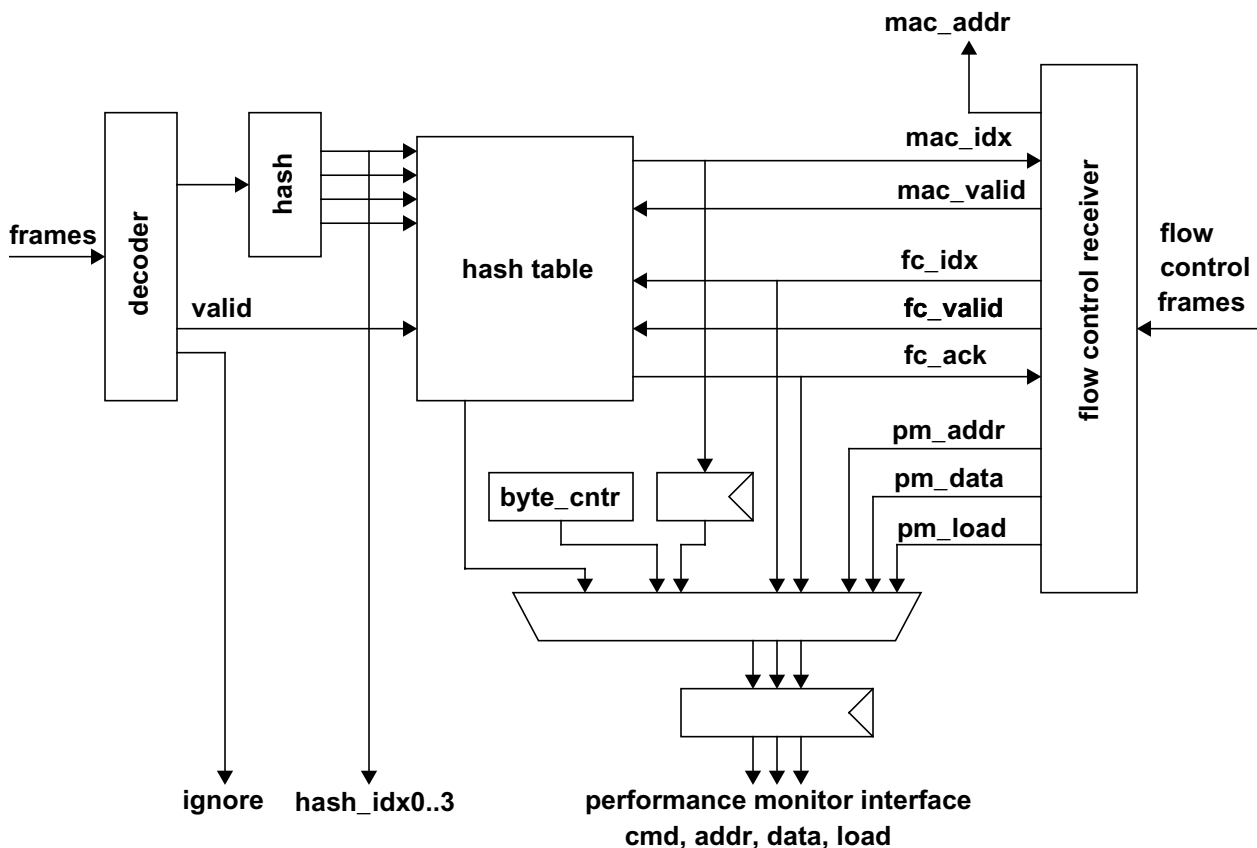


Figure 11: Load Balancer Organization

Key global signals include a 96-bit frame header register with associated valid signal and the flow control index with valid-signal. The decoded header includes the last two IP header words (not

including the IP-options field), which contain the source and destination address. In addition, the first payload word is provided, which in case of TCP and UDP contains the source and destination port. An active-high valid signal indicates a valid header, it is active for one cycle after decoding is complete. The decoder currently ignores any Ethernet packet that is not of an IP type.

The decoded and latched packet header is passed to a hashing module that produces four 12-bit hash indices. A 12-bit mask effectively reduces the number of valid index bits depending on the number of sensors. For sensors counts of 32 and above, all index bits are valid. For counts between 16 and 31, the lower 11 bits are valid, and so on. Finally, for 2 or 3 sensors, only 8 index bits are valid, and for 1 or 0 sensors the entire index is forced to 0. As a result, the load balancer does not forward packets when no sensor is registered, since the hash index is forced to 0 and the destination MAC addresses are not rewritten.

The flow-control index specifies which sensor issued a flow control signal, it is an index into the internal sensor table. The index is identical to the one produced by the load balancer when rewriting MAC addresses.

The load balancer also connects to the performance monitor via a FIFO interface. Performance monitoring data consists of a command, 16-bit aligned address, 16-bit data. An active-high load signal shifts the command into the FIFO. Performance monitoring data comes from four sources. During initialization, the flow control receiver writes the MAC and IP address of each attached sensor in 5 consecutive cycles. Next, after completely initializing the hash tables, the hash table module writes the number of hash buckets assigned to each sensor to the performance monitoring array. During normal operation, When a flow control index is acknowledged (and removed from the flow control FIFO), that index is also written into the performance monitor FIFO along with the proper command. Finally, for each outgoing frame, the frame length in bytes and destination index is written. In case of a conflict between outgoing frame and flow control data, the flow control data is written first followed by the frame data. A simple state machine observes both the frame length and MAC rewrite index and only loads the frame data when both are valid.

1.2 Interface

The loadbalancer receives both incoming frames as well as flow control frames. It produces a new MAC address and interfaces to the performance monitor. In addition it provides control signals for a variety of performance counters.

- clk: clock, synchronous to transmit PHY
- reset: asynchronous reset signal, driven by initialization logic
- data_i[15:0]: 16-bit frame data, not including checksum (input)
- valid_i, be_i: frame valid and byte-enable signals (input)
- data_f[15:0]: flow control and configuration frame data (input)
- valid_f, be_f: flow control and configuration frame valid and byte-enable (input)
- init_start: signals end of initialization-request phase, triggers load balancer configuration (input)

- `init_done`: signals end of initialization phase (input)
- `no_routing`: disables load balancer operation (input)
- `mac_addr[47:0]`: new destination MAC address (output)
- `mac_load`: signals valid MAC address (output)
- `packet_ignore`: indicates an ignored (non-IP) packet (output)
- `sensor_count[6:0]`: number of sensors currently signed on (output)
- `pulse`: periodic pulse to convert counts into rates (input)
- `threshold[7:0]`: bucket intensity threshold for intensity-based promotion policy (input)
- `bucket_count[3:0]`: number of hash buckets to move or promote when load balancing (input)
- `random_val[4:0]`: random value (input)
- `lb_mode[2:0]`: move/promote policy (input)
- `perfmon_cmd[1:0]`: performance monitoring command (output)
- `perfmon_addr[9:0]`: 16-bit aligned performance monitoring address (output)
- `perfmon_data[15:0]`: performance monitoring data (output)
- `perfmon_load`: active-high valid/load signal (output)
- `last_hash0[11:0]` - `last_hash3[11:0]`: last hash index for levels 0 - 3 (output)
- `move`, `promote`, `demote`: indicates a hash bucket has been moved, promoted or demoted, active for one cycle (output)
- `route_10`, `route_11`, `route_12`, `route_13`, `route_14`: indicates a packet has been routed through hash level 0, 1, 2 or 3, active for one cycle (output)
- `pci_clk`: PCI interface clock (input)
- `pci_addr[11:0]` word-aligned PCI address (input)
- `pci_data_table0[31:0]` - `pci_data_table3[31:0]`: data from hash table addressed by `pci_addr` (output)
- `mult0_op0[31:0]`, `mult0_op1`, `mult1_op0[31:0]`, `mult1_op1[31:0]`: operands for multiplication in intensity-based policy (output)
- `mult0_res[63:0]`, `mult1_res[63:0]` : result of multiplication (output)

2 Packet Decoder

files: loadbalancer/decoder.v, loadbalancer/decodecntl.fsm

2.1 Description

The packet decoder decodes and latches incoming packets for further processing by the load balancer. A state machine observes the frame signal and controls a 96-bit latch that holds the parts of the packet header used by the hash functions. The state machine skips the packet preamble and MAC addresses and then decodes the Ethernet type field. In case of a SNAP or VLAN header, it skips an additional two words and then decodes the Ethernet type. If the frame is not an IP packet, the state machine signals an ignored packet and waits for the end of the frame. Otherwise, it latches the IP header length in an auxiliary register and latches the 4th and 5th IP header words into the decoder register. These words contain the IP source and destination address used by the hash functions. The state machine then skips any additional IP header words, if any, as indicated by the IP header length. Finally, the state machine latches the first word of the IP payload, which contains the source and destination port in case of UDP or TCP packets. Once done, the state machine raises the header-valid signal for one cycle to indicate that a complete and valid header has been latched and is ready for processing.

2.2 Interface

- clk: clock, synchronous to transmit PHY
- reset: asynchronous reset signal, driven by initialization logic
- data_i[15:0]: 16-bit frame data, not including checksum (input)
- valid_i, be_i: frame valid and byte-enable signals (input)
- packet_header[95:0]: latched portion of packet header - src IP, dest IP, ports (output)
- packet_valid: active-high valid signal (output)
- packet_ignore: indicates an ignored (non-IP) packet (output)

3 Hash Functions

files: loadbalancer/hashers.v

3.1 Description

The hash module produces four different 12-bit hash functions based on a 96-bit input value derived from the packet header. Hash indices are computed synchronously and are masked with a 12-bit mask value, thus effectively scaling the hash value range depending on the number of active sensors. The first-level hash function (idx0) computes the XOR of consecutive 12-bit chunks of the input value. The second hash function uses a similar approach, but shifts the initial bit and also splits some 12-bit chunks over multiple sections of the input bit field. The third hash function hashes the IP addresses in 12-bit chunks with the sum of the port numbers, and the fourth and final hash function hashes the port numbers with the sum of the IP addresses.

3.2 Interface

- clk: clock signal
- header[95:0]: packet header to be hashed (input)
- mask[11:0]: bit mask to scale hash functions depending on number of sensors (input)
- idx0[11:0], idx1[11:0], idx2[11:0], idx3[11:0]: hash function output (output)

4 Hash Table

files: *files: loadbalancer/hash_table.v loadbalancer/table_cntl.fsm
loadbalancer/table_clear.fsm loadbalancer/pulse_cntl.fsm*

4.1 Overview

The main routing hash table consists of 4 SRAM arrays of 4096 entries each. Each entry records the current destination sensor as a 6-bit value, the packet count since the last clear operation (24 bits), a promotion-bit indicating that the next level of hashing is to be used for routing and a 5-bit timeout counter. A fifth level of routing consists of a counter that is incremented every time a routing decision reaches this level. The counter wraps around when reaching the number of sensors currently signed up, thus effectively implementing a round-robin packet distribution scheme among all sensors. The contents of all hash tables is accessible to the PCI interface for debugging purposes.

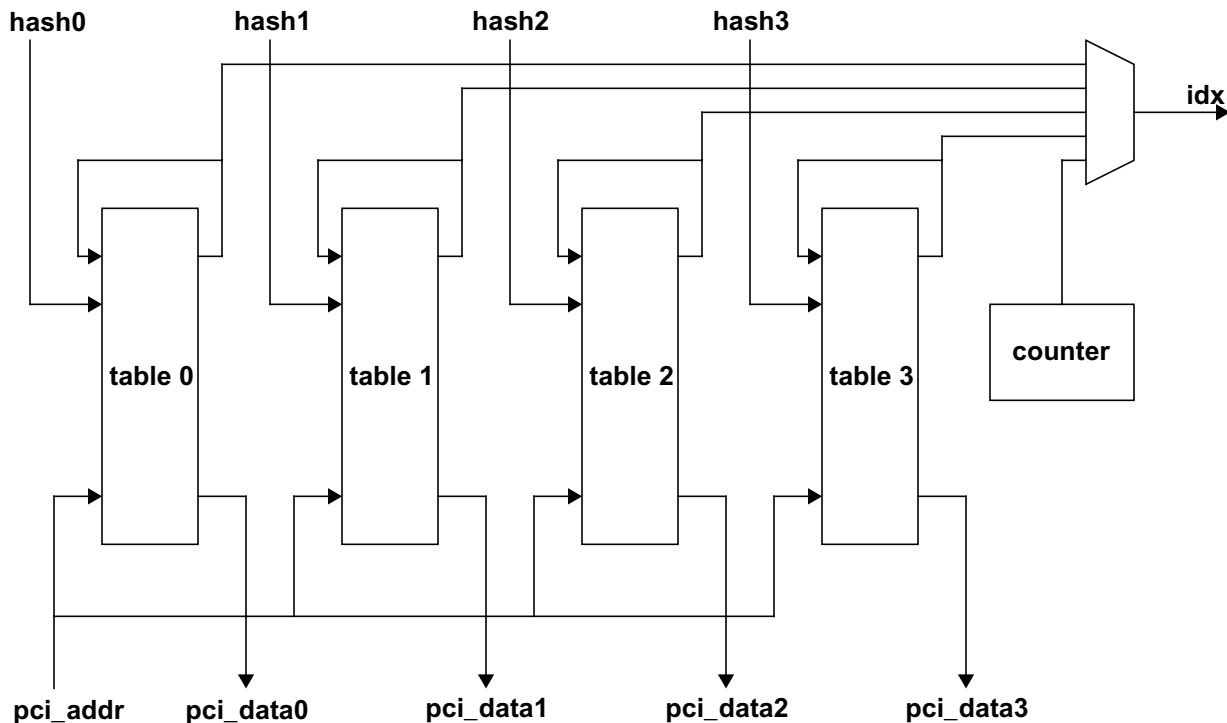


Figure 12: Hash Table Overview (partial)

During the initialization stage, after all sensors have signed on and the current sensor-count is established, the hash tables are initialized with the initial destination sensors and promote-bits set to 0. Depending on the number of sensors, only the subset of the hash table actually used will be written. Destination sensors are assigned by a counter starting at zero and wrapping around when reaching the number of sensors. For example, given 3 sensors hash buckets 0, 3, 6, 9 and so on are assigned to sensor 0. This scheme achieves a nearly even distribution of buckets over all sensors in all cases. Each time a bucket is assigned to a destination sensor, the per-sensor packet count table

is also updated. At the end of the initialization sequence, the four independently running address counters wait for each other to finish and then trigger writing of the current sensor bucket counts by the per-sensor bucket count table to the performance monitor.

4.2 Routine

During normal operation, a central control state machine performs the routing of each packet. When a valid packet is received and the header is latched, the state machine performs a read operation of the first hash table with the hash value as address. In the subsequent cycle, the same table entry is written with the packet count incremented by one. In the same cycle, the next hash table is read. Finally, while writing the current entry, the state machine inspects the promote bit to determine whether the next hash level needs to be used or not. When finding a non-promoted hash bucket, the routing state machine latches the most recent sensor index and signals completion of the routing. The flow control receiver (external to this module) then looks up the destination MAC address and provides it to the delay pipeline where it is inserted into the packet header. When reaching the fourth level of hashing, the state machine uses the round-robin counter value as sensor index and increments the counter by one. A three-bit level signal indicates the hash level used for the current routing decision. This signal control the index multiplexer, the external counters that count packets routed through the different levels as well as the hot list and per-sensor packet count table.

4.3 Table Clear

Periodically, all hash table entries are inspected to compute packet rates and to potentially demote previously promoted hash buckets. Scanning the table entries is triggered by the external pulse counter that produces a one-cycle pulse at a programmable rate between 4 times a second and every 8 seconds. The same address counters performing the initialization are used to read each table entry in turn. Following a read, the packet count is shifted right by one bit, thus dividing it by 2, and written back into the hash bucket entry.

If a promoted hash bucket is encountered, the clear-control state machine allows an additional 5 cycles to determine whether the bucket can be demoted, and then writes the new bucket entry back. Promotion and demotion is described in more detail in a later section. Since the table scan competes with routing and feedback processing for memory accesses, different tables may be cleared at different rates. When done scanning the current table, the clear-control state machines wait for all other tables to complete their scan before driving a global clear-done signal. This signal then triggers clearing of the hot list and shifting of all entries in the per-sensor packet count table.

4.4 Synchronization

Three entities compete for access to an individual hash table: the central routing state machine, feedback processing, table clear and the initialization control. Each hash table arbitrates among these requests independently using a strict priority scheme. The SRAM addresses and write data as well as the write-enable signals are selected by a prioritized multiplexer, as follows:

- init: clear-addr and clear-data from counters, else
- routing: hash index as address, previous bucket value with packet count incremented as data
- clear: clear-addr and previous bucket value with packet count shifted right by one bit, timeout value decremented if not zero
promote-bit is cleared when demoting a hash bucket
- flow-control: bucket index from hotlist as address, new sensor index from per-sensor packet count table (cold sensor list) if not promoting, otherwise promote bit and timeout value from random-number generator

The various control state machines observe each others requests and stall until higher-priority accesses are complete. During the initialization phase, no other accesses happen and consequently the init-start signal has highest priority. During normal operation, routing accesses indicated by the central routing state machine have highest priority. As a result, this state machine does not need to be aware of competing accesses, it simply signals its own access when it reads or writes a table. The table-clear state machines have next highest priority. These state machines observe the routing request signals and restart the clear operation for the current hash bucket if a conflict is detected. Restarting requires performing the read operation again, regardless whether the read or write access encountered a conflict. Flow control processing has the lowest priority since it is expected to happen relatively infrequently, and the flow control receiver provides ample buffering of flow control indexes. The flow control receiver thus observes both the routing and clear busy signals to detect table contention.

As an additional level of synchronization, flow control processing and table scanning (clear) are mutually exclusive. Flow control processing requires a consistent hash table and hot list state which can not be ensured while the tables are scanned and cleared. Flow control processing thus stalls until the current table scan is complete. Conversely, clearing the table is delayed until the current flow control operation is complete. The periodic pulse is latched by a state machine and held until the clear operation has started, while the clear state machines observe the flow control operation and stall until it is complete.

In addition to the hash tables, access to the hot list is shared between the flow control state machine and the internal hot list controller. Clearing the hotlist is an iterative process that must be completed before a new flow control message can be processed. The hot list indicates when it is busy, and the flow control state machine stalls until the hotlist becomes available.

Lastly, access the two multiplier units and the per-sensor packet count and per-sensor bucket count tables is shared between the table clear operations and the flow control handler. If a promoted bucket is encountered, it is evaluated for possible demotion. When using the intensity heuristic, this requires access to the per-sensor packet and bucket count tables and the multiplication units. Similarly, when determining whether to promote a bucket after a flow control packet, the flow control handler requires access to the same resources. Again a strict priority scheme is used with a multiplexer selecting the sensor index for the per-sensor packet and bucket count tables and selecting the packet count as input into one of the multipliers. Table 0 has the highest priority,

followed by tables 1, 2 and 3 and lastly the flow control handler. Each state machine observes the request signals of the higher-priority entities and stalls until the resources become available.

4.5 Interface

The hash table module receives four hash indexes and flow control indexes from the flow control receiver FIFO, and produces a destination sensor index. It also interfaces to the performance monitoring FIFO and provides a large number of debugging and testing signals that connect to the PCI interface. Finally, a number of policy configuration signals are provided from the PCI interface.

- clk, reset: clock and reset signal (input)
- init_start: signals start of hash table initialization after all sign-on messages have been received (input)
- init_done: signals reverting to normal operation after initialization is complete (input)
- sensor_count[5:0]: number of sensors signed on (input)
- pulse: periodic pulse (input)
- threshold[7:0]: promotion threshold for intensity-based heuristic (input)
- buckets[3:0]: number of buckets affected when processing flow control information (input)
- random_val[4:0]: random number between programmable min and max (input)
- lb_mode[2:0]: move/promote heuristic (input)
- move_bkt, promote_bkt, demote_bkt: signal move, promotion or demotion of one bucket to external counters (output)
- route_l0, route_l1, route_l2, route_l3, route_l4: signal routing of a packet through the respective level of hashing (output)
- hash0[11:0], hash1[11:0], hash2[11:0], hash3[11:0]: hash values produced from current header (input)
- hash_mask[11:0]: hash value mask depending on number of active sensors (input)
- lookup: triggers routing operation, indicates hash values are valid (input)
- sensor_idx[5:0]: destination sensor for current frame (output)
- load_idx: sensor_idx is valid, signals completion of routing (output)
- fc_idx[5:0]: sensor index for current flow control operation (input)
- fc_val: fc_idx is valid, triggers flow control processing (input)
- fc_ack: acknowledge processing of flow control information, shifts data out of FC fifo (output)
- perfmon_addr[9:0]: performance monitor address (output)
- perfmon_data[15:0]: performance monitor data (output)
- perfmon_load: load performance monitor address and data into buffer (output)
- perfmon_busy: performance monitor interface busy, signals contention (input)

- pci_clk: PCI interface clock signal (input)
- pci_addr[11:0]: PCI interface word address (input)
- pci_data0[31:0], pci_data1[31:0], pci_data2[31:0], pci_data3[31:0]: hash table data for PCI interface (output)
- mult0_op0[31:0], mult0_op1: most recent operands for first multiplier module, latched (packet rate and sensor bucket count) (output)
- mult0_res[63:0]: most recent result of first multiplier module (output)
- mult1_op0[31:0], mult1_op1: most recent operands for second multiplier module, latched (sensor packet count and threshold) (output)
- mult1_res[63:0]: most recent result of second multiplier module (output)

5 Sensor Feedback Processing

files: loadbalancer/

Flow control processing is at the core of the Spanids load balancer. Feedback frames received from one of the sensors are decoded and converted into a sensor index by the flow control receiver. The flow indexes are buffered in a FIFO that constitutes the interface to the flow control processing logic inside the hash table.

The hot list maintains a list of the 16 most active hash buckets per sensor. It is updated with a new packet count every time a packet is routed. For each feedback message, a programmable number of hash buckets is either moved to another sensor, or promoted to the next level if hashing. The per-sensor packet table maintains a the total packet counts for all sensors, as well as a list of the four least busy sensors. When moving hash buckets, they are reassigned to the least busy sensor.

Five policies exist to decide whether a hash bucket is moved or promoted. The policy is controlled by an external register and can be changed dynamically via the PCI interface.

- always-move: move buckets to alternate sensor
- always-promote: always promote buckets to the next level of hashing
- random: use random number value to decide whether to move or promote
- intensity: promote if hottest bucket receives more then N times the average per-bucket packet rate for that sensor, where N is a programmable threshold
- static: never adjust any buckets

Note that all of these policies affect the entire flow control operation. For instance, when randomly deciding to move, all buckets affected by the current flow control operation are moved.

Hash buckets promoted to the next level are periodically evaluated for possible demotion. At the time of the initial promotion, a random time out value is assigned to the bucket and stored in the table entry. Triggered by a periodic pulse, the clear state machines scan all hash tables, shift the packet counts by one bit to produce a weighted average rate, and check all promoted hash buckets. For each promoted hash bucket, the time out value is decremented by one. If it is zero and the bucket is still promoted, it is evaluated for possible demotion according to the promotion policy.

- always-move: no bucket ever promoted, does not apply
- always-promote: bucket is demoted unconditionally
- random: bucket is demoted unconditionally
- intensity: demote if bucket intensity is less then N times the average per-bucket packet rate for that sensor, where N is a programmable threshold
- static: does not apply

Note that for all but one policy, promoted buckets are always demoted when the timeout expires. For the intensity-based policy, the same calculation is performed that lead to the original promotion. If the bucket remains promoted, a new random timeout value is assigned.

6 Flow Control Receiver

files: *loadbalancer/fc_rcv.v, loadbalancer/fc_rcv_cntl.fsm, loadbalancer/perfmon_load.fsm, counters/counter6.fsm, counters/counter7a.fsm, memory/ramd.v*

The flow-control receiver module receives, decodes and partially handles all flow control and initialization frames received from any of the sensors. During the initialization phase it builds an internal table of attached sensors. Other load balancer modules index into the table to determine the MAC address of a sensor. Internally, the flow control receiver performs an associative lookup when receiving a flow control frame to determine the index associated with a MAC address.

6.1 Structure

At the core of the flow control receiver is a multi-ported 64-entry register array that stores the MAC addresses of all attached sensors. The array is written during the initialization phase. A 6-bit index counter provides the address of the next available entry, and also exports the total number of sensors. When a valid init-reply packet is received, the MAC address transmitted in the frame payload is written into the array and the valid-bit is set, while the index counter is incremented. In addition, the MAC and IP address of the current packet are loaded into the (external) performance monitoring FIFO during 5 consecutive cycles, along with the appropriate ‘write’ command.

During normal operation, external modules can look up MAC addresses by providing an index. The corresponding MAC address is provided synchronously at the next rising clock edge. In addition, the module responds with the valid bit for that index to allow external modules to check that a valid entry is referenced.

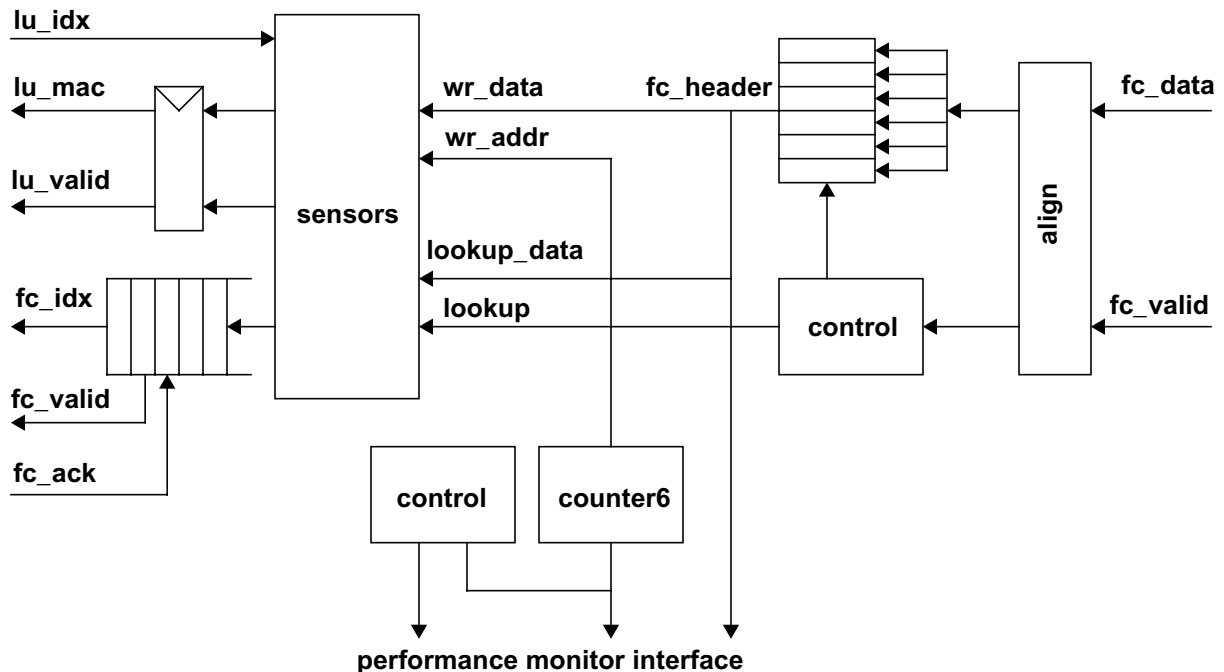


Figure 13: Flow Control Receiver Organization

Incoming packets are first aligned at a 16-bit boundary and then stored in a 96-bit register that contains the 16-bit op-code, MAC address and IP address. Alignment is necessary to account for the variable-length Ethernet preamble. As a result, the proper frame data may start at the even or odd byte. To simplify decoding, the alignment module switches even and odd bytes while delaying the odd byte by one cycle if it detects the SFD token in the even slot. Otherwise, the frame passes unmodified. Packets are partially decoded to ensure that only valid IP/UDP frames that are addressed at the load balancer MAC and IP address and UDP port number are processed.

When receiving a flow control packet, an associative lookup of the MAC address transmitted in the payload is performed and the resulting index is shifted into a synchronous FIFO. The associative lookup is implemented as a two-stage pipeline. The first stage compares the incoming MAC address with all 64 entries in parallel and produces a bit vector of matches. Only entries with the valid-bit set are considered during comparison. The second stage converts the one-hot encoding into a binary value that corresponds to the index. Note that this design assumes that exactly one table entry matches any incoming MAC address. The flow control index FIFO is a core generated by the Xilinx implementation tools.

6.2 Interface

The flow control receiver module interfaces with the receive-section of the sensor-side PHY, provides lookup and flow-control signals to other load balancer modules and connects to the performance monitoring subsystem.

- clk: clock, synchronous to transmit PHY
- reset: asynchronous reset signal
- data_i[15:0]: 16-bit frame data, not including checksum (input)
- valid_i, be_i: frame valid and byte-enable signals (input)
- data_f[15:0]: flow control and configuration frame data (input)
- sensor_count[5:0]: number of attached/initialized sensors (output)
- fc_idx[5:0]: index of sensor issuing flow control (output)
- fc_valid: flow control index valid, active high (output)
- lu_idx[5:0]: index to lookup MAC address (input)
- lu_mac[47:0]: MAC address corresponding to lu_idx (output)
- lu_valid: MAC address valid (output)
- perfmon_addr[9:0]: performance monitoring address (output)
- perfmon_data[15:0]: performance monitoring data (output)
- perfmon_load: active-high performance monitoring data valid/load signal (output)

7 Sensor Packet Rate Table and Cold List

files: *loadbalancer/sensor_packets.v, loadbalancer/sensor_packets_cntl.fsm*

7.1 Description

The sensor packet rate table maintains a table of packet counts per sensor. These packet counts can be interpreted as rates if they are periodically cleared. The table is implemented as a memory array of 64 entries of 24 bits each. After reset, the control state machine clears the entire table. Table entries are updated/incremented by driving a sensor index on *sensor_up* and asserting *update* for one cycle. The sensor index is then latched in an internal register, the corresponding entry is read in the following cycle and written with the original value incremented by one in the last cycle. The update operation takes thus two cycles, and a new sensor entry can be incremented every two cycles.

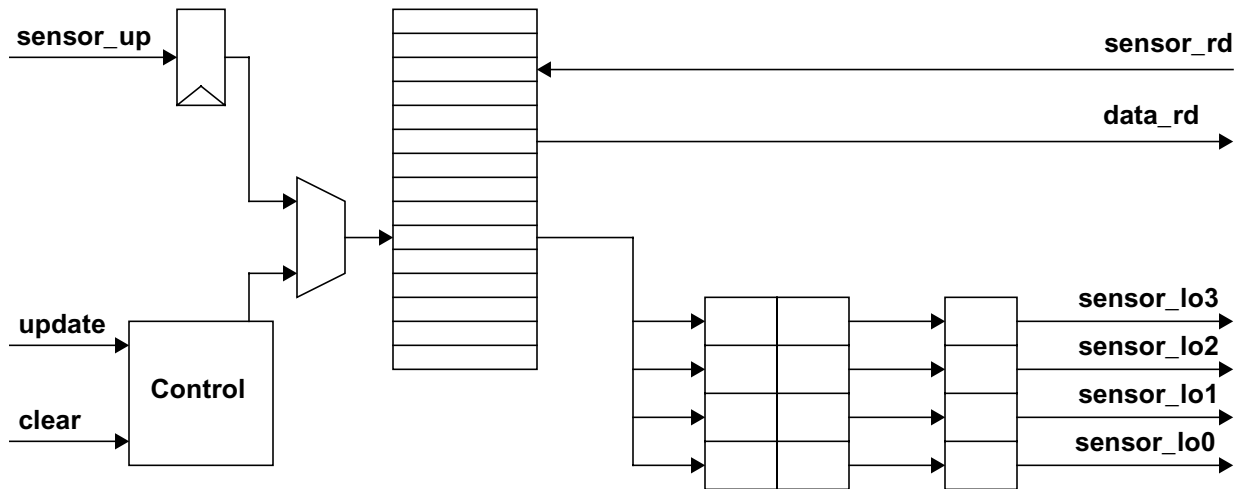


Figure 14: Packet Rate Table Organization

The control state machine continually scans or reads the table (up to the maximum sensor count) to update the cold list of the least busy sensors. The state machine is stalled when an update is in progress, otherwise it continually presents new addresses to the table. The actual cold list is implemented as a double-buffered sorted list with four entries. Each entry consists of a 6-bit sensor index and an intensity value. When a new scan cycle starts, the internal working list is initialized with 0xFFFFF to represent the highest possible packet rate. For each valid read (a read not overwritten by an update), each hot list entry compares the table value to its current intensity. If the new intensity is less than the stored value, that table entry may be updated. A priority decoder determines where in the cold list a new entry is inserted, if at all. In case of equal values, the hot-list entry is shifted up and replaced by the new value and sensor ID.

When a scan-cycle is complete, the internal working hot list is latched into the externally visible hot list. This hot list only stores the sensor indexes and not the intensities. Updating of the external list can be disabled by asserting *lock_lo*, for instance if the feedback controller is currently

inspecting the cold list. Note that this scheme results in a somewhat imprecise cold list, since table entries may be updated at any time, including after they have been scanned in the current cycle. However, given that individual entries change only by one for each update, the loss in precision is insignificant.

The packet rate table can be cleared at any time by driving *clear*, however the clear cycle only starts at the end of the current scan cycle. The clear signal is latched into a RS-flipflop and thus needs to be active for only one cycle. When completing the current scan cycle, the control state machine starts clearing the table and resets the internal flipflop. The cold list will be updated after the clear and the first scan cycle complete.

Finally, any table entry can be read via the secondary memory port. Presenting a sensor index at *sensor_rd* results in the corresponding packet intensity appearing on *data_rd* at the next clock edge.

7.2 Interface

- *clk*: clock, synchronous to transmit PHY
- *reset*: asynchronous reset signal, driven by initialization logic
- *sensor_count*[6:0]: number of active sensors (input)
- *clear*: initiates clearing the table (input)
- *sensor_up*[5:0]: sensor index to be updated/incremented (input)
- *update*: trigger update of sensor (input)
- *sensor_rd*[5:0]: secondary read port, sensor index (input)
- *data_rd*[23:0]: packet rate of sensor (output)
- *sensor_lo0*[5:0]: least-busy sensor (output)
- *sensor_lo1*[5:0]: second least-busy sensor (output)
- *sensor_lo2*[5:0]: third least-busy sensor (output)
- *sensor_lo3*[5:0]: fourth least-busy sensor (output)

8 Sensor Bucket Count Table

files: *loadbalancer/sensor_buckets.v*, *loadbalancer/sensor_buckets_cntl.fsm*

8.1 Description

The sensor bucket count table maintains the number of level-0 hash buckets currently assigned to each sensor. It provides two means to update the table, and a secondary port to read any table entry. In addition, it interfaces to the performance monitor to update the sensor bucket count.

The control state machine clears the table after reset, and then awaits commands. During the load balancer initialization, buckets are assigned to sensors in an initial round-robin scheme. Each time a bucket is assigned to a sensor, that sensor number is provided on *incr_idx* and the *incr* signal is driven. The state machine then increments that sensor's bucket count in the table. The increment operation takes two cycles and cannot be pipelined, hence buckets can be assigned to sensors at a maximum rate of one every two cycles.

The entire table contents can be dumped into the performance monitor buffer FIFO by asserting *dump* for one cycle. The control state machine reads all entries and writes the corresponding values to the performance monitor interface. If the *pm_busy* signal is active during this time, indicating that the performance monitor interface is used by another component, the state machine stalls.

During normal operation, hash buckets are moved between sensors. For such move operations, the sensor bucket table is updated by providing the old and new sensor index on *move_down_idx* and *move_up_idx* and asserting *move* for one cycle. A move operation takes a minimum of four cycles and is implemented as two read-write sequences. When writing the new value, the performance monitor is also updated. If *pm_busy* is active at this time, the state machine stalls.

Finally, any table entry can be read at any time via a secondary interface. Driving a sensor index results in the corresponding bucket count after one cycle.

8.2 Interface

- *clk*: clock signal
- *reset*: asynchronous reset signal, driven by initialization logic
- *incr_idx*[5:0]: sensor index to increment (input)
- *incr*: initiate increment operation (input)
- *dump*: initiate writing table contents to performance monitor (input)
- *move_up_idx*[5:0]: sensor index to be incremented during move operation (input)
- *move_down_idx*[5:0]: sensor index to be decremented during move (input)
- *move*: initiate move operation (input)
- *read_idx*[5:0]: sensor index to read on secondary port (input)
- *read_val*[11:0]: bucket count corresponding to *read_idx* (output)
- *pm_addr*[9:0]: performance monitor address (output)

- pm_data[15:0]: performance monitor data (output)
- pm_load: load performance monitor FIFO with addr/data (output)
- pm_busy: performance monitor FIFO busy (input)

9 Sensor Hot List

files: *loadbalancer/hotlist.v, loadbalancer/hotlist_cntl.fsm*

9.1 Description

The sensor hot list maintains a list of the 16 hottest buckets (with highest packet rates) for each sensor. The list is arranged as memory array of 16x64 entries, with each entry containing a 12-bit bucket number and a 24-bit intensity. The hot list is updated with new bucket intensities, and can be cleared. The control state machine in conjunction with the base address counter clears the entire table after reset, and on demand when the *clear* signal is active. The *clear* signal is latched internally and needs to be active for only one cycle.

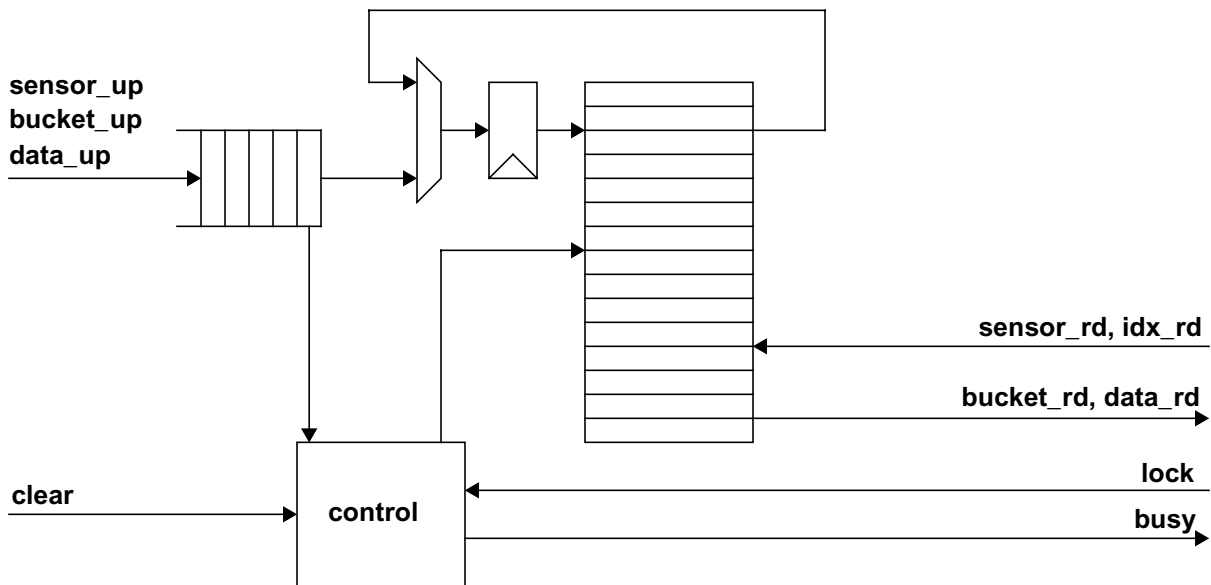


Figure 15: Hotlist Structure

Updates of the hot list are initiated by shifting a sensor index, bucket number and bucket intensity into the command FIFO. The command FIFO is required to queue commands in case the state machine is either clearing the entire list, or is locked out due to external accesses. While an individual update takes only 32 cycles, lock-outs can extend this time past the minimum packet duration.

The control state machine observes the command FIFO, and starts an update operation when it is not empty. The sensor index forms the base address for all memory references. The state machine reads and possibly writes each hot list entry, beginning from the first (hottest). Data read from the list is compared with the value stored in a register. Initially, the register is loaded with the new intensity, but during scanning it is replaced with values that have been overwritten. In the write state (after reading the existing entry), the state machine compares the old intensity with the new value. If the new intensity is higher, it will be written into the hot list entry, while the previous value is latched into the working register. In the next cycle, the replaced value will be compared to the

next entry and will overwrite it, with that previous entry being latched into the working register. As a result, once the new intensity has overwritten a hot list entry, the remaining entries are shifted down.

At the same time, each hot list entry's bucket number is compared to the new bucket number. If the new bucket number matches an entry found in the hot list, shifting stops at this entry, thus eliminating duplicate bucket number in the hot list. When reaching the bottom of the hot list, or a duplicate entry, the state machine shifts the current command from the input FIFO.

An external interface provides access to any hot list entry. Presenting a sensor index and a hot list index results in the bucket number and intensity appearing at the output after one cycle. This interface synchronizes with the control state machine. The *lock* signal indicates the intention of an external component to read hot list entries. The state machine will not start a new update or clear operation if that signal is asserted, but it finishes its current operation. Thus, the external component must observe the *busy* signal to determine when it is safe to read entries. Note that the *lock* signal can stall the state machine between clearing individual hot lists. The synchronization thus happens at the individual hotlist granularity. When lock is asserted and acknowledged with busy being inactive, no changes will happen to any hotlist. However, some hotlist may be cleared, while others still contain the previous values.

9.2 Interface

- clk: clock signal
- reset: asynchronous reset signal, driven by initialization logic
- clear: initiate clearing of entire hot list (input)
- busy: state machine is currently updating or clearing a hot list (output)
- sensor_up[5:0]: sensor index to be updated (input)
- bucket_up[11:0]: bucket number to be inserted (input)
- data_up[23:0]: intensity of bucket to be inserted (input)
- update: load sensor_up, bucket_up and data_up into FIFO (input)
- sensor_rd[5:0]: sensor index for external accesses (input)
- idx_rd[3:0]: hotlist entry for external accesses (input)
- bucket_rd[11:0]: bucket number in hotlist entry corresponding to sensor/idx (output)
- data_rd[23:0]: intensity in hotlist corresponding to sensor/idx (output)
- lock: request atomic access to a hotlist (input)

10 Hash Table Feedback Processing

files: loadbalancer/fc_cntl.fsm

The core feedback processing happens inside the hash table module. The number of the sensor transmitting a feedback packet is buffered in a FIFO by the feedback receiver and provided to the feedback controller. The controller first shifts the oldest entry out of the FIFO to make the feedback sensor index available. If a table clear operation is in progress at this time, the controller stalls. Similarly, if the hotlist is busy clearing its internal table, the feedback controller stalls.

As soon as the feedback FIFO is not empty, the feedback controller asserts control signals requesting access to the hotlist and per-sensor packet and bucket count tables. It also locks the hotlist to prevent updates while the current feedback packet is processed.

Once the controller gains access to the tables, it waits for four cycles before latching the result of the feedback policy decision. Depending on the move/promote policy, calculating the decision for the current feedback packet may take either one or 4 cycles. Static policies such as always-move, always-promote and static load balancing require a single cycle. The intensity-based policy requires a lookup in three SRAM tables followed by two concurrent multiplications and a comparison, the result of which is latched into the policy register.

In the intensity-based move/promote policy, a hash bucket group is promoted if the packet rate of the hottest bucket is greater than N times the average packet rate for all hash buckets assigned to a sensor. The average packet rate can be calculated by dividing the total number of packets routed to a sensor by the number of hash buckets currently assigned to it. The following formula expresses the policy decision:

```
promote if packet-rate > threshold * total-packets / bucket-count
```

To avoid implementing a division circuit, the formula can be rearranged to use only multiplications, as follows:

```
promote if packet-rate * bucket-count > threshold * total-packets
```

The packet rate is provided by the hotlist, the current sensor bucket count is maintained by the per-sensor bucket count table and the total packet count is provided by the per-sensor packet count table. All three tables are indexed by the feedback sensor index within one cycle after the index is available. To improve the resolution of the threshold operand, it is provided in units of 0.25. In effect, the second operand on the right side of the formula above is multiplied by 4, or shifted left by 2 positions. Similarly, both operands of the left side in the formula are shifted by 2 bits, which is equal to a left-shift of the result of four positions. For argument sizes of 24 bits for the packet counts and 12 bits for the bucket count, a unsigned multiplication requires close to 14 ns. Given the significant clock-to-output delay of the SRAM tables, the multiplication cannot be fed directly from the table. Instead, the operands as well as the result are latched in registers associated with the multiplier core. The latches results of both multiplications are then passed through a 40-bit

comparator and are latched into the policy register. The following figure shows the data path for the intensity-based policy, both for promotion and demotion decisions.

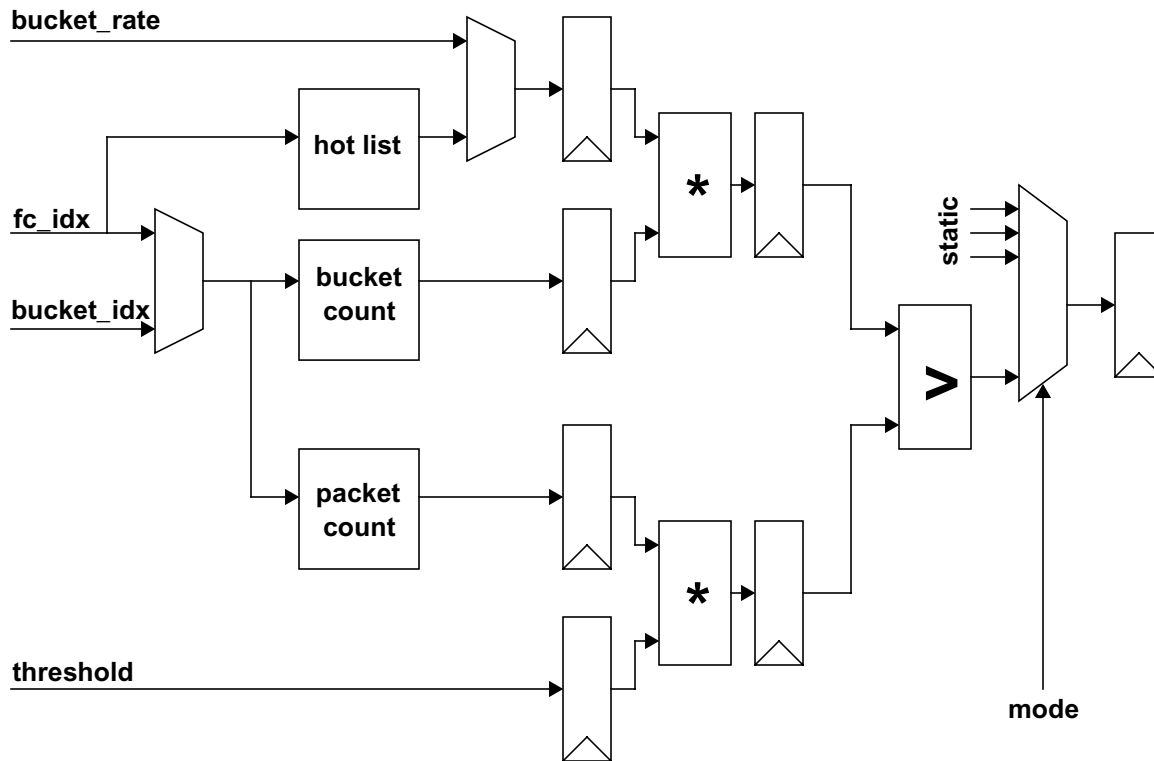


Figure 16: Intensity-based Promotion/Demotion Policy

Note that the same datapath is used for promotion and demotion. When processing a feedback packet, the table index is provided by the flow control buffer FIFO. On the other hand, when deciding whether to demote a bucket during a periodic table scan, the sensor index is provided by the hash table bucket currently under consideration. Similarly, the packet rate is not provided by the hotlist but by the hash table entry itself. Given that there are four hash tables, the packet rate operand and the table index is driven by one of five sources.

Access to this shared resources is arbitrated among the table clear state machines and the feedback processor in a static priority scheme, where the level-0 hash table has highest priority. Each subsequent entity observes the request signals from higher-priority entities and stalls until the resource is available. Note that although the feedback processor has lowest priority, it is stalled during a table clear anyway since the hash tables and hotlist contain inconsistent data during this time.

After calculating and latching the decision whether to promote the set of hot buckets, the feedback controller performs a read operation of the hottest table entry followed by a write. The hash table address as well as the hash level are provided by the hotlist, which is indexed by the flow control index while the offset within a sensors set of hot buckets is selected by the feedback controller itself. During the read-write sequence, the feedback controller indicates its access to the hash table while observing any higher-priority accesses from the routing controller. If it detects any

contention, the read-write sequence is repeated. The data written into a hash bucket includes the original packet count. If the bucket is promoted, the promote-bit is set and a random timeout value is written along with the original destination sensor. Otherwise, a new destination sensor index provided by the per-sensor packet count table is written along with a cleared promote-bit and a zero timeout value. After writing, the feedback controller proceeds to the next hotlist entry and repeats until all of the programmable number of hotlist entries have been modified. When done updating the hash buckets, the feedback controller initiates the clearing of all hotlist entries for that sensor since the buckets may no longer be assigned to that sensor.

Periodically, all hash table entries are scanned and cleared. This operation shifts all packet count values right by one bit, effectively dividing them by two and producing a weighted average packet rate. Any non-zero timeout values for promoted buckets are decremented by one as well. If the clear state machine encounters a promoted bucket with a timeout value of zero, it initiates a reevaluation of that bucket to possibly demote it. The state machine requests access to the move/promote policy datapath, thus signalling to lower-priority entities (other clear state machine and the feedback controller) that the datapath is in use, while also setting the bucket-index and bucket-rate multiplexers to accept input from its table. These multiplexers are priority encoded to match the arbitration policy. Thus, when reevaluating a bucket, the current destination sensor is used as index into the per-sensor packet and bucket count tables. Similarly, the current buckets packet rate is used as multiplier input rather than the hotlist entries. The clear state machine then waits for three cycles before latching the result. Note that the demote register is separate from the promote register since the default decisions for the static policies differ between promoting and demoting. After the three-cycle delay, the clear state machine writes the new hash table entry. If the demote-register is set, the promote-bit is cleared, otherwise a new timeout value is written. During the policy calculation and read-write cycles the state machine observes arbitration signals from any higher-priority entities. If a conflict is detected, the entire sequence needs to be restarted.

Section IV: Test Environment

files: top/test.v

The simulation environment consists of the actual loadbalancer and a small number of surrounding modules. The environment modules provide the exact interfaces available on the prototype board, but do not necessarily implement the precise functionality. Instead, most modules either provide a stimulus or display debugging information.

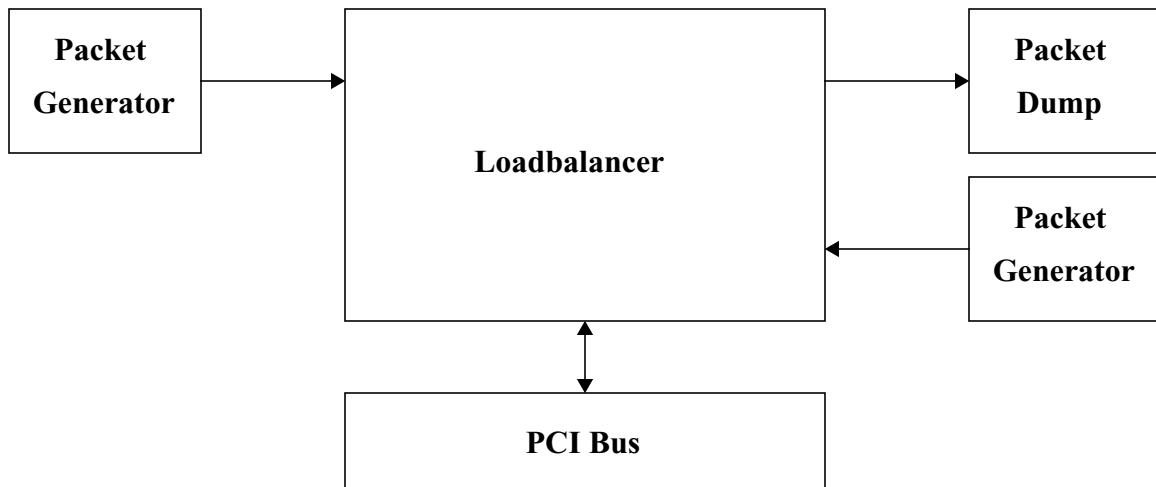


Figure 17: Test Environment

Two packet generator modules provide incoming traffic (from the network tap) and initialization and flow control traffic from the sensors. The packet-dump module displays any outgoing traffic. The PCI bus module issues any number of PCI transactions to stimulate the PCI interface.

The following sections describe the individual modules in more detail.

1 Packet Generator

files: packets/packet_generator.v

The packet generator serves the purpose of generating Ethernet packets or frames for simulation. As such, it provides an abstract model of the Metanetworks PHY daughterboard. The packet generator reads complete frames from text files and drives them on the 16-bit interface specified above.

1.1 Ethernet Packet Description

Each packet or frame is stored as a separate file with following contents:

- destination and source MAC address of 6 bytes each
- 2 bytes frame type or length
- N words (4 bytes each) IP header
- data, padding
- 4 bytes Ethernet checksum

Note that preamble and inter-packet gap are not part of the stored frame files, these are added by the packet generator. The packet content is represented as a sequence of bytes in hexadecimal notation, with white space or newline separating bytes.

1.2 Packet Generator Functionality

As a model of the Metanetworks PHY daughtercard, the packet generator provides all interface signals of the receive channel, including the clock. Asynchronous reset is the only input signal. A simple delayed inverter provides a 62.5 Mhz clock. Both clock period and offset are parameterized.

The current packet is stored in a 8-bit wide memory structure. A global variable keeps track of the current file number, whenever this number changes the memory structure is cleared to contain only 'X' and the next file is read into the array using the *\$readmemh* task.

All data and control signals are derived from the clock signal generated by the packet generator module. At every rising clock edge, a state machine performs a number of tasks. Before starting to drive any data, the control state machine inserts a configurable number of idle cycles. Idle cycles can be specified for each individual packet, and are added to the required minimum packet gap of 6 cycles (12 bytes).

After a required number of idle cycles, the packet generator drives four cycles of Ethernet preamble (0x5555, 0x555D in fourth cycle), while asserting the *valid* and *be* signal. Following the preamble, the packet generator begins reading two bytes per cycle from the memory structure and drives them on the data output. Since the memory array is initialized to contain X, two consecutive X values indicate the end of the frame. In this case, the *valid* signal is deactivated and the file number is incremented, causing a new frame file to be read into the memory. In addition, if only the second byte is X, an odd-size frame is encountered and the byte-enable signal (*be*) is deactivated.

2 TCPDump Packet Generator

files: packets/packet_generator_file.v, pli/tcpdump.c, libpcap/...

An alternate version of the packet generator reads packets from a file in TCPDump format. A combination of PLI code and Verilog code, similar to the other packet generator, read individual packets from a trace file, pad the packet to the true length and drive the data according to the Metanetworks PHY interface. Since TCPDump files usually do not contain complete packets, packets are padded to the correct length and an Ethernet checksum is added. Currently, neither IP, UDP or Ethernet checksum are recalculated after padding since the load balancer does not inspect or check any of these.

2.1 Functionality

PLI code implements a routine \$tcpdump() that takes as arguments a dump file name, a memory array and an integer. The memory array needs to be 8 bits wide and contain at least 1522 elements, thus providing sufficient space for a complete Ethernet frame. The integer argument is set to the pre-frame delay. Three aspects of the PLI interface are currently supported. The 'check' routine performs a simple argument check. The 'misc' routine is called for various events. For an end-of-compile event, the routine allocates a persistent work area, opens a tcpdump file using a libpcap function and obtains node information for the memory array argument. At the end of a simulation, the tcpdump file is closed and the work area is released. Note that the pcap-library has been modified to accept compressed (with gzip) as well as uncompressed packet trace files.

During a simulation run, the actual calls to the PLI routine are handled by a third C function. This routine first initializes the memory array with X-values and then reads the next packet from the dump file. If the frame is not an IP frame, nor a VLAN frame with an embedded IP packet, the packet is dropped and the next packet is read from the trace file. Once a valid IP packet is found, it is copied into the memory array. In most cases, the actual packet is longer than the captured data, so the packet is padded with 0x00 values. Since the padding changes the payload of the original packets, all checksums need to be recomputed. The PLI routine first decodes the IP header as well as the header of the embedded IP payload, if applicable. Currently, supported transport layer protocols are UDP, TCP, ICMP, IGMP and GRE. For each, the routine recomputes and inserts the checksum. Next, the IP checksum is calculated and inserted, and finally the Ethernet checksum is appended. Note that while the headers are correct, the payload is still not meaningful or valid. Since most traces only capture a fraction of each packet, it is impossible to completely restore the original packet. The PLI code displays a warning for any non-IP frames, and also reports when the end of the trace file is reached. The return value of the PLI function indicates the total number of packets read, or -1 if an error is encountered.

The PLI code also uses the packet time stamps embedded in the trace to report a pre-frame delay to the calling Verilog code. For each frame, the PLI code computes and saves the time in microseconds at which the frame ends. When reading the next frame, it calculates the difference between the new start time and the previous end time stamp, converts it into a number of cycles (corresponding to 125 MHz, 8 nanoseconds period), and stores it in the third argument. In addition,

the code ensures that the pre-frame delay satisfies the minimum inter-packet gap requirement (12 cycles).

The corresponding Verilog code is similar to the packet generator described previously, except that it calls the PLI routine to obtain the next packet. The latency parameter delays the first packet by the specified number of cycles. Subsequent packets are either driven with the minimum inter-packet gap, or based on the pre-frame delay computed from the trace file.

2.2 Interface

The packet generator module provides the following interface:

- clk: 62.5 Mhz clock (output)
- reset: asynchronous reset (input)
- data[15:0]: 16-bit frame data (output)
- valid: active-high frame valid signal (output)
- be: active-high byte-enable for lower 8 bits of data (output)
- err: active-high error and carrier-extension signal (output)

In addition, the following parameters specify clock period and skew as well as the names of packet dump file, with the default value specified in parenthesis.

- clk_period: period of clock provided by module (16)
- clk_skew: skew/offset of clock provided by module (0)
- filename: name of TCP Dump file (““““)
- latency: delay before the first packet is driven (0)
- true_gap: respect trace file time stamps (0)

3 Packet Dump

files: `packets/packet_dump.v`

3.1 Functionality

The packet dump module is an abstract model of an external PHY transmit port. As such it accepts a 62.5 Mhz clock signal as well as all other signals of a PHY transmit port. To aid in debugging and testing, the module displays all data transmitted and indicates a number of abnormal signalling conditions.

If the valid signal is active at a rising clock edge, the module prints the data to the simulator console. Odd frames are marked with a note. In addition, the module detects and signals invalid signal values such as 'X' or 'Z', and displays a warning if the *err* signal becomes active. In addition to displaying the raw data, the module decodes and annotates a number of packet field, including the Ethernet addresses and frame type, IP addresses, protocol type and packet length.

3.2 Interface

The packet dump module provides the following interface:

- `clk`: 62.5 Mhz clock (input)
- `reset`: asynchronous reset (input)
- `data[15:0]`: 16-bit frame data (input)
- `valid`: active-high frame valid signal (input)
- `be`: active-high byte-enable for lower 8 bits of data (input)
- `err`: active-high error and carrier-extension signal (input)

In addition, the parameter *print* can be set to 1 or 0 to enable or disable printing of frame data to the console screen. By default, printing is enabled (1).

4 Packet Monitor

files: packets/packet_monitor.v

4.1 Description

The packet monitor observes both directions of the network link between the load balancer and the sensors. Thus, it monitors data packets sent to one of the sensors as well as initialization and flow control packets. It is mainly intended to collect statistics about the load balancer behavior.

The monitor maintains a list of sensors with each entry consisting of the MAC addresses, packet count and flow control packet count. This list is built during initialization and is used to map MAC addresses to sensors during normal operation.

For data packets, the monitor only latches the destination MAC address. A broadcast packet is assumed to be the initialization request, it clears the internal sensor list and resets the sensor count. For any other packet, the monitor scans the list of sensor MAC addresses to determine the destination sensor. It then increments that sensors packet count, and optionally displays an informational message.

For feedback packets, the monitor latches the source MAC address and the opcode. If the packet is an init-reply packet, the MAC address is added to the sensor list and the sensor count is incremented. Optionally, an informational message is displayed. For flow control packets, the monitor scans the list of sensors to find a matching MAC address. It then increments that sensors flow control packet count and optionally displays a text message.

4.2 Interface

- clk_d: clock signal for data from load balancer to sensors
- data_d[15:0]: data transmitted from load balancer to sensors
- valid_d: frame-valid signal corresponding to data_d
- clk_f: clock signal for feedback data (from sensors to load balancer)
- data_f[15:0]: feedback data
- valid_f: feedback packet valid

In addition, the parameter *print* controls whether the module displays frame information in the simulator console while frames are being observed. By default, this flag is set to 1.

5 PCI Bus

files: `pci/pci_bus.v`

The PCI bus module implements a behavioral PCI master or bridge. It provides a 33 Mhz clock and generates single-cycle configuration and memory read/write transactions to stimulate the PCI target module.

5.1 Structure

The PCI bus module consists of a number of tasks that implement basic PCI bus transactions, a checksum calculation block, and some error checking blocks. The `$pci_init_read` task takes as arguments a configuration register address and a data register that receives the value read. The task first drives `idsel` and `frame` signals to initiate a transaction. During the first cycle, the address/data bus is driven with the target register number. Following the address cycle, the task asserts `irdy` and waits for `trdy` to become active. Once `trdy` is active, indicating that the device is responding with data, the data is latched from the address/data bus into the data argument, and the transaction is terminated by deasserting `idsel` and `irdy`. Other tasks such as `$pci_init_write`, `$pci_read` and `$pci_write` are implemented similarly, with a few modifications to account for the different direction of data transfers, command encodings and control signals.

Any of the bus transaction tasks are called from within an initial-block. It is important to first configure the PCI target device by (at a minimum) reading a base address register and writing a base address into it, and then enabling the device by writing into the command/status register. Other functionality of the PCI bus model include a parity generator that computes the PCI parity for the address/data bus and command/byte-enable bus for any cycle that these buses are driven by the bus model. A warning message is displayed whenever `serr` or `derr` are active.

5.2 Interface

The PCI bus model provides essentially all relevant signals of a 32-bit PCI bus.

- `clk`: PCI clock (output)
- `reset_1`: asynchronous active-low reset signal (output)
- `ad[31:0]`: PCI address and data (bidirectional, tristate)
- `c_be_1`: PCI command and active-low byte-enable (output)
- `par`: PCI parity (bidirectional, tristate)
- `perr_1`, `serr_1`: active-low PCI error signals (input, tristate)
- `frame_1`, `irdy_1`: active-low PCI frame and initiator-ready signals (output, tristate)
- `trdy_1`: active-low PCI target-ready (bidirectional, tristate)
- `stop_1`: active-low PCI stop signal (input, tristate)
- `devsel_1`: active-low PCI device-select signal (input, tristate)
- `idsel`: PCI device select during configuration (output)