NetFPGA Hardware Modules for Input, Output and EWMA Bit-Rate Computation

[1]Alfio Lombardo, [2]Diego Reforgiato, [2]Vincenzo Riccobene, [1]Giovanni Schembra
[1]*Dipartimento di Ingegneria Elettrica, Elettronica e Informatica*
[1]University of Catania
[2]*LightComm s.r.l.*
Email: [1]{alfio.lombardo, schembra}@dieei.unict.it,
[2]{diego.reforgiato, econet}@lightcomm.it

March 21, 2012

Abstract

NetFPGA is a hardware board that it is becoming increasingly popular in various research areas. It is a hardware customizable router and it can be used to study, implement and test new protocols and techniques directly in hardware. It allows researchers to experience a more real experiment environment. In this paper we present a work about the design and development of four new modules built on top of the NetFPGA Reference Router design. In particular, they compute the input and output bit rate run time and provide an estimation of the input bit rate based on an EWMA filter. Moreover we extended the rate limiter module which is embedded within the output queues in order to test our improved Reference Router. Along the paper we explain in detail each module as far as the architecture and the implementation are concerned. Furthermore, we created a testing environment which show the effectiveness and efficiency of our modules.

1 Introduction

The NetFPGA [11] is an accelerated network hardware that augments the functions of a standard computer. It provides an open router with four 1 Gbps Ethernet ports largely used in the research community, to develop and test innovative networking solution on a real environments. At the center of the NetFPGA board is a Xilinx FPGA device. Surrounding the FPGA are four memory devices, two Static RAMs (SRAMs) and two second-generation Double Date Rate (DDR2)

SDRAM devices. On the left side of the platform, a quad-port physical-layer transceiver (PHY), that enables the platform to send and receive packets over four standard twisted-pair Ethernet cables, is provided. On the right side of the board, two Serial ATA (SATA) connectors on the platform allow multiple NetFPGAs within a system to exchange data at high speeds without using the PCI bus. The NetFPGA core clock works at 125 MHz, meaning that each clock cycle lasts 8 ns. The FPGA directly handles all data-path switching, routing, and processing operations of Ethernet frames and Internet packets, leaving software to handle control-path functions only [19]. The NetFPGA fits into a host PC via a PCI slot. Software and gateware (Verilog HDL source code) packages are available for download under an open source license from the NetFPGA website [1].

Working with NetFPGA platform, a developer can either implement its own project or extend existing ones in order to augment their functionalities. Therefore, this allows jump starting prototypes and quickly building on existing projects already developed in the NetFPGA (see the NetFPGA project page [2]). One of the main NetFPGA projects, the Reference Router [3], is a complete IPv4 router which is able to simultaneously forward packets from all four 1 Gbps interfaces on the NetFPGA card.

The NetFPGA board is programmable using the Verilog language. Each time a new or extended Verilog project is completed, it can be uploaded on the board using appropriate software tools released with the platform. Of course, it is required a communication between the NetFPGA board and the host computer especially if the latter has to show some current results of the board to the user. This is accomplished through the use of the Register System [4], which is a standard set of registers that can be used by hardware and software modules to read and write data and, consequently, to exchange data between them. These registers contain some parameters for general use, typically used in order to control and monitor the platform in almost all aspects. The Register System provides a mechanism for:

- specifying the registers supplied by each module;
- specifying the modules used by each project;
- generating a register map/memory allocation for each project.

Information for each project (eg. name, list of modules, location of modules in memory space) and each module (eg. name, list of registers) is specified in an XML file. The register generation tool, provided within the platform, reads the XML file of the project and the XML files of the included modules, performs memory allocation, and then outputs a set of files with the memory allocation/register map to files for use in Verilog, C, and Perl.

For example, one of these registers (the CPCI_CNET_CLK_SEL_REG register) is responsible for the NetFPGA core clock. Its setting allows to switch the NetFPGA core clock from 125 MHz to 62.5 MHz and vice-versa. Registers provide several information about the underlying project loaded into the NetFPGA board. For example, as far as the Reference Router project is concerned, registers provide various information such as the number of bytes or packets received within the input and output queues, the number of dropped packets within the input and the output queues, the number of packet waiting into the input and output queues, etc.

However, important information is still missing within the registers and in order to compute that, one has to carefully change the original design, find the required signals from the Verilog code (it may be needed some hardware computation according to what kind of information is required), and output those in new ad-hoc registers if they need to be read or write from software. For example, one could be interested in the effective input bit rate into the input queues, the effective output bit rate from the output queues, or an estimation of the input bit rate: in fact, a large set of applications may require that kind of information.

As far as the bit rate computation is concerned, it is necessary to read the number of bits received in a certain time window. The number of bytes or packets received to the Reference Router is an information provided within the NetFPGA registers and accessible from the software using C or PERL or bash script commands. However, reading hardware registers from the software takes about 500 μs whereas the board works at 8 ns per cycle. Thus, if we want to accurately compute the input bit rate from software, we would need to read the number of bytes received within the input queues each T μs (with T much higher than 500 to not incur in errors caused by time delays of software reads).

Therefore, in order to efficiently and precisely compute the input and output bit rate it is necessary to add some hardware modules to the original Reference Router design. This allows to work in the nanoseconds domain rather than microseconds. These modules handle important signals from the user data path and according to the current clock rate compute efficiently and accurately the input bit rate, the output bit rate and the exponentially weighted moving average input bit rate. Moreover, we have extended the original rate limiter module by adding a more fine-grained limit range for the output bit rate, in order to create the same conditions when a congestion occurs and test our modules accordingly.

In this paper we have built on top of the Reference Router project four Verilog modules which work in parallel with respect to the original pipeline. The four modules are:

1. the Input Bit Rate Calculator;
2. the Output Bit Rate Calculator;
3. the EWMA Bit Rate Calculator.
4. an extension of the original rate limiter module.

The first and the second of them are used to compute the current input and output bit rate, whereas the third is used to estimate the input traffic bit rate through an exponentially weighted moving average filter. The forth module is an extension of the original rate limiter module which lies within the Output Queues and it allowed us to test the other modules.

This paper is organized as follows. Section 2 discusses some related works within the NetFPGA platform that have built on top of the Reference Router. Section 3 describes the architecture we have designed as well as the technical details of each implemented module. Section 4 gives the details of the experiments we have carried out, our network topology and how we have tested our modules. Section 5 compares the device utilization of the Reference Router design with that of our modules. Finally, Section 6 ends the paper with conclusions and future directions where we are headed.

2 Related Work

Since its release, the NetFPGA has been enhanced and improved with several modules and standalone projects for different applications. In this section we will briefly list the main and most recent projects that have been created on top of the Reference Router.

In [8] the Reference Router has been augmented for real-time extraction of URLs from packets. This implementation modifies the gateware to filter packets containing a HTTP GET request and sends a copy to the host. Host software is implemented to extract URLs and search terms. The software integrates with a database facility and a GUI for offline display of web-access and search term profiles. On the same topic, authors in [15] proposed a hardware-based HTTP GET flooding detection and defense system, which can protect a given web server farm by filtering out malicious HTTP requests based on the difference of the behavior between normal browsers and bots.

The work done in [21] describes a traffic monitor system implemented on the NetFPGA Reference Router. It allows network packets to be captured and analyzed from up to all four of the Gigabit Ethernet ports. Moreover, a developed graphical user interface shows the traffic of any port. The same authors, in [20], built a system in order to hijack the incoming packets according to rules specified by the user through ad-hoc NetFPGA registers. This means that the authors were

able to change any field of any incoming packets. Certainly, depending on whether they are changing TCP or IP header fields, they need to recompute the TCP or IP checksum and store them back into the network packets. Their implementation works at user data path level and modifies packet fields if certain conditions defined by the user through NetFPGA registers are satisfied.

In many mission critical real-time networked systems, such as those used in financial institutions, incoming data (such as market feeds) is brought in on redundant links. These links are generally provided by separate providers for maximum redundancy. Although the data on both of these links is expected to be the same, there are delays and packet losses that can be different. In [13] the authors describe a NetFPGA module which can accurately measure these delays to help the institutions to evaluate the quality of service provided to them by their vendors.

Authors in [23] have proposed a light-weight queue management scheme to tackle the problem of bursty traffic. This scheme was called bounded jitter policy and has been evaluated using testbed experiments on NetFPGA. Basically, the Reference Router was changed so that each packet is stored in the SRAM immediately after arrival.

Furthermore, authors in [22] presented an implementation of a Crosspoint-Queued switch output controller on the NetFPGA where the output controller is a part of design that implements functionality of Crosspoint Queued Ethernet switch and it performs a scheduling algorithm on the crosspoint buffers. Round robin algorithm is chosen as a scheduling algorithm. Besides the basic scheduling functions, the output controller performs other functions such as de-segmentation and error detection, which are needed in order to make a device fully functional in the real network environment.

Authors in [14] presented the design and prototype of a hardware implementation of a packet pacing system based on the NetFPGA system. Results showed that traffic pacing can be implemented with few hardware resources and without reducing system throughput.

A practical and general coder and decoder of network coding has been developed in [24] within the NetFPGA board where the entire logic of the IP layer has been redesigned.

In [12] authors presented a NetFPGA Logic Analyzer with a triggering mechanism that captures the control signals and datapath of the NetFPGA at the full 125M samples per second for the allotted duration. The triggering mechanism is a programmable pattern matcher module with mask that can be modified while the system is online.

Authors in [17] presented an approach to provide a robust solution by remodeling NetFPGA reference architecture for deep packet inspection such that the packet processing delay is highly negligible. It is discussed the respective implementation of devising high speed FSMs in a

pipelined architecture that has been validated for maintaining throughput of 1 Gbps with a set of SNORT based signatures. Besides that, authors in [18] presented a compact implementation for on-line traffic change detection on a NetFPGA platform as sketch-based algorithms are widely applied in various networking applications.

Authors in [16] proposed a Layer 2 congestion control mechanism for high-speed data center networks and a prototyping this mechanism on NetFPGA.

Finally, [7] describes an implementation of a high-speed firewall on NetFPGA, in which the authors changed the output port lookup in order to read the packets content and analyze it.

3 The proposed architecture

As mentioned above, we have been working within the NetFPGA Reference Router project [3] extending its design. It consists of a set of Verilog modules working in pipeline. The main component is the User Data Path: this module takes its inputs from the the input queues and sends its output to the output queues[1]. Its primary function is to elaborate the incoming packets and decide what to route in each of the output port. In order to achieve this, the User Data Path is composed of three main sub-modules:

1. the Input Arbiter module, which takes the network packets from the input queues;
2. the Output Port Lookup module, which processes the network packets based on the information contained in the routing table;
3. the Output Queues module, which routes network packets in the correct output queue, based on the decision of the Output Port Lookup module.

As far as the pipeline is concerned, the network packets are forwarded from module to module according to predefined schedules. Each module performs some tasks on the packets and then forward them during predefined time frames. New modules may be developed and located in the original pipeline. In our work, we have extended the standard Reference Router architecture, including three new hardware modules in order to compute the input and output bit rate and the ewma bit rate. Moreover, we have improved and generalized an existing module which limits the output bit rate according to user specific values. In particular, we have first analyzed the whole Reference design [5] in order to localize the signals that we were interested for our

[1] Along this paper we will refer to the input queues as Rx queues and to the output queues as Tx queues

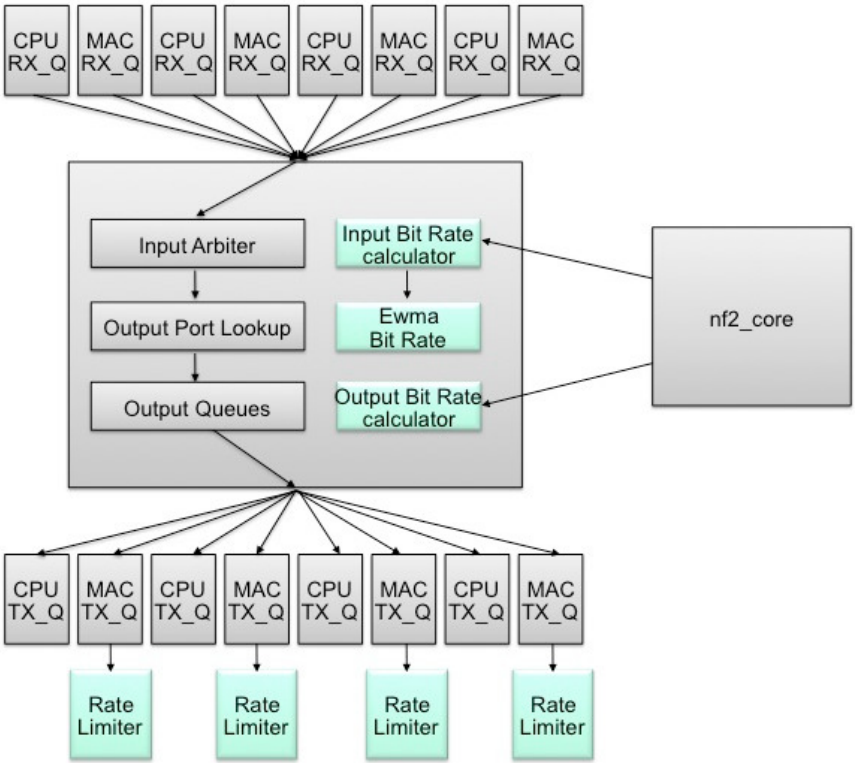

Figure 1: Architecture pipeline of the extended Reference Router.

purposes. Then, we were able to modify the original design in order to feed those signals to the User Data Path module. Then, it was easy to capture these signals from the new developed modules within the User Data Path. Figure 1 shows an architecture of the extended Reference Router. The lighter gray boxes indicate the new modules whereas the dark gray boxes refer to the original design.

Sections 3.1, 3.2, 3.3 and 3.4 describe, respectively, the Input Bit Rate Calculator module, the Output Bit Rate Calculator module, the EWMA Bit Rate Calculator module, and the extended Rate Limiter module.

3.1 Input Bit Rate Calculator

The Input Bit Rate Calculator is a hardware module that provides the input bit rate which is received within the input queues from the Ethernet ports. In particular, this module computes two things:

1. the input bit rate in each queue,
2. the overall bit rate (summing up the four input bit rates).

The reader notices that information needed to compute the input bit rate is already available within the NetFPGA registers. In particular, the number of bytes received in each input queue is accessible through the MAC_GRP_i_RX_QUEUE_NUM_BYTES_PUSHED_REG register (where i ranges in $\{0, \ldots, 3\}$ as we have four input queues). Each of the four registers contains the progressive number of bytes received in the corresponding input queue. Each register has a fixed dimension of 32 bits: when the value of the register reaches the maximum it restarts from 0. In order to compute the bit rate for the input queue i we need to read the value of the ith register each time window w and convert the number of received bytes within w in a bit rate value.

The rationale behind this is the following: when the NetFPGA works at 125 MHz, each clock cycle lasts 8 ns. If r_{i_1} and r_{i_2} are the two values read from the MAC_GRP_i_RX_QUEUE_NUM_BYTES_PUSHED_REG register, respectively, before and after w, the obtained bit rate for the queue i is equal to:

$$bitrate_i = (r_{i_2} - r_{i_1}) \cdot \frac{w}{10^9} \cdot \frac{1}{V_{freq}} \tag{1}$$

where V_{freq} is equal to 1 if the current clock frequency is 125 MHz and 2 if it is 62.5 MHz (when the NetFPGA works at 62.5 MHz the clock cycle becomes 16 ns). The user notices that the lower w, the higher the sampling frequency that we can calculate; conversely, the higher w, the lower the resolution. Thus, according to the specific application or domain where our project will be run, it is possible to give higher

CYCLE register	w (ns)	Minimum bit rate (kbps)
1	$8 \cdot 10$	10^6
2	$8 \cdot 10^2$	10^5
3	$8 \cdot 10^3$	10^4
4	$8 \cdot 10^4$	10^3
5	$8 \cdot 10^5$	10^2
6	$8 \cdot 10^6$	10

Table 1: Minimum supported bit rate in kbps for each value of the CYCLE register and corresponding time w.

Register name	Description
CYCLE	Input register containing the time information for sampling
Q0_BITRATE	Output register containing bit rate for queue 0
Q1_BITRATE	Output register containing bit rate for queue 1
Q2_BITRATE	Output register containing bit rate for queue 2
Q3_BITRATE	Output register containing bit rate for queue 3
TOTAL_BITRATE	Output register containing the total bit rate

Table 2: New registers defined within the Input Bit Rate Calculator module.

importance to the sampling frequency rather than the resolution or vice-versa. This is accomplished by properly setting the value of the CYCLE register that we have introduced within the Register System. Table 1 shows the values taken into account for the CYCLE register, the related time w, and the minimum supported bit rate for each queue that can be captured. Therefore, if our application needs a higher sampling rate then we should use lower values of CYCLE as long as the current bit rate is supported. Thus, if for example the observed bit rate is lower than 1000 kbps and we want the highest sampling rate able to capture the bit rate, we need to use a value for CYCLE equal to 5.

Besides the CYCLE register, we have defined within the Register System five more registers (Table 2 lists them all) which store the resulting bit rate. In particular, we store the input bit rate of each single queue and the overall input bit rate of the router. Clearly, these registers may be read from any software module.

As far as the existing Verilog modules are concerned, we have slightly changed the $nf2\_core.v$, $mac\_grp\_regs.v$, and $nf2\_mac\_grp.v$ modules in order to forward the $reg\_file['MAC\_GRP\_RX\_QUEUE\_NUM\_BYTES\_PUSHED']$ signal to the User Data Path module. This signal has the same value corresponding to the MAC_GRP_i_RX-

_QUEUE_NUM_BYTES_PUSHED_REG registers for $i = \{0, \ldots, 3\}$. It counts the number of bytes that have been sent into the Rx queues. Within the User Data Path we have defined our Input Bit Rate Calculator which takes as input the signal indicated above.

The bit rate computation takes as input this signal for each queue and it is implemented by the state machine shown in Fig. 2. The state machine initially idles in the *Increment* state waiting for the amount of cycles corresponding to w (which depends on the CYCLE register value). When w expires it computes the bit rate for each input queue using Equation (1). The next state is *Computation* which calculates the overall bit rate. Finally, the last state is *Update* which stores in local variables the value of each MAC_GRP_i_RX_QUEUE_NUM_BYTES-_PUSHED_REG registers, for $i = \{0, \ldots, 3\}$. We let the reader notes that we have distributed the computation along three different states for time constraints purposes.

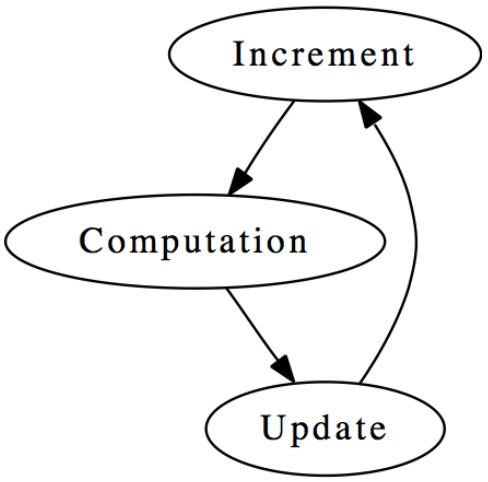

Figure 2: Diagram of state machine for the Input Bit Rate Calculator module.

3.2 Output Bit Rate Calculator

Similarly to the Input Bit Rate Calculator module, the Output Bitrate Calculator is a hardware module that provides the output bit rate for each Tx queue and the overall output bit rate as well. The same concepts and formulas discussed in Section 3.1 apply even in this case. Moreover, the same number of registers have been defined for the output bit rate and have a similar purpose as the ones in 3.1.

The Output Bit Rate Calculator module receives a signal from each

of the MAC output queues. This signal has the same value corresponding to the MAC_GRP_i_TX_QUEUE_NUM_BYTES_PUSHED-_REG registers for $i = \{0, \ldots, 3\}$. It counts the number of bytes that have been sent out of the Tx queues to Ethernet.

We have slightly changed the $nf2\_core.v$, $mac\_grp\_regs.v$, and $nf2\_mac\_grp.v$ modules in order to forward the $reg\_file['MAC\_GRP\_TX\_QUEUE\_NUM\_BYTES\_PUSHED']$ signal to the User Data Path module. Therefore, defining the Output Bit Rate Calculator module within the User Data Path allowed us to capture the signals above and process them. Then we have proceeded analogously to Section 3.1 using a similar state machine architecture.

3.3 Bit Rate EWMA Calculator

An exponential moving average (EMA), also known as an exponentially weighted moving average (EWMA), is a type of infinite impulse response filter that applies weighting factors which decrease exponentially. The weighting for each older data point decreases exponentially, never reaching zero. The formula that has been taken into account for the development of an EWMA module of the input bit rate is the following:

$$ewma\_bit\_rate_n = \alpha \times bit\_rate_n + (1 - \alpha) \times ewma\_bit\_rate_{n-1} \quad (2)$$

where

- the coefficient α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher α discounts older observations faster;
- bit_rate_n is the observation (overall bit rate) at a time period n; this value is forwarded by the Input Bit Rate Calculator described above;
- $ewma\_bit\_rate_{n-1}$ is the value of the EWMA bit rate at a time period $n - 1$.

We have introduced a new register, called ALPHA, which stores the value for α chosen by the user. The Bit Rate EWMA Calculator module gets an input signal from the Input Bit Rate Calculator module which corresponds to the overall input bit rate.

As far as the calculation of the Equation 2 is concerned, the reader notices that the α parameter could get any decimal value in the range [0,1]. Taking into account all the possible values in that range would have meant to complicate a lot the logic and therefore the performances of the module as floating point numbers multiplication is not standard in Verilog and requires very high CPU computational power. For such

ALPHA register	α	**1-α**	**operation**
0	1	0	bit rate$_n$
1	0.125	0.875	$(bit\_rate_n >> 3) +$ $((ewma\_bit\_rate_{n-1} >> 3) \times 7)$
2	0.25	0.75	$(bit\_rate_n >> 2) +$ $((ewma\_bit\_rate_{n-1} >> 2) \times 3)$
3	0.375	0.625	$((bit\_rate_n >> 3) \times 3) +$ $((ewma\_bit\_rate_{n-1} >> 3) \times 5)$
4	0.5	0.5	$(bit\_rate_n >> 1) +$ $(ewma\_bit\_rate_{n-1} >> 1)$
5	0.625	0.375	$((bitrate_n >> 3) \times 5) +$ $((ewma\_bit\_rate_{n-1} >> 3) \times 3)$
6	0.75	0.25	$((bitrate_n >> 2) \times 3) +$ $(ewma\_bit\_rate_{n-1} >> 2)$
7	0.875	0.125	$((\text{bit rate}_n >> 3) \times 7) +$ $(ewma\_bit\_rate_{n-1} >> 3)$

Table 3: Values for the ALPHA register and corresponding operations of the second member of Equation 2.

a reason, we have simplified the EWMA formula of Equation 2 by using right shift and multiplication operators instead of the regular division. On the one hand, proceeding like that, we could consider only a finite number of values for α; on the other hand, these values range from 0 to 1 in steps of 0.125 and give enough choice to the user for the EWMA calculation. Table 3 shows the value of the ALPHA register and the corresponding values for α, $1 - \alpha$, and the Equation 2 with only right shifts and multiplications.

3.4 Extended Rate Limiter

The NetFPGA V2.0 platform contains a rate limiter module for the second MAC Tx queue only. This module allows us to limit the output bit rate of the second MAC Tx queue according to 16 values (from 244 kbps to 1Gbps) written into one ad-hoc hardware register. In particular, within the rate limiter module, eight registers have been introduced (two per each output queue): RATE_LIMIT_i_ENABLE_REG and RATE_LIMIT_i_SHIFT_REG for i ranging in $\{0, \ldots, 3\}$. The former contains a boolean value indicating that the rate limiter is either enabled or disabled whereas the latter contains a value which limits the corresponding output queue bit rate. We have slightly changed the *rate_limiter.v*, *rate_limiter_regs.v* and *user_data_path.v* files in

order to take into account 32768 fine-grained values of the bit rate[2] (and not just 16) and to provide each output queue (not the second queue only) with such a rate limiter functionalities. Of course, we have also changed the java-based graphical user interface released together with the NetFPGA platform in order to take into account the 32768 bit rate levels. This allowed us to limit the output bit rate of each queue and try different settings in order to create the same conditions that occur during a congestion.

4 Experimentation

The experimentations we have carried out have been performed using two NetFPGA hosts systems. One of them ($\text{NetFPGA}_{bit\_rate\_calc}$) was loaded with the project we propose in this paper whereas the other (NetFPGA_{pkt_gen}) was loaded with the Packet Generator project [10]. The packet generator application allows Internet packets to be transmitted at line rate on up to four Gigabit Ethernet ports simultaneously. Data transmitted is specified in a standard PCAP file, transferred to local memory on the NetFPGA card, then sent on the Gigabit links using a precise data rate, inter-packet delay, and number of iterations specified by the user. Figure 3 shows the adopted topology.

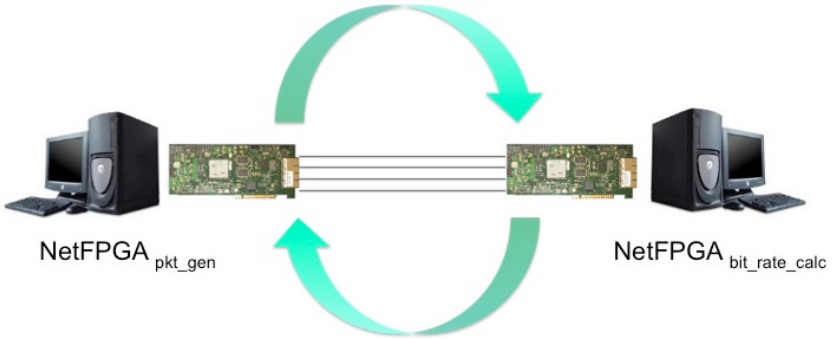

Figure 3: Topology used for the experimentation results. Network packets are sent from NetFPGA_{pkt_gen} to $\text{NetFPGA}_{bit\_rate\_calc}$ where they are sent back to NetFPGA_{pkt_gen}.

The four ports of the two NetFPGA boards were connected each other. Therefore, using the Packet Generator we sent network packets out of the four ports of NetFPGA_{pkt_gen} to $\text{NetFPGA}_{bit\_rate\_calc}$. The

[2]It was enough to replace within the *rate_limiter.v* Verilog file a shift operation with a multiplication.

packets received within NetFPGA$_{bit_rate_calc}$ were sent back again to NetFPGA$_{pkt_gen}$ in order to calculate the output bit rate value from NetFPGA$_{bit_rate}$. In order to send the packets from NetFPGA$_{pkt_gen}$ we had to carefully prepare the PCAP files by choosing appropriate IP and MAC addresses, and IP and MAC checksums. Moreover we had to appropriately set the MAC addresses of the Ethernet ports and IP addresses of NetFPGA$_{pkt_gen}$. In NetFPGA$_{bit_rate_calc}$ the configuration has been set up running the SCONE[3] software.

Finally, playing with the rate limiter on each of the output queues we could observe the behavior of our developed modules as far as the input bit rate and the output bit rate are concerned. In particular, we have developed a bash script which sends network packets from NetFPGA$_{pkt_gen}$ with an average rate of 2 Gbps; moreover we have written a C program which runs every 1000 ms within the NetFPGA$_{bit_rate_calc}$ and increases the rate limiter value of each output queue of a fixed amount. Clearly, the input bit rate read using our module had to be equal to that sent out from NetFPGA$_{pkt_gen}$ whereas the output bit rate value was lower according to what we fixed within the rate limiter of each queue. Figure 4 shows how the input and output bit rate values read using our modules changed according to the initial bit rate set within the NetFPGA$_{pkt_gen}$ and the values in the rate limiter module. At instant 0 the initial bit rate sent out from NetFPGA$_{pkt_gen}$ was set to about 2 Gbps and the rate limiter of each queue was set to 406.25 Mbps. Then, after each 1000 milliseconds, the rate limiter of each queue was incremented by 15.62 Mbps. After 6000 milliseconds, the rate limiter was set to 500 Mbps. Thus, as within the same interval time the input bit rate reached a value slightly lower than 2 Gbps (the maximum value which could be read by all the output queues each having the current rate limiter set to 500 Mbps), the output queues were able to read the entire bit rate; therefore the two curves in the figure overlap. The user notices that when the rate limiter value was changed, the time needed to make the change effective was about 500 μs which corresponds to 0.5 ms. That explains the vertical lines in the figure.

5 Device Utilization

The device utilization of the hardware component of our modules is almost identical to that of the Reference Router design and is displayed in Table 4.

[3]The router SCONE is a user level router that performs IPv4 forwarding, handles ARPs and various ICMP messages

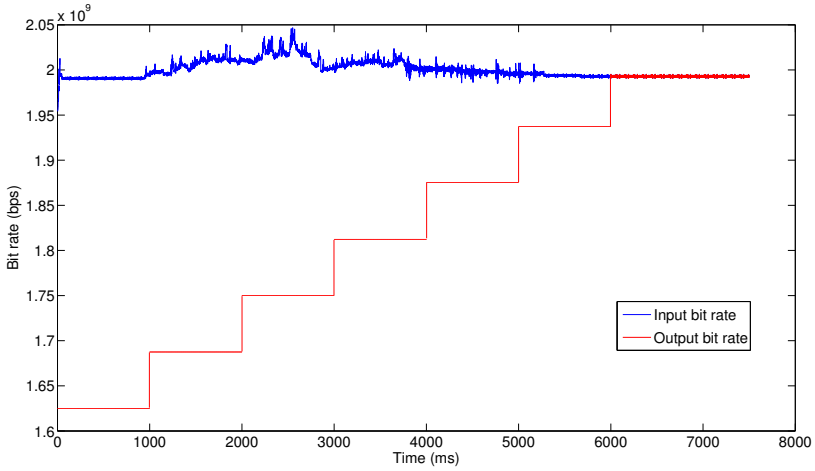

Figure 4: Variations of the output bit rate with respect to the input bit rate according to progressive values of rate limiter for each queue.

Resources	**XC2VP50 Utilization**	**Utilization Percentage**
Slices	14.160	59%
4-input LUT	19.503	41%
Flip Flops	12.387	26%
Block RAMs	27	11%
External IOBs	360	52%

Table 4: Device utilization for our modules.

6 Conclusions and Future Work

Our modules are implemented on the NetFPGA platform and they perform input, output and ewma bit rate computation. We have shown how our modules are effective with simple tests. The implementation process of our modules were simplified by the pipelined architecture of the Reference Router. Furthermore, by reusing the Reference Router design, the development time of our modules was greatly reduced as we did not have to start from scratch. Our code has been released, following the guidelines in [9], to the larger community for re-use, feedback, and enhancement. The wiki entry of the project can be seen at [6].

7 Acknowledgement

The work described in this paper was performed with the support of the ECONET project (low Energy COnsumption NETworks), funded by the EU through the FP7 call.

References

[1] *NetFPGA Team. NetFPGA website. http://netfpga.org.*

[2] *NetFPGA Projects . http://netfpga.org/ foswiki/bin/view/ NetFPGA/OneGig/ ProjectTable.*

[3] *NetFPGA Reference Router project. http://netfpga.org/ foswiki/bin/view/NetFPGA/ OneGig/ReferenceRouterWalkthrough.*

[4] *NetFPGA Register System. http://netfpga.org/ foswiki/ bin/view/NetFPGA/ OneGig/RegisterSystem.*

[5] *NetFPGA Walkthrough the Reference Design. http://netfpga.org/ foswiki/bin/view/ NetFPGA/OneGig/ Guide#Walkthrough_the_Reference_Design.*

[6] *NetFPGA Projects . http://netfpga.org/ foswiki/bin/view/ NetFPGA/OneGig/ InputOutputEwmaBitrate.*

[7] Mou-Sen Chen, Ming-Yi Liao, Pang-Wei Tsai, Mon-Yen Luo, Chu-Sing Yang, and C. Eugene Yeh. Using netfpga to offload linux netfilter firewall. In *2nd North American NetFPGA Developers Workshop; Stanford, CA; August 13*, 2010.

[8] Michael Ciesla, Vijay Sivaraman, and A. Seneviratne. Url extraction on the netfpga reference router. *Developers Workshop 2009*, 2009.

[9] G. A. Covington, G. Gibb, J. Naous, J. Lockwood, and N. McKeown. Encouraging reusable network hardware design. *IEEE*

International Conference on Microelectronics System Education (MSE), San Francisco, CA, 2009.

[10] G. Adam Covington, Glenn Gibb, John W. Lockwood, and Nick Mckeown. A packet generator on the netfpga platform. In *Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, FCCM '09, pages 235–238, Washington, DC, USA, 2009. IEEE Computer Society.

[11] Glen Gibb, John W. Lockwood, Jad Naous, Paul Hartke, and Nick McKeown. Netfpga. an open platform for teaching how to build gigabit-rate network switches and routers. *IEEE Transactions on Education*, 2008.

[12] Andrew Goodney, Shailesh Narayan, Mengchen Wang, Peigen Sun, Vivek Bhandwalkar, and Young H. Cho. Netfpga logic analyzer. In *2nd North American NetFPGA Developers Workshop; Stanford, CA; August 13*, 2010.

[13] Adwait Gupte and John Lockwood. Precise latency comparison module for the netfpga. In *2nd North American NetFPGA Developers Workshop; Stanford, CA; August 13*, 2010.

[14] Y.S. Hanay, A. Dwaraki, and T. Wolf. High-performance implementation of in-network traffic pacing. In *High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference*, pages 9–15, 2011.

[15] Jinghe Jin, Nazarov Nodir, Chaetae Im, and Seung Yeob Nam. Mitigating http get flooding attacks through modified netfpga reference router. In *1st Asia NetFPGA Developers Workshop; Daejeon, Korea; June 14*, 2010.

[16] Abdul Kabbani and Masato Yasuda. Data center quantized congestion notification (qcn): Implementation and evaluation on netfpga. In *1st Asia NetFPGA Developers Workshop; Daejeon, Korea; June 14*, 2010.

[17] Bokil Kanchan. Remodeling the netfpga architecture for content processing and filtering. In *Developers Workshop; Stanford, CA; August 13*, 2010.

[18] Yu-Kuen Lai, Nan-Cheng Wang, Tze-Yu Chou, Chun-Chieh Lee, Theophilus Wellem, and Hargyo Tri Nugroho. Implementing online sketch-based change detection on a netfpga platform. In *1st Asia NetFPGA Developers Workshop; Daejeon, Korea; June 14*, 2010.

[19] J.W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga - an open platform for gigabit-rate network switching and routing. In *In International Conference on Microelectronic Systems Education*, 2007.

[20] Alfio Lombardo, Carla Panarello, Diego Reforgiato, Enrico Santagati, and Giovanni Schembra. A module for packet hijacking in netfpga platform. In *Proceedings of the 2011 14th Euromicro Conference on Digital System Design*, DSD '11, pages 283–286, Washington, DC, USA, 2011. IEEE Computer Society.

[21] Alfio Lombardo, Diego Reforgiato, and Giovanni Schembra. An accelerated and energy-efficient traffic monitor using the netfpga (abstract only). In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 277–277, New York, NY, USA, 2011. ACM.

[22] Danilo Misovic, Nikola Ljumovic, Milutin Radonjic, and Igor Radusinovic. Implementation of the crosspoint-queued switch's output controller on the netfpga platform. In *Proceedings of ELMAR 2011*, ELMAR '11, pages 235–2238, 2011.

[23] Hamed Tabatabaei and Yashar Ganjali. Preserving pacing in real networks - an experimental study using netfpga. In *2nd North American NetFPGA Developers Workshop; Stanford, CA; August 13*, 2010.

[24] Minglong Zhang, Hui Li, Fuxing Chen, Hanxu Hou, Huiyao An, Wei Wang, and Jiaqing Huang. A general co/decoder of network coding in hdl. In *Network Coding (NetCod), 2011 International Symposium*, pages 1–5, 2011.