Design of the Switching Controller for the High-Capacity Non-Blocking Internet Router

Miloš Petrović, *Student Member, IEEE,* Aleksandra Smiljanić, *Member, IEEE,* and Miloš Blagojević

***Abstract*—The sequential greedy scheduling (SGS) algorithm is a scalable maximal matching algorithm. This algorithm was conceptually proposed and well received since it provides non-blocking in an Internet router with input buffers and a cross-bar, unlike other existing implementations. In this paper, we implent a new design of the SGS algorithm, and determine its exact behaviour, performance and QoS that it provides. We examine different design options and measure the performance of their implementations in terms of their scalability and speed. It will be shown that multiple scheduler modules of a terabit Internet router can be implemented on a low-cost FPGA device, and that the processing can be performed within the desired time slot duration. Proper functioning of the implemented scheduler was confirmed through thorough software and hardware testing.**

I. Introduction

The fast growth of bandwidth demand on the Internet has led to a need for high-capacity packet routers, with large number of ports and high port speeds. Routers with input buffers are the most scalable single-stage routers. In this architecture, packets are split into cells of fixed length and stored at the input ports. Based on the information about the outstanding packets, the scheduler determines the cross-bar configuration in each time slot, i.e. the input-output pairs that should be connected. When the number of ports increases, the allocation of outputs to inputs in these routers becomes computationally intensive [1]-[4].

The router with input buffers is non-blocking if appropriate scheduling algorithms are implemented. But, the algorithm must not limit the router scalability either. It has been recognized that maximal matching algorithms provide non-blocking while requiring significantly lower computing complexity compared to maximum matching algorithms [1],[5]-[8]. The maximal matching between inputs and outputs do not leave input-output pairs unmatched if there is traffic between them [3],[5],[9]. The Sequential Greedy Scheduling (SGS) algorithm [1], [5] is a maximal matching algorithm that provides non-blocking through a cross-bar with the speedup of two, when the traffic is policed, and consequently provides delay guarantees to the sensitive applications, as well as flexible admission control.

This work was supported by the Ministry of Science and Enviromental Protection of the Republic of Serbia, and company Pupin Telecom DKTS.

M. Petrović is with the School of Electrical Engineering (Belgrade University), Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia. (e-mail: pmilos@etf.bg.ac.yu)

A. Smiljanić is with the School of Electrical Engineering (Belgrade University), Serbia, and the Stony Brook University, USA. (e-mail: aleksandra@etf.bg.ac.yu)

M. Blagojević is with the School of Electrical Engineering (Belgrade University), Serbia. (e-mail: milosdb@etf.bg.ac.yu)

The SGS algorithm can be implemented using pipeline technique, and involves a scheduler with a simple structure. For a router with N ports, the scheduler has N modules and each of the modules communicates only with adjacent scheduler modules. The SGS algorithm is performed in N steps. In each step, one of the inputs chooses the first output for which it has packets to send, and which was not assigned in previous steps. Then, this input updates the set of available outputs and forwards it to the next input in the chain. Using pipeline technique, the schedule is calculated during multiple time slots. In each time slot, the scheduler calculates in parallel schedules for N future time slots. Thus, each input port is given significantly longer time to perform the output selection.

The scheduler should perform an output selection so that the overall pipeline delay is a negligible part of the packet delay. The packet delay (including queuing delay) depends on the cell duration and the granularity of bandwidth reservations. It is equal to the policing interval (frame) in unicast routers with input buffers controlled by the SGS [1], [5]. Policing interval equals $F \cdot T_c$, where F is the number of cells per policing interval, and T_c is the cell duration. We have shown that the frame size should be $F >> N$, so that not much bandwith is wasted when many ports exchange small amounts of traffic [10]. If T_s denotes the time required for the output selection, then the pipeline delay is NT_s. The pipeline delay is negligible if $NT_s << FT_c$, which holds when $T_s \approx T_c$ if we assume high worst-case efficiency, i.e. $F >> N$. So, when the pipeline technique is introduced, a processing time of each input is relaxed N times (from T_c/N to T_c) without sacrificing the performance.

Scalability of the scheduling algorithm may be estimated, but it can be ascertained only when the algorithm is actually implemented. Field programmable gate arrays (FPGAs) are convenient for the implementation of the SGS algorithm. They are cost-effective, and relatively easy to program. Testing of the FPGA design is significantly simplified compared to the application-specific integrated circuit (ASIC) design, which speeds up the FPGA design and reduces its cost. Performance of the implementation (its speed and scalability) depends on software tools used for synthesis, and placement and routing on the FPGA device [10]-[12]. Since in the pipelined SGS algorithm, the cell duration, T_c, should be greater than the output selection time, T_s, and the delay is proportional to the cell duration, the output selection time should be as low as possible in order to provide low packet delays.

Other implementations of the schedulers for the high-capacity packet routers have been previously suggested [2]-[3]. These implementations, however, are neither scalable since

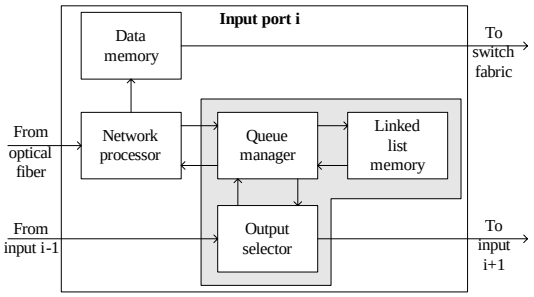

Fig. 1. The internal architecture of an input port i

the complexity of the design increases with the square of the number of ports, nor non-blocking for all types of traffic.

In this paper, we propose a new design of a scalable non-blocking scheduler for the high-capacity packet routers. Particular scheduler components are first described. Next, we present the optimizations of the scheduler subcomponents, and prove their correct functioning. Then, we propose two design options for the SGS scheduler, and examine the performance of their implementation in terms of the speed and scalability. The number of scheduler modules that can fit one FPGA device and the output selection time T_s are calculated for various router sizes. The implemented design is thoroughly tested using both software and hardware checks.

II. Design of the Scheduler

The internal architecture of the input port is given in Fig. 1. The input port consists of several components: network processor, data memory, linked list memory, queue manager, and output selector. When a packet arrives to the router, its destination IP address is read by network processor (NP) and the router output to which the packet should be sent is determined. The NP also divides packets into smaller fixed length cells and stores them in the appropriate virtual output queue (VOQ). In each input buffer, there are N VOQs that comprise cells bound for particular outputs. Data memory stores the incoming packets/cells until they are scheduled and sent through the switching fabric. Linked list memory stores the data memory addresses of cells in VOQs. Queue manager performs operations on virtual queues. Output selector calculates the schedule for the outstanding cells and stores it in output memory until the cells are read.

We have implemented scheduler modules comprising queue manager, linked list memory and output selector on Altera Cyclone II FPGA. These components will be presented in more details in the following subsections. Multiple scheduler modules can be placed on a single FPGA device.

A. Linked List Memory

Cells in data memory that are bound for the same destination form a VOQ. There are N VOQs, corresponding to N outputs. Linked list memory stores addresses of cells in different VOQs and addresses of empty locations. Each location in one virtual queue linked list (VQL) contains the address of the next location in that VQL, or zero (NULL) if no more locations belong to the VQL. Similarly, each location in the empty queue linked list (EQL) contains the address of the next location in EQL. The size of the linked list memory is defined by the number of cells that ought to be stored in the data memory of the router. The data memory should be able to store the number of cells equal to the frame length F. Thus, the linked list memory has F locations.

We designed the linked list memory so that it uses M4K memory blocks of the Altera Cyclone II FPGA.

B. Queue Manager

Queue manager performs operations when a cell arrives to the queue manager, when it is scheduled by the output selector, or when it departs the router. The queue manager stores pointers to VQLs. There are three pointers to each VQL: to the beginning of the VQL, the first unscheduled packet in the VQL, and the end of the VQL. Similarly, the EQL is managed with two pointers: to the beginning and the end of the EQL. The pointers are updated each time one of the operations is performed. Multiple operations are carried out in the same time slot, and in general on different VQLs.

At the beginning, all memory locations belong to EQL, and each location contains the address of the next location in the memory (except the last that points to NULL). The memory location is removed from the beginning of the EQL to the end of the VQL of some output, when a cell bound for that output arrives to the router. The cell is stored to the address from this memory location. When input sends a cell to some output, the cell address in the data memory is read from the heading location of the corresponding VQL, and this memory location is added to the end of EQL. When a cell is scheduled, the corresponding VQL is updated. If the pointer to the first unscheduled cell from the same VQL points to the last cell in that VQL, the pointer is set to NULL. Else, the pointer to the first unscheduled cell is set to point to the next element in that VQL.

The queue manager was designed in VHDL, and it can be scaled easily for different router sizes.

C. Output Selector

The output selector of the associated input schedules a cell by choosing the first available output from the set of outputs for which the given input has cells to send. The result of the scheduling process is a vector in which only one bit, which corresponds to the scheduled queue, is set to logical one.

The structure of the optimized output selector is shown in Fig. 2 [12]. Bits denoted by R contain information about cells of a particular input, which participate in the contention process. Bit R_j is set to '1' only if there are unscheduled cells for the j-th output port, and the j-th output port was not selected by previous input ports. As a result of the scheduling process, Q_j is set to '1' when the j-th output port is chosen by the given input. The information about the remaining output ports is forwarded to the next input port in the chain.

The output selector is implemented recursively. The two-port output selector is shown in Fig. 2a. The output selector for a router with 2^{k+1} output ports can be built recursively

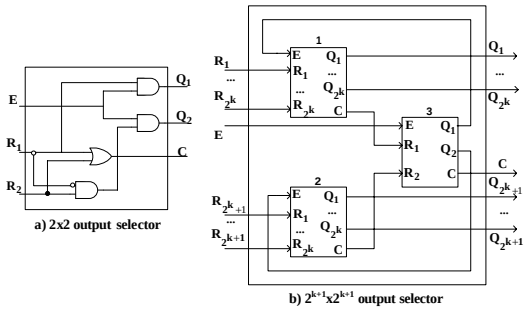

Fig. 2. The output selector structure

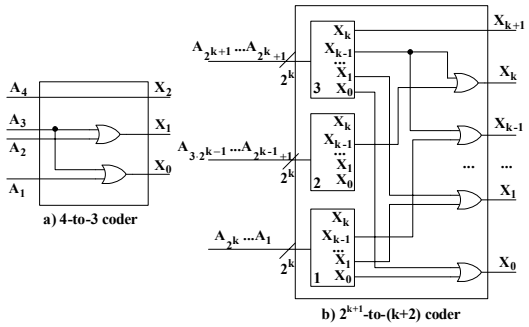

Fig. 3. The coder structure

by using two output selectors for 2^k ports and one two-port output selector, as shown in Fig. 2b.

Theorem 1: For the $2^k \times 2^k$ output selector the following equations hold:

$$C = \sum_{m=1}^{2^k} R_m, \tag{1}$$

$$Q_i = E \cdot R_i \cdot \prod_{j=1}^{i-1} \overline{R_j}, 1 \leq i \leq 2^k \tag{2}$$

that is, the output selector chooses the first available output for which an associated input control module has cells to send.

Proof: The theorem 1 was proven in [12]. ■

The Q vector is forwarded to the optimized coder shown in Figure 3, which determines the position of a logical one in that vector and thus the output that has been scheduled for the observed time slot. The coder allows the output selector to propagate the information about the scheduled queue with less bits. The output selector and the coder are connected in the following manner: the j-th output of the output selector drives the j-th input of the coder. When Q_j ='1', then A_j ='1', and the coder returns j coded binary. The coder is also implemented recursively, as shown in Figure 3.

Theorem 2: For the 2^k-to-$(k+1)$ coder, when A_x ='1' ($1 \leq x \leq 2^k$), the outputs of the coder X_i ($0 \leq i \leq k$) comprise the binary representation of x:

$$x = \sum_{i=0}^{k} X_i \cdot 2^i, \tag{3}$$

and when all the inputs are equal to zero, the coder's output is zero, as well.

Proof: We have proven the theorem 2 in [12]. ■

The number of the scheduled output is sent to the queue manager, and also stored in the output memory until the time the cell should be read, according to the pipeline technique. The output selector for the input i in time slot k calculates the schedule and reserves the output for the time slot:

$$t_f = k + N + 3 - i. \tag{4}$$

Therefore, the numbers of the scheduled VOQs need to be stored in the output memory until the time slot when they need to be read by the queue manager. The output memory was designed to use M4K memory blocks functioning as an embedded shift register. The additional delay of three time slots needs to be introduced (since the minimum tap distance of the embedded shift register is three). The delay incurred in this way is still a negligible part of the total packet delay.

III. The Performance Analysis

We have implemented two different design options of the scheduler that runs the SGS algorithm. In this section, we will discuss their performance and measure it in terms of the design scalability, i.e. the number of scheduler modules that can fit one FPGA device. We will also examine the speed of the design, i.e. the minimum achievable output selection time, T_s. The Altera Cyclone II EP2C35F672C6 device [13] that we used for the implementation consists of 33216 logic elements (LEs) grouped into logic array blocks (LABs), 100 M4K memory blocks containing 4608 memory bits each, and 475 user IOs on the periphery.

We have shown in [12] that the maximum number of scheduler modules, which can fit a single FPGA device, does not change if the optimized structures are used in place of the simpler VHDL design. However, it was proven that the minimum output selection time does significantly decrease when the optimized structures are implemented. The gain of the optimization was shown to increase with router size.

A. Case A

The state machine for the queue manager operation comprises four states: swait, write, schedule, and read, and the state transitions are unconditional, i.e. in each clock cycle the state machine switches to the next state. The swait state is needed to ensure the correct functioning of the controller, since the synchronous operations of the M4K memory require one additional cycle between an address setup cycle, and a cycle in which the data is read. The pointers to VQLs are implemented using LEs of the Altera Cyclone II.

TABLE I
Resource utilization and timing characteristics, $F = 8N$

N	N_P	LE	$M4K$ [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_s [ns]
16	16	17331	32	339	/	121.1	33
32	17	32096	34	441	LE	84.1	47.4
64	8	29507	16	331	LE	82.6	48.4
128	3	24599	12	215	LE	68.4	58.4

To obtain a rate granularity for high efficiency, the frame length should be at least ten times the number of ports, as we pointed out before. Tables I and II contain the FPGA resource utilization, timing characteristic and limiting factors when the frame size is eight and sixteen times greater than the

TABLE II
RESOURCE UTILIZATION AND TIMING CHARACTERISTICS, $F = 16N$

N	N_P	LE	$M4K$ [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_s [ns]
16	16	18895	32	371	/	106.5	37.5
32	15	31300	30	427	LE	97.2	45.1
64	8	31736	32	347	LE	81.2	49.2
128	3	26658	21	221	LE	69.4	57.6

number of ports, respectively, for different number of input ports N, and for maximum number of scheduler modules N_P which could have been fit onto a single device. The minimum output selection time equals the time required for the linked list updates, since the time required for the output selector and coder to schedule a cell of a given input is not critical. Thus, the minimum output selection time can be calculated from the maximum clock frequency as: $T_s = 4/f_{MAX}$.

The tables also show in the column "lim" which factor limits the number of scheduler modules that fit one FPGA device: the number of LEs (LE), the number of memory blocks (mem), or the pin number (pins). The number of used pins can be calculated as:

$$pins(N, N_P, F) = (2\lceil log_2 F\rceil + log_2 N + 1)N_P + 2N + 3, \quad (5)$$

because $log_2 N + 1$ pins per module are used for incoming cell's VOQ numbers, $2\lceil log_2 F\rceil$ pins per module for addresses to which network processor stores and reads cells from, $2N$ pins for control bits, and three pins for two clocks and a reset signal. The number of scheduler modules is limited by memory to 50, since there are 100 M4K blocks, and every scheduler module uses at least two M4K blocks (one for linked list memory and one for output memory). This number can further decrease, when some memories span multiple M4K blocks. The number of M4K blocks required for the linked list memory and the output memory can be calculated as:

$$mem(N, N_P, F) = (\lceil F\lceil log_2 F\rceil/4096\rceil + 1)N_P, \quad (6)$$

since the linked list memory has the size of $F\lceil log_2 F\rceil$ bits per module and the output memory in none of the observed cases reserves more than one M4K block per module. The linked list memory reserves $\lceil F\lceil log_2 F\rceil/4096\rceil$ M4K blocks per module, since the Quartus II software we used for Altera FPGA devices limits the use of M4K block to 4096 bits. So, the maximum number of control modules per chip should satisfy the inequalities:

$$\begin{aligned} &pins(N, N_P, F) \leq 475, \\ &mem(N, N_P, F) \leq 100. \end{aligned} \quad (7)$$

When some of the equalities in (7) is first met, the corresponding chip resource is limiting the number of modules on the FPGA chip. The number of LEs that are used for the implementation cannot be calculated deterministically.

It can be observed from Table I and Table II that the maximum clock frequency of the design somewhat decreases with the increase of the number of ports because it causes the increase of the number of required LEs. However, the minimum output selection time does not change significantly with the frame length F, as expected, and will remain well below 100ns for the finest granularities as desired.

B. Case B

The state machine for the queue manager operations has three states, and the state transitions are unconditional. Operations on VQLs are distributed over these three cycles (states) so that the fourth state required in case A is avoided. To increase the scalability of the queue manager, the pointers have been implemented in the M4K blocks of the Cyclone II FPGA. Since some of the pointer updates require multiple memory accesses and due to the synchronous operations of M4K memory, pointer updates cannot be performed within three states. So, we access the pointer memory at twice the speed of the linked list memory, and a separate state machine with six states and also unconditional transitions regulates the pointer operations. The two state machines are synchronized.

To increase the number of modules that can fit one FPGA, the pins' signalling speed is increased two times. The data memory addresses to be written to and read from, which are exchanged between network processor and the implemented scheduler, are similarly sped up. In this way, the number of pins required to ensure the given throughput of control information is decreased nearly two times. In this case, the minimum output selection time can be calculated from the maximum clock frequency as $T_s = 6/f_{MAX}$, since the time slot comprises six cycles of the clock that manages pointer updates.

The number of modules that can fit the FPGA may be limited by the number of pins, the number of M4K blocks or the number of LEs. When the pins are sped up, the number of used pins can be calculated as:

$$pins(N, N_P, F) = (2\lceil\lceil log_2 F\rceil/2\rceil + log_2 N + 1)\cdot N_P + N + 4, \quad (8)$$

because the number of pins for addresses is halved to $2\lceil\lceil log_2 F\rceil/2\rceil$ pins per module, for control bits to N pins, and one additional pin for the clock is used.

The number of scheduler modules is limited by memory to 20 since there are 100 M4K blocks, and every scheduler module uses at least five M4K blocks (three memories for pointers, one for linked list memory, and one for output memory). This number further decreases when the linked list memory spans multiple M4K blocks. The number of utilized M4K blocks is described by equation:

$$mem(N, N_P, F) = (\lceil F\lceil log_2 F\rceil/4096\rceil + 4)\cdot N_P, \quad (9)$$

because in the cases observed neither pointer memories nor output memories reserve more than one M4K block.

Again, the maximum number of control modules that can fit one device should fulfill inequalities (7). When some of the equalities in (7) is first met, the corresponding chip resource (pins or memory blocks) is limiting the number of modules on the FPGA chip.

The resource utilization, timing characteristics and limiting factors for $F = 8N$ and $F = 16N$ are given in tables III and IV respectively. It can be observed from the tables that the minimum time slot duration slowly increases with the router size and with the frame length. For $N = 16$, the increase in the speed of the pins was not necessary since the entire switching controller fits the chip.

TABLE III
RESOURCE UTILIZATION AND TIMING CHARACTERISTICS, $F = 8N$

N	N_P	LE	$M4K$ [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_s [ns]
16	16	4866	80	340	/	138.2	43.4
32	20	8833	100	316	mem	133.7	44.9
64	20	14202	100	408	mem	121.5	49.4
128	14	17626	98	384	mem	119.5	50.2

TABLE IV
RESOURCE UTILIZATION AND TIMING CHARACTERISTICS, $F = 16N$

N	N_P	LE	$M4K$ [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_s [ns]
16	16	5122	80	372	/	130.1	46.1
32	20	9155	100	356	mem	129.9	46.2
64	14	10276	98	306	mem	121.2	49.5
128	10	12816	100	332	mem	116.9	51.3

The minimum time slot is slightly longer than in the case A for smaller routers, i.e. $N = 16$. However, as the router size increases, the minimum time slot duration becomes shorter than in case A, and the number of scheduler modules that can fit the FPGA in case B is significantly larger than in case A. That is, for $F = 16N$, in case B ten modules fit one chip for $N = 128$; while in case A three modules fit the chip for $N = 128$. The minimum time slot is limited to T_s=51.3ns in case B, which is still well below 100ns.

TABLE V
MAXIMUM NUMBER OF SCHEDULER MODULES N_P PER CHIP

	$F = 8N$			$F = 16N$		
N	ps	nops	mem	ps	nops	mem
16	16	16	16	16	16	16
32	31	18	20	29	17	20
64	25	13	20	23	12	14
128	19	7	14	18	7	10

In Table V, we calculate the maximum number of scheduler modules that would fit one FPGA device, using formulas (5), (8), and (9). The cases with (ps) and without pins' speedup (nops) are considered. In case A, the number of LEs is main scalability limitation. In case B, the scalability significantly improves when the pins are sped up two times, and the number of M4K blocks becomes a limitation shown in column "mem".

IV. SIMULATION AND TESTING

Both design options were tested in software and in hardware, in order to prove their correct functioning. First, the built-in Quartus II simulation tool was used, and the designs were tested for different sets of input stimuli. In order to reliably perform the testing, a software was developed which generates random stimuli, reads the Quartus II simulation results and compares them with the expected values from the algorithm emulation written in C. The designs were verified for diffent router sizes, frame durations, and simulation lengths [11].

Then, the hardware testing was performed. The designs were downloaded into the Cyclone II chip on the Altera DE2 development board. On-board memory was used to store input stimuli, and the testing was performed in real-time, under minimum timing requirements, as given in Tables II-IV. The output results were recorded by the Tektronix TLA5203 logic analyzer, and then, as before, compared with the expected results from the algorithm emulation. In all the cases, the correct functioning of the switching controller was confirmed.

V. CONCLUSION

In this paper, we proposed the design of the scheduler for the non-blocking Internet router based on the SGS algorithm, and presented its implementation. We optimized the scheduler subcomponents, so as to provide lower packet delays, and proved their correct functioning. It was shown that placing pointers to virtual queue linked lists into the FPGA memory and speeding up the pins significantly improves the design scalability, and the processing speeds in the larger routers. The preferred design option is highly scalable: up to 14 scheduler modules of an 128×128 router can be placed on a single low-cost FPGA device. The port bit rate is limited by the network processor capability, and available network processors handle the port speed of 10Gb/s. The scheduler modules designed on one low-cost Altera FPGA may control router with hundreds of ports, i.e. with terabit capacity, which confirms anticipated scalability of the SGS algorithm. The output selection time remains below 60 ns in discussed high-capacity Internet routers. The proper functioning of the scheduler was confirmed through thorough software and hardware testing.

The implemented scheduler can be used in high-capacity routers that provide delay guarantees for sensitive applications, and the fine granularity of reservations.

REFERENCES

[1] A. Smiljanić, "Flexible bandwidth allocation in terabit packet routers," in *Proc. IEEE HPSR (Best Paper Award),* June 2000, pp. 233-241.

[2] H. J. Chao, "Saturn: A terabit packet router using dual round-robin," *IEEE Commun. Mag.* vol. 38, no.12, pp. 78-84, December 2000.

[3] N. McKeown *et al.*, "The Tiny Tera: A packet router core," *IEEE Micro*, vol. 17, no. 1, pp. 26-33, Jan.-Feb. 1997.

[4] Y. Tamir, H. C. Chi, "Symmetric crossbar arbiters for VLSI communication routers," *IEEE Trans. Parallel Distrib. Syst.,* vol. 4, no. 1, pp. 13-27, 1993.

[5] A. Smiljanić, "Flexible bandwidth allocation in high-capacity packet routers," *IEEE/ACM Trans. Netw.,* pp. 287-293, April 2002.

[6] A. Smiljanić, "Bandwidth Reservations by Maximal Matching Algorithms," *IEEE Commun. Lett.,* pp. 177-179, March 2004.

[7] G. Dai, B. Prabhakar, "The throughput of data routers with and without speedup," in *Proc. IEEE INFOCOM 2000,* pp. 556-564.

[8] E. Leonardi, M. Mellia, A. Marsan, F. Neri, "Stability of maximal matching scheduling in input-queued cell routers," in *Proc. IEEE ICC 2000,* pp. 1758-1763.

[9] T. E. Anderson, S. S. Owicki, J. B. Saxe, C. P. Thacker, "High-speed router scheduling for local-area networks," *ACM Transactions on Computer Systems,* vol. 11, no. 4, pp. 319-352, November 1993.

[10] M. Petrović, A. Smiljanić, "Design of the Scheduler for the High-Capacity Non-Blocking Packet router," in *Proc. IEEE HPSR 2006,* pp. 397-404.

[11] M. Petrović, M. Blagojević, V. Joković, A. Smiljanić, "Design, implementation, and testing of the controller for the terabit packet router," in *Proc. IEEE ICCCAS 2006,* Vol. 3, pp. 1701-1705.

[12] M. Petrović, A. Smiljanić, "Optimization of the Scheduler for the Non-Blocking High-Capacity Router," *IEEE Commun. Lett.,* vol. 11, no. 6, June 2007.

[13] Altera Corporation, "Cyclone II Device Handbook", February 2007, http://www.altera.com/literature/hb/cyc2/cyc2_ cii5v1_ 01.pdf