Report Number: WUCSE-2002-20

2002-07-17

Statistic Counter for Networking Hardware Modules

Michael Attig and John W. Lockwood

Hundreds of types of events can occur in a complex networking device, and millions of these events occur every second. In order to debug and manage a complex networking device, it is necessary to count events and track statistics. A device has been implemented that counts 256 events from three independent sources. The module provides a simple interface to read back the number of occurrences of each event. To most efficiently utilize the area on the chip, the counting module stores the counter values in block memory on a FPGA device rather than in flip-flops.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2002-20

Statistic Counter for Networking Hardware Modules

Authors: Attig, Michael; Lockwood, John W.

July 17, 2002

Abstract: Hundreds of types of events can occur in a complex networking device, and millions of these events occur every second. In order to debug and manage a complex networking device, it is necessary to count events and track statistics. A device has been implemented that counts 256 events from three independent sources. The module provides a simple interface to read back the number of occurrences of each event. To most efficiently utilize the area on the chip , the counting module stores the counter values in block memory on a FPGA device rather than in flip-flops.

Statistic Counter for Networking Hardware Modules

Michael Attig
John W. Lockwood

Department of Computer Science
Applied Research Lab
Washington University
1 Brookings Drive, Box 1045
Saint Louis, MO 63130

WUCS-2002-20
http://www.arl.wustl.edu/arl/projects/fpx

July 17, 2002

Abstract

Hundreds of types of events can occur in a complex networking device, and millions of these events occur every second. In order to debug and manage a complex networking device, it is necessary to count events and track statistics. A device has been implemented that counts 256 events from three independent sources. The module provides a simple interface to read back the number of occurrences of each event. To most efficiently utilize the area on the chip , the counting module stores the counter values in block memory on a FPGA device rather than in flip-flops.

1 Introduction

Tracking events is a key method for debugging network hardware. The circuit described in this report provides a simple increment pulse and counter number interface to track statistics in field programmable gate array (FPGA) hardware. Originally designed to be placed within the Control Cell Processor (CCP)[1] of the FPX [2] [3] [4], the statistics counter can also be used to interface with a wide range or other network hardware modules.

The statistics counter utilizes fully synchronous design. All transfers of data and state transitions occur on the rising edge of the clock. In addition, the design is isolated from external modules by flip-flops on all inputs and outputs (excluding clock). The statistics counter has run through simulation with ModelSim, and it has been synthesized with the Xilinx back-end tools. The statistics counter can operate at a clock frequency of 76.25 MHz, encompassing only 1% of the Xilinx 2000E-6-FG680. The statistics counter has been integrated into the CCP, allowing control cells to initiate up to four counter reads.

2 Interface

The statistics counter is intended to be instantiated by an exterior module that will generate the increment/read signals and supply the counter number. A strobed increment/read pulse and counter number inform the statistics counter to increment/read the specified counter number. As shown in Figure 1, there are ten inputs and three outputs to the statistics counter.

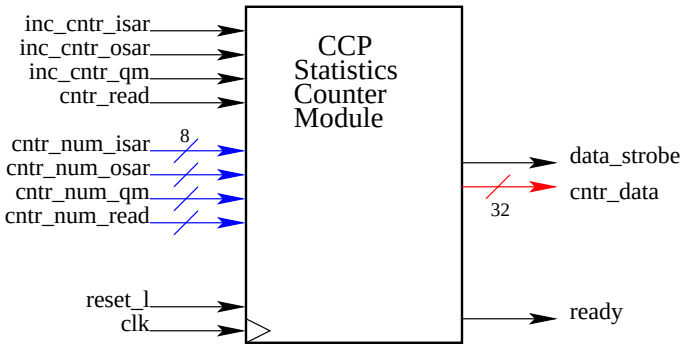

Figure 1: The Statistics Counting Module Interface. Inputs are on the left, and outputs are on the right. The supplied counter numbers are 8 bits wide, while the counter value that is returned via *cntr_data* is 32 bits.

To be used with the CCP, the statistics counter required three separate increment signals, one to go with each type of event that can occur in the Multi Service Router (MSR). The MSR generates count events for Input Segmentation and Reassembly (ISAR), Output Segmentation and Reassembly (OSAR), and Queue Manager (QM) [1].

3 Signal Specifications

3.1 Input Signals

When the input signals *inc_cntr_isar*, *inc_cntr_osar*, *inc_cntr_qm*, and *cntr_read* pulse, the corresponding 8-bit counter number must have a valid counter number. Note that if a counter number is in use by ISAR, then it is expected that OSAR and QM will not utilize the same counter number. The same holds for OSAR and QM counter numbers.

An individual increment or read signal cannot pulse on every clock edge. In order to be able to service four distinct events, an individual signal can only pulse once every four clock ticks. This enables the statistics counter to be able to initiate service for each event. The four pulse signals *inc_cntr_isar*, *inc_cntr_osar*, *inc_cntr_qm*, and *cntr_read* can pulse at any time relative to each other so long as the same signal pulse is separated by four clock cycles.

3.2 Output Signals

The output *ready* is used to specify when the statistics counter is configured to begin counting events. The statistics counter will not be configured until *reset_l* has been de-asserted for 256 clock ticks. During these 256 ticks, the statistics counter resets all the counter values to 0.

The other outputs provide the read interface to exterior modules. Depending on when the *cntr_read* input was pulsed, the counter value will be returned within three to six clock ticks following the pulse. Valid counter data is signaled by the assertion of *data_strobe*. The exterior module needs to latch the value on *cntr_data* when *data_strobe* is asserted.

3.3 Example

The following example, shown in Figure 2, shows one way that increment and read pulses could occur. Note that when an increment or read is requested, the counter number is supplied on the corresponding 8-bit

cntr_num line. For example, if *inc_cntr_isar* was pulsed, the corresponding ISAR counter number would arrive on the input *cntr_num_isar*.

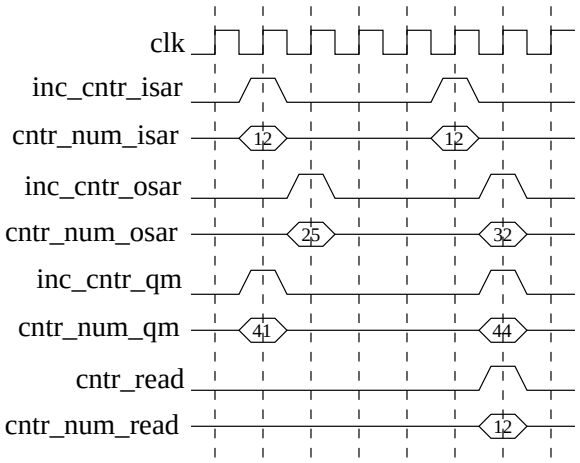

Figure 2: An example of how a increment is initiated. In the second clock tick, *inc_cntr_isar* pulses, meaning that one of the ISAR counters, in this case 0x12, needs to be incremented. *Inc_cntr_qm* also pulses in clock tick two, and counter number 0x41 is to be incremented. In clock tick three, *inc_cntr_osar* pulses, indicating that OSAR counter number 0x25 is to incremented. In clock tick six, *inc_cntr_isar* pulses again. Note that this is the earliest that *inc_cntr_isar* could have pulsed again. In clock tick seven, *inc_cntr_osar*, *inc_cntr_qm*, and *cntr_read* pulse, and each gives their respective counter numbers.

4 Architecture

The statistics counter is implemented using three files: *cntr_ram.vhd*, *cntr_fsm.vhd*, and *ccp_cntr.vhd*. *Cntr_ram.vhd* describes the usage of the dualport block RAMs located on the FPGA. The *cntr_fsm.vhd* file describes the finite state machine (FSM) used to service increment and read requests. *Ccp_cntr.vhd* wires up the block RAMs and FSM, and it describes multiplexors, flip-flops, counters, and increment logic. Refer to Figure 3 for a top-level block diagram of the statistics counter.

4.1 Dualport Block RAM

Two dualport block RAMs are used by the statistics counter. One contains the upper 16 bits of the 32-bit counter value, and the other contains the lower 16 bits. Each RAM is 16 by 256. Port A is used as the read

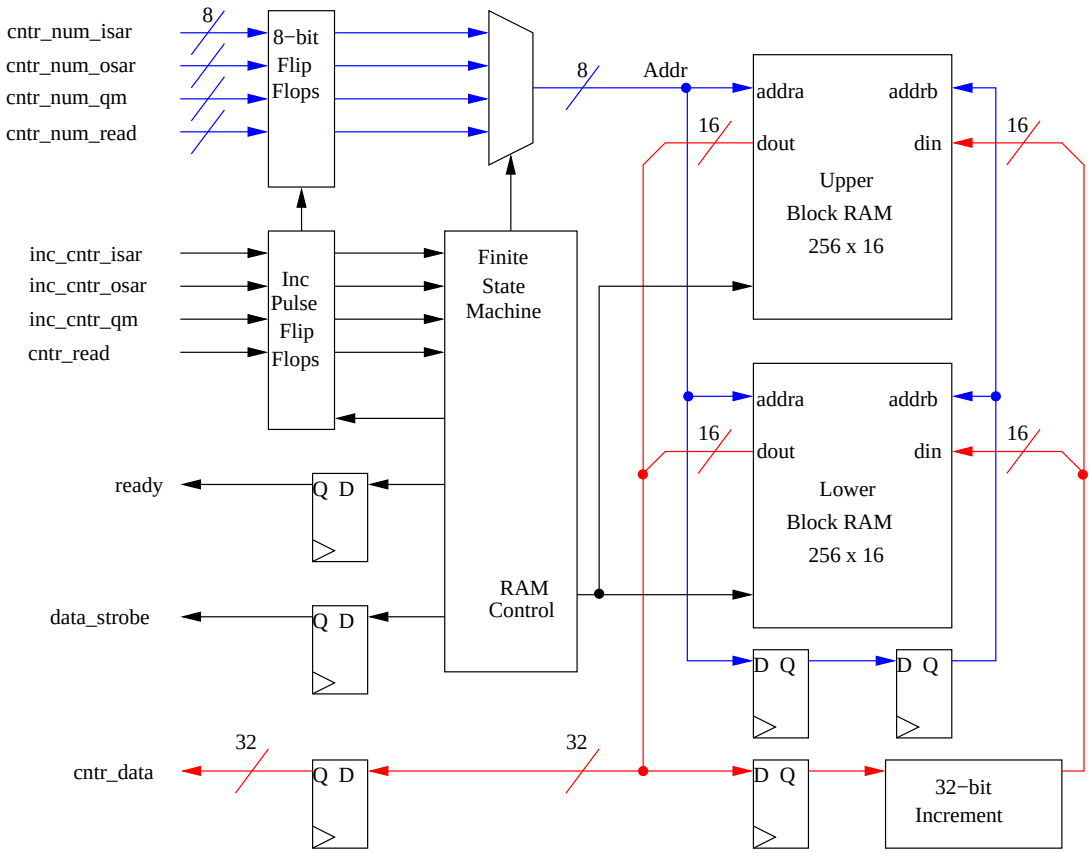

Figure 3: A top-level block diagram showing the statistics counter. Note that this figure only shows the high-level address, data, and control path signals and components.

port and port B is used as the write port. *Cntr_ram.vhd* was generated using Xilinx CORE Generator.

4.2 Finite State Machine

Cntr_fsm.vhd consists of three processes and output assignments. Only five states are necessary to implement the statistics counter: *RST_MEM*, *ISAR*, *OSAR*, *QM*, and *CNTR_READ*. In *RST_MEM*, the FSM asserts the signals necessary to reset all 256 32-bit counter values. The other four states are continuously cycled through, and the flopped outputs of *inc_cntr_isar*, *inc_cntr_osar*, *inc_cntr_qm*, and *cntr_read* are checked for requests. If a request occurs, the FSM passes the appropriate address and enables the read from RAM. Refer to Figure 4 for a bubble diagram of state transitions.

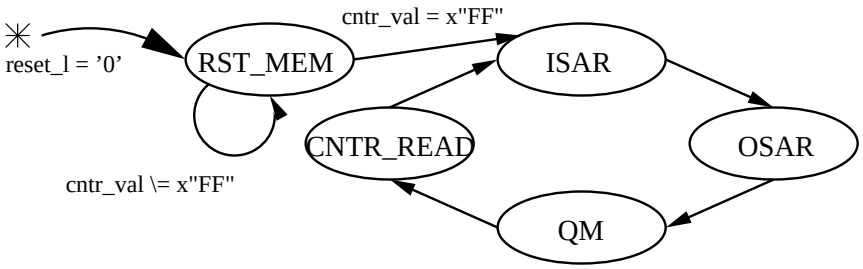

Figure 4: The FSM state transitions. Note that once the FSM leaves the *RST_MEM* state, there are no conditions on transitions from state to state. The FSM loops through each of the four states, checking to see whether an increment or read request has been issued. If so, it initiates service.

The first process, sensitive to the rising edge of clock, latches the next state. The second process describes the combinational logic to generate the next state. It is sensitive to state and *cntr_val*, where *cntr_val* is the value of an 8-bit counter used as the address when resetting memory. The third process describes the combinational logic to generate the FSM's outputs. This process is sensitive to state, *inc_cntr_isar_sav*, *inc_cntr_osar_sav*, *inc_cntr_qm_sav*, *cntr_read_sav*, and *ena_delay_2*. *Ena_delay_2* informs the FSM if it needs to issue a write enable to the write port of the dualport block RAM.

4.3 *CCP_cntr.vhd* Structural Description

To use the statistics counter, a hardware module should instantiate a *ccp_cntr* component. *Ccp_cntr.vhd* instantiates the two dualport block RAMs and *cntr_fsm.vhd*. It also describes two 32-bit flip-flops, six 8-bit flip-flops, eleven single-bit flip-flops, two 32-bit 2-to-1 multiplexors, one 8-bit 4-to-1 multiplexor, one 8-bit 2-to-1 multiplexor, one 8-bit up-counter, and combinational logic to perform a 32-bit increment.

4.3.1 Address Path

The address supplied to the two dualport block RAMs can originate from *cntr_num_isar*, *cntr_num_osar*, *cntr_num_qm*, or *cntr_num_read*. Each of these 8-bit addresses passes through a flip-flop, which is loaded when the corresponding pulse is strobed. A 4-to-1 multiplexor selects which address to pass through based on the current state of the FSM. For example, if the current state is *OSAR*, then the address *cntr_num_osar_sav* is passed through. The multiplexor output address is tied to the inputs of the read ports on both block RAMs. Two 8-bit flip-flops delay the presented read address by two clock ticks, which allows for the address to be

used for a write if an update is necessary. Refer to Figure 5 for a detailed view of the address path.

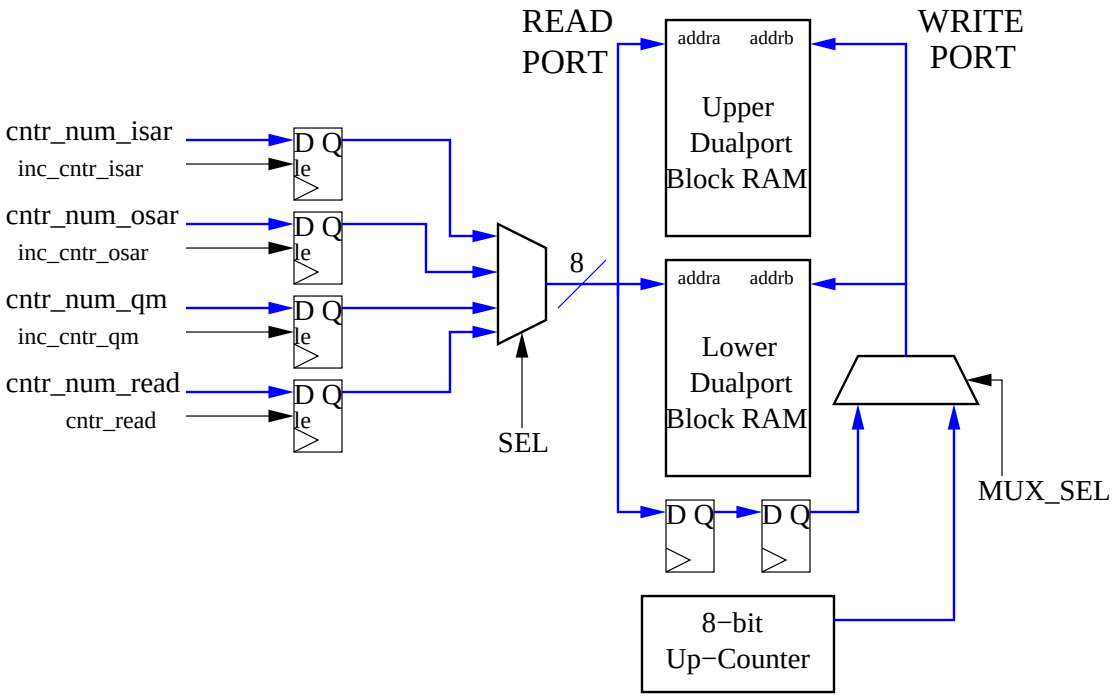

Figure 5: The address, supplied as an input, is passed through an 8-bit flip-flop and a 4-to-1 multiplexor before it is presented at the read port of the dualport block RAMs. The address is then delayed by two clock ticks before it is presented to the write port, after passing through a 2-to-1 multiplexor.

A 2-to-1 multiplexor on the write port selects whether to take the twice-delayed read address or an 8-bit counter value. The counter value is selected only during the *RST_MEM* state in which all addresses in RAM need to be reset to zeroes. In all other states, the delayed read address is selected.

4.3.2 Data Path

The data path loops around the two dualport block RAMs, passing through multiplexors, flip-flops, and combinational logic. One clock tick after the address has been presented to the read port, the output data is valid. The two 16-bit values are combined to form a 32-bit bus. This bus passes through a 32-bit multiplexor that selects between the data from the read port and the output of a flip-flop that separates the read and write ports. The output of the flip-flop is selected only in the case when the read port address matches the write

port address and a simultaneous read and write are attempted. The 32-bit flip-flop and 2-to-1 multiplexor are necessary because the read port output is undefined when a write to the same address is attempted within the same clock tick. Refer to Figure 6 for a detailed view of the data path.

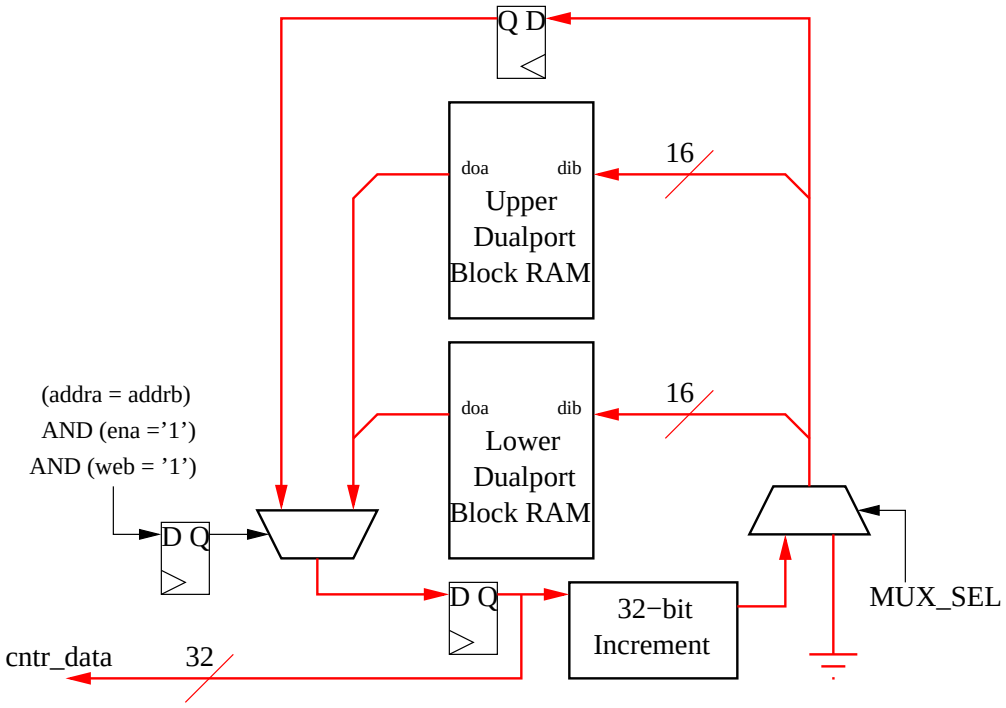

Figure 6: Data is outputted from the read port of the two dualport block RAMs and passed through a multiplexor to be used as the input for a 32-bit flip-flop. The output of this flop leaves the entity as *cntr_data* and is an input for a 32-bit increment circuit. Once incremented, the data passes through another multiplexor, which is used as an input for a flip-flop or as an input for the two block RAMs.

Once the appropriate value has been selected, a 32-bit flip-flop is loaded. The output of this flop, *cntr_data*, is used as an input for a combinational logic circuit to perform the 32-bit increment. The output of the increment circuit is routed into another 2-to-1 multiplexor. This multiplexor selects either the incremented data or ground. Ground is selected only in the case when the FSM is in the *RST_MEM* state. The output of this multiplexor is used as an input for the flip-flop which lies between the read and write ports and it is bus-ripped. The upper 16-bits go to the upper block RAM, and the lower 16-bits go to the lower block RAM.

4.3.3 Control Path

The control path consists of increment and read requests, the flip-flop load enables, multiplexor selectors, RAM control signals, and a counter enable. Refer to Figure 7 for a detailed view of the control path.

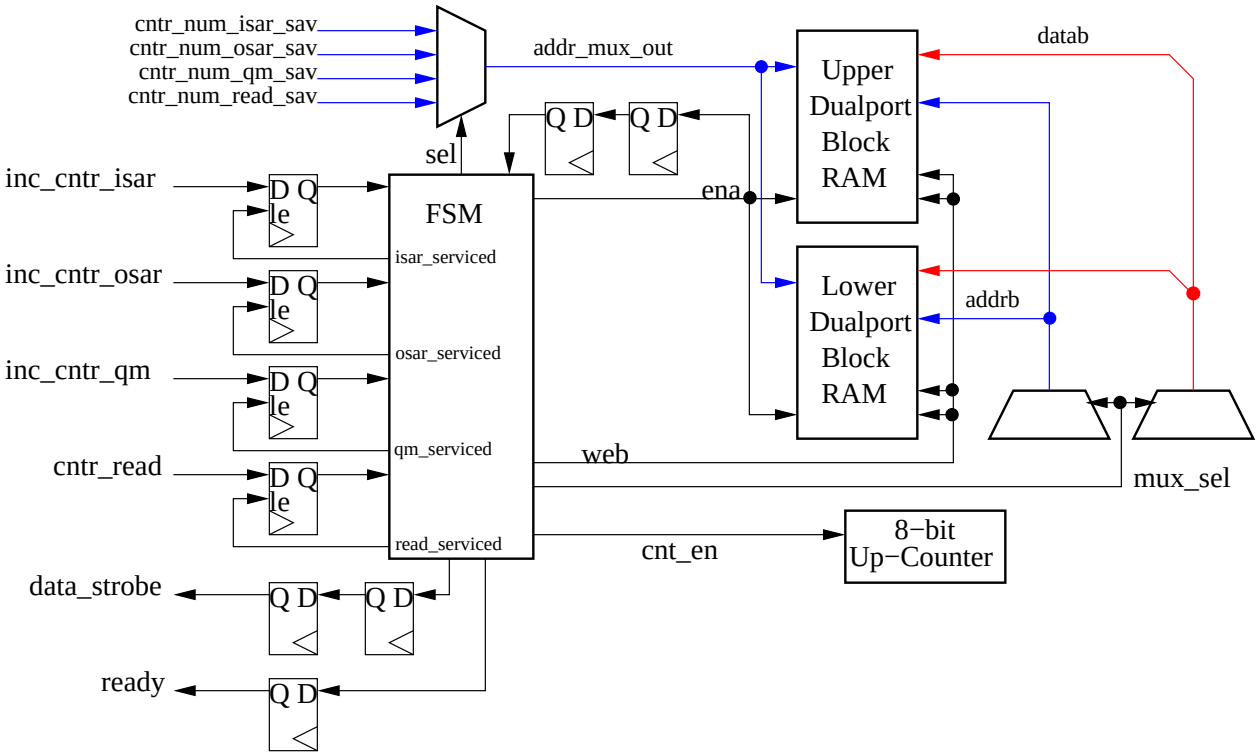

Figure 7: The control path consists of the increment and read pulse signals, flip-flop load enables, multiplexor select signals, RAM control signals, and signals to be output out of the entity informing external components when the statistics counter is ready to begin counting events and when valid data is present.

First, the signals *inc_cntr_isar_sav*, *inc_cntr_osar_sav*, *inc_cntr_qm_sav*, and *cntr_read_sav* are checked when the FSM is in the corresponding state. For example, if the current state is *QM*, then the FSM will check *inc_cntr_qm_sav* to see if a pending increment request exists. If the flip-flop output signal is asserted, the FSM asserts *ena*, which initiates a read from RAM. Since the increment or read request has been serviced, the corresponding *STATE_serviced* signal is asserted. The signals *isar_serviced*, *osar_serviced*, *qm_serviced*, and *read_serviced* are the load enable of their respective pulse flip-flops. The pulse load enables are asserted when the flop output is '0' or when the request has just been serviced.

Second, there are two multiplexor selects: *SEL* and *mux_sel*. *SEL* selects which of the four address to present to the RAMs. For example, if the FSM is in the *CNTR_READ* state, it will select the address *cntr_num_read_sav* to present to the RAMs. *Mux_sel* selects between address and data associated with an incremented value or the address and data for the case when memory is being reset.

Third, RAM control consists of two FSM output signals *ena* (enable read port) and *web* (enable write port). *Ena* is asserted when the flip-flop output signal is asserted in the corresponding state. For example, if *inc_cntr_isar_sav* is asserted and the current state is *ISAR*, then *ena* will be asserted. *Ena* is passed through two delay flip-flops and then returned to the FSM. If *ena_delay_2* is asserted in states *ISAR*, *QM*, or *CNTR_READ*, then *web* is asserted. After delaying the *ena* from *CNTR_READ* by two clock ticks, the FSM will be in the *OSAR* state. Because reading a counter should not update the value, the *OSAR* state is not included in the list above.

Finally, three miscellaneous signals *ready*, *data_strobe*, and *cnt_en* are part of the control logic. *Ready* signals that the statistics counter has been configured to begin accepting increment and read requests. *Ready* is asserted when the state of the FSM is not *RST_MEM*. A *data_strobe* is issued in the *CNTR_READ* state. This is delayed by two clock ticks before it is outputted from the entity because one clock tick is needed to retrieve data from RAM, and one clock tick is needed to pass the data through a flip-flop. Finally, *cnt_en* is a signal used to control the 8-bit address counter used when memory is being reset.

5 Operation and Timing

The following timing diagrams show some of the fundamental aspects of the statistics counter. The fastest that individual increment signals can be pulsed, the general read interface, and the memory latency are discussed.

5.1 Normal Operation

In Figure 8, an ISAR event occurs, triggering a *inc_cntr_isar* pulse. Concurrently, an OSAR , a QM, and a CNTR_READ event occur. The second set of pulses occur exactly four clock ticks after the preceding set, which is the earliest that the second set could have pulsed. The third set of pulses occur 4 ticks later for ISAR and QM, and 5 ticks later for OSAR and CNTR_READ.

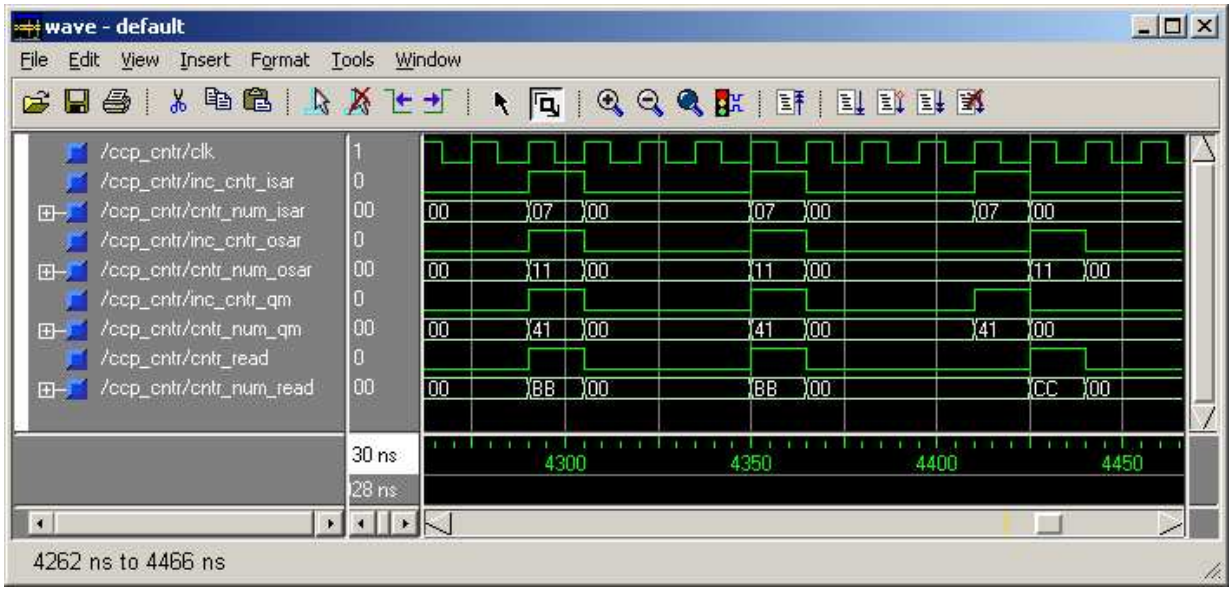

Figure 8: The first set of pulses happen to occur in the same clock tick. The statistics counter will properly initiate service for all of these. Note that the second set of pulses occur four clock ticks after the preceding group.

5.2 Counter Reads

To initiate a counter read, the exterior module must pulse the *cntr_read* signal and supply the counter number to be read during the same pulse time. In Figure 9, a read is conducted on counter numbers 0x07, 0x11, and 0x41. Note that the spacing between the *cntr_read* pulses is four, which is the smallest the spacing can be.

Three clock cycles later, the corresponding counter value is returned via *cntr_data* with the corresponding *data_strobe*. Note that the returned counter value is only valid for one cycle during the strobe. Also note that the returned counter values are from the incremented counters in Figure 8, which explains why three is returned.

5.3 Memory Latency

Due to the cyclic nature of the FSM, the latency of the response once a counter read is requested varies between three and six clock ticks after the *cntr_read* pulse. Figure 10 shows the four cases, starting with the three tick wait and ending with the six tick wait.

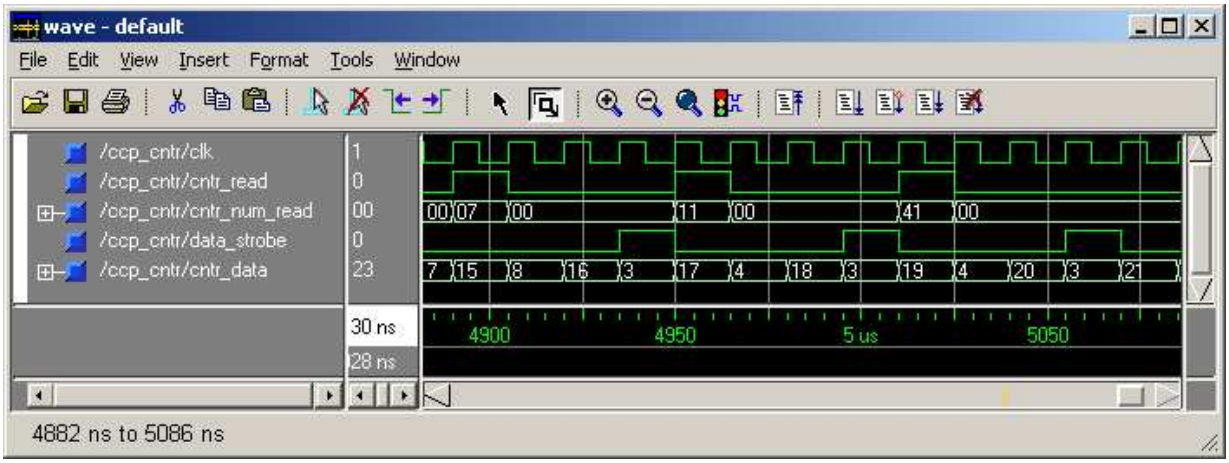

Figure 9: Once a read has been requested, the data is returned three to six clock ticks later accompanied by a *data_strobe* signifying that the data is valid.

As Figure 10 shows, a three tick latency occurs if the read request occurs in the *QM* state. A four tick, five tick or six tick latency occurs if the request occurs in the *OSAR*, *ISAR*, or *CNTR_READ* state, respectively. Three clock ticks is the minimum latency because one clock is necessary for the request to pass through the first flip-flop, one clock is needed to retrieve the counter value from the dualport block RAMs, and one clock is necessary for the counter value to exit the entity through another flip-flop.

5.4 Separation to Ensure Updated Read

In the case of concurrent, or near-concurrent increment and read requests of the same address, the returned counter value does not always reflect the most recent increment request. Due to the state of the FSM when the requests occur, the counter value in RAM may not have been written yet. Refer to Figure 11 for an example where a QM increment request has been issued and a read is simultaneously attempted of the same counter. The counter does not reflect the most recent increment request.

If the most up-to-date event counter value is not necessary for the application, as it was not in the case of the CCP, then a read request can be issued at any time. However, if the most recent request needs to be included, the read must be delayed 4 ticks, 4 ticks, or 6 ticks if the increment request was ISAR, OSAR, or QM, respectively.

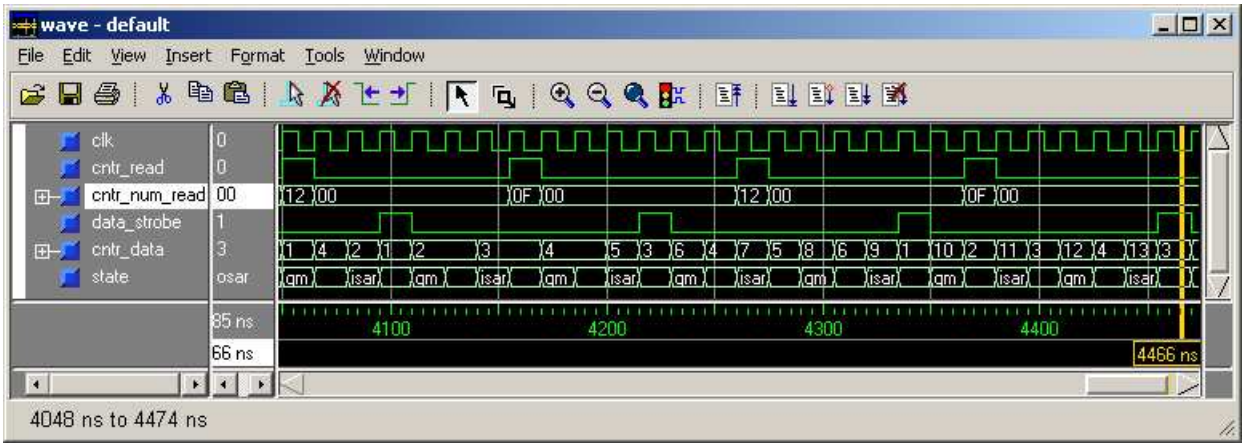

Figure 10: A simulation showing the four possible latencies from *cntr_read* pulse to the corresponding *data_strobe*. Due to the cyclic nature of the FSM, the latency can be from 3 to 6 clock ticks.

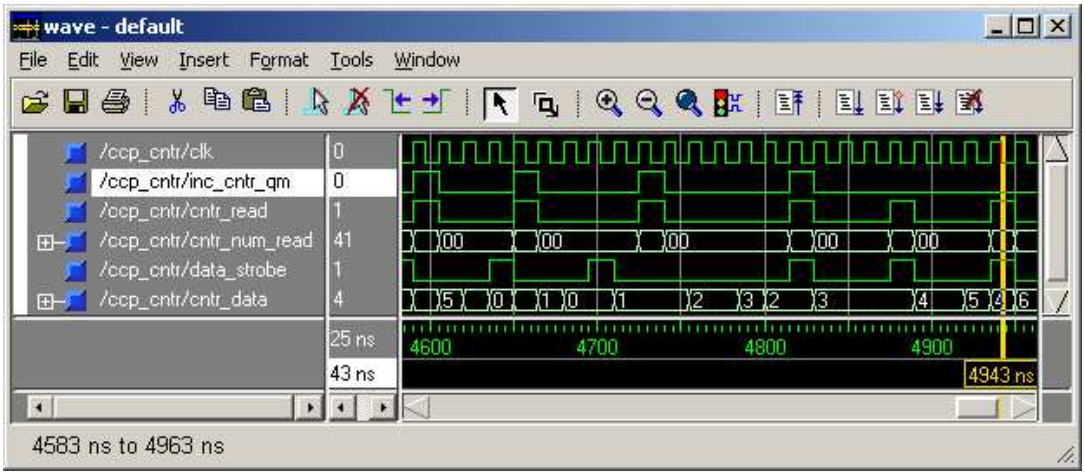

Figure 11: When an increment and read request occur concurrently on the same counter value, the value returned is not always the most recent. This waveform shows a QM increment and read from the same address. The first and second *data_strobe* do not reflect the previous two increments. The third and fourth are one behind, and the fifth read request shows the correct value.

6 CCP Modifications

6.1 Interface

In order to place the statistics counter within the CCP, a statistics counter interface was created. This interface communicates with the main FSM of the CCP. The new states *start_stat_op* and *stat_op* were added. During a statistics operation, payloads 2, 4, 6, or 8 are loaded with the counter data requested. The statistics read interface can be seen in Figure 12.

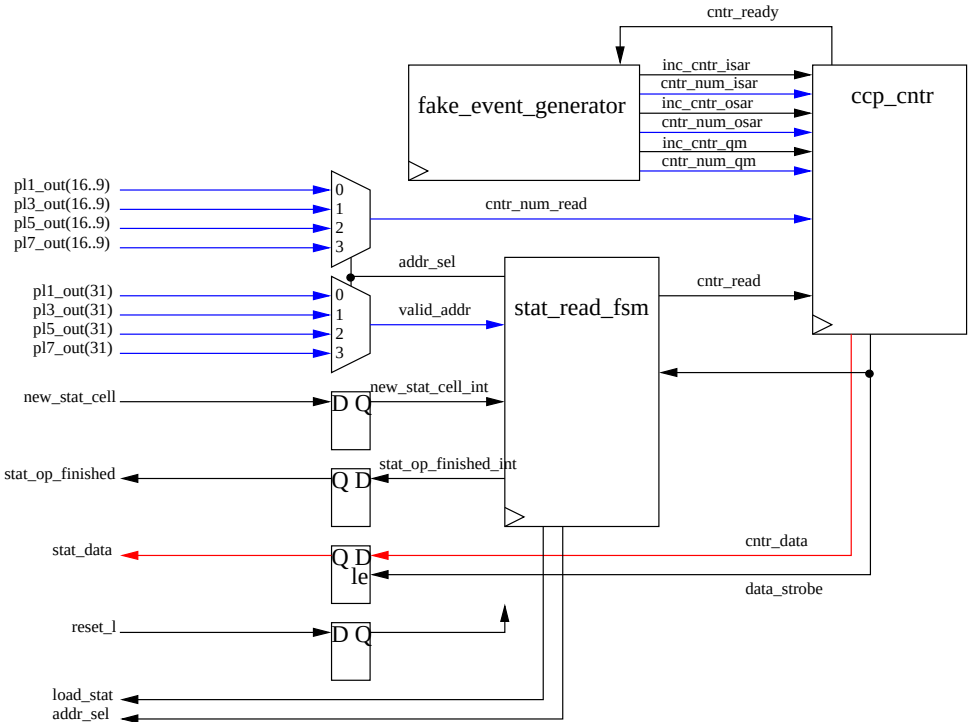

Figure 12: The statistics interface instantiates a ccp_cntr, serves as the read interface, and currently generates fake events for testing purposes.

The statistics interface is composed of a *ccp_cntr*, a *stat_read_fsm*, and a *fake_event_generator*. The *fake_event_generator* will be discarded when real events are being recorded, and the increment signals and counter numbers will enter this entity. Payloads 1, 3, 5, and 7 contain the counter address to read. The *stat_read_fsm* chooses which payload to query. If an address is to be read, the *stat_read_fsm* issues a *cntr_read* pulse and waits for the corresponding data. Once data has been retrieved, the FSM asserts *load_stat*, which

loads the appropriate payload field.

6.2 Cell Format

In one control cell, four counters can be read. The control cell format used is shown in Figure 13. To initiate a counter read, a control cell of VCI 0x23 is passed into the RAD module. Using OpCode 0x18 to specify a counter read, the user places the desired counter address in bits 16 down to 9 and sets bit 31 of the corresponding payload. Up to four counters can be read by one control cell. These address are placed in payloads 1, 3, 5, and 7, and the corresponding data is returned in 2, 4, 6, and 8, respectively.

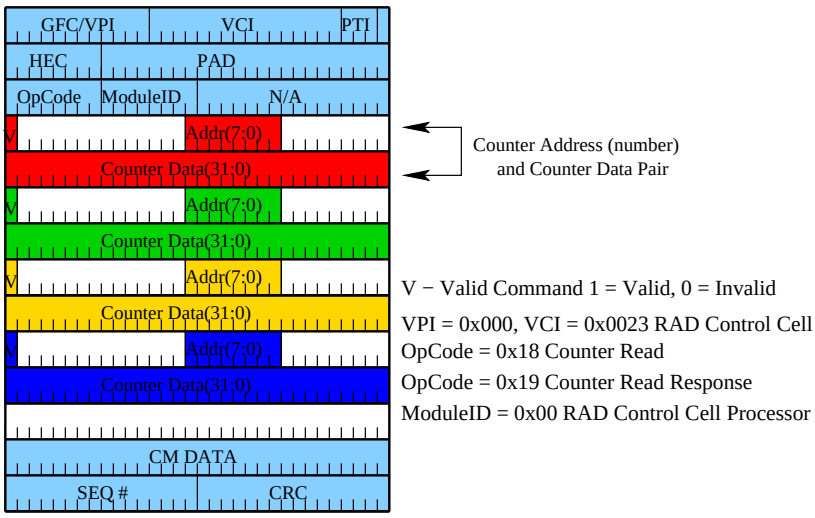

Figure 13: The cell format used to conduct statistics reads is similar to memory control cell formats [5]. A HDR, HEC, and OpCode field make up the first three words. Counter addresses are placed in bit 16 down to 9 of payloads 1, 3, 5, and 7, and counter data is returned in 2, 4, 6, and 8. A valid address is communicated by setting bit 31 of payload 1, 3, 5, or 7.

6.3 Testing

The statistics counter was tested using the modified CCP. Control cells of the format shown in Figure 13 were passed through the CCP. In the following example, payloads 1, 3, and 5 have valid addresses. Counters 0x0F, 0x15, and 0x40 are to be read, and the results of the reads will be placed in payloads 2, 4, and 6, respectively.

The ISAR and OSAR counters are triggered to stop when *inc_cntr_isar* has pulsed 0xFFFF times. Thus, the value returned in payload two is 0xFFFF. When *inc_cntr_isar* has pulsed 0xFFFF times, *inc_cntr_osar* has only pulsed 0xFFFE times. Thus, 0xFFFE is returned in payload 4. The results of the test cell can be seen in Figure 14.

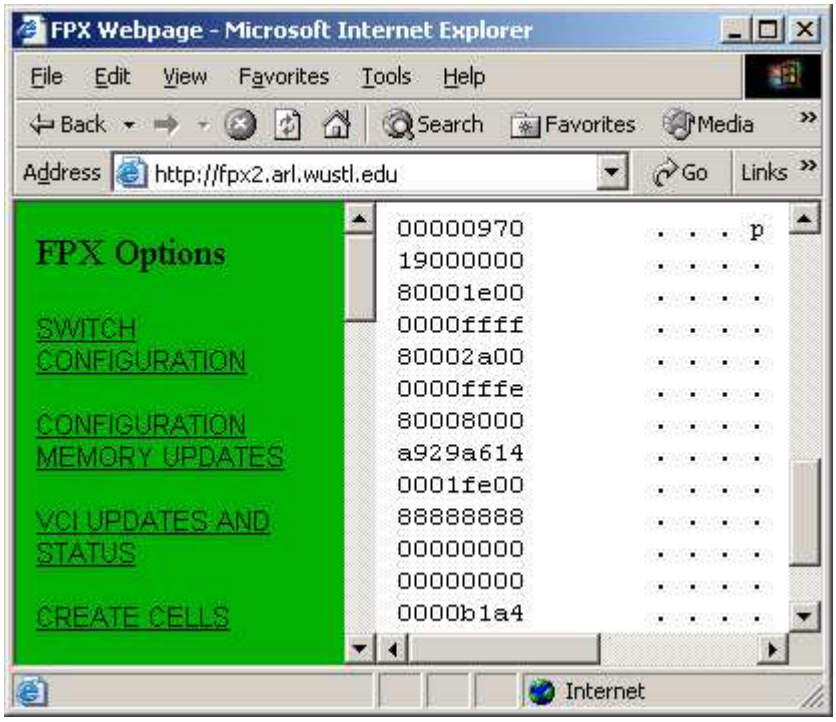

Figure 14: Counters 0x0F, 0x15, and 0x40 return 0x0000FFFF, 0x0000FFFE, and 0xA929A614, respectively. Since the valid bit was not set in payload 7, no counter data is returned in payload 8. The data present there when the cell arrived returns unchanged when the cell leaves.

7 Access

The statistic counter can be found in the CVS tree under /project/arl/fpx/cvsroot/FPX_ROOT/RAD/INFRASTRUCTURE/CCP_CNTR/. A CCP instantiating the statistic counter with interfaces for SRAM and SDRAM can be found in the CVS tree under /project/arl/fpx/cvsroot/FPX_ROOT/RAD/TOP/RAD_CCP-_STAT_CNTR/. Also, a CCP with an instantiated statistic counter and SRAM interface can be found in the

CVS tree under /project/arl/fpx/ cvsroot/FPX_ROOT/RAD/TOP/RAD_CCP_STAT_CNTR_NO_SDRAM/.

8 Conclusion

The statistics counter was used to track three different groups of MSR events: ISAR, OSAR, and QM. Each of these groups has many types of events, specified by a number. By asserting an increment request pulse and supplying the event number, the statistics counter will record the event, storing it in 1 of 256 locations in RAM. The statistics counter is a versatile entity that could easily be modified to track other groups of events. Also, the statistics counter is small and could be used with a clock frequency of up to 76.25 MHz. It only requires 1% of the Xilinx2000E-6-FG680.

9 Acknowledgements

Thanks are due to David Taylor, who specified the interface between the MSR modules and the statistics counter.

10 Future Work

The statistics counter described in this paper could be enhanced to support a significantly larger number of input events and counters by storing portions of the counters in off-chip SRAM or SDRAM. In the architecture of the Largest Counter First Counter Management Algorithm (LCF-CMA) [6], small portions of counters are stored on-chip, while the remaining portion of the counter is stored off-chip. Next, specific counter sizes are allocated and time slots to access memory are scheduled to ensure that an event is always tracked. Such an architecture would allow up to a million events to be tracked on network links operating at tens to hundreds of Gigabits per second.

References

[1] D. E. Taylor, S. Dharmapurikar, J. W. Lockwood, J. S. Turner, Y. Chen, A. Chandra, and W.-J. Tang, "Field-programmable Port eXtender (FPX) Support for the Multi-Service Router (MSR) Version 1.0," May 2002.

[2] "Field Programmable Port Extender Homepage." Online `http://www.arl.wustl.edu/arl-/projects/fpx/`, Aug. 2000.

[3] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, (Monterey, CA, USA), pp. 137–144, Feb. 2000.

[4] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, (Monterey, CA, USA), pp. 87–93, Feb. 2001.

[5] D. E. Taylor, J. W. Lockwood, and N. Naufel, "RAD Module Infrastructure of the Field-programmable Port eXtender (FPX)," tech. rep., WUCS-01-16, Washington University, Department of Computer Science, July 2001.

[6] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a Statistics Counter Architecture," in *Proceedings of Symposium on High Performance Interconnects (HotI'01)*, (Stanford, CA, USA), pp. 107–111, Aug. 2001.