Design of the Scheduler for the High-Capacity Non-Blocking Packet Switch

Miloš Petrović[1], Aleksandra Smiljanić[1,2]
[1]*Belgrade University, Belgrade, Serbia and Montenegro*
[2]*Stony Brook University, New York, USA*
pmilos@ieee.org, aleks@ieee.org

***Abstract*— The sequential greedy scheduling (SGS) is a scalable maximal matching algorithm that provides non-blocking in a packet switch with input buffers and a cross-bar. In this paper, we propose the design of the SGS scheduler, and present its FPGA implementation. We examine different design options and measure these implementations in terms of their scalability and speed. It will be shown that multiple input modules of a terabit packet switch can be implemented on one low-cost FPGA device and that the processing can be performed within desired time slot duration.**

I. Introduction

The fast growth of bandwidth demand on the Internet has led to a need for high-capacity packet switches and routers, with large number of ports and high port speeds. Switches with input buffers are the most scalable single-stage switches. In this architecture, packets are stored at the switch inputs. 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 switches becomes computationally intensive [1]-[5].

It has been recognized that maximal matching algorithms provide non-blocking while requiring significantly lower computing complexity compared to maximum matching algorithms [1], [6]-[9]. Sequential Greedy Scheduling (SGS) [1], [6] is a maximal matching algorithm that provides non-blocking through a cross-bar with the speedup of two, and consequently provides delay guarantees to the sensitive applications, as well as flexible admission control.

The SGS algorithm can be implemented using pipeline technique, and involves a scheduler with simple structure. The scheduler has N input modules and each of the modules communicates only with adjacent input modules, where N is the number of switch ports. The SGS algorithm is performed in N steps. In each step, one of the inputs chooses one of the outputs for which it has packets to send, and which was not assigned in the previous steps. Then, this input module updates the set of available outputs and forwards it to the next input module. Using pipeline technique, the schedule is calculated during multiple time slots, and in each time slot the central controller calculates in parallel schedules for N future time slots. So, each input port is given more time to choose an output. The advantage of this approach is that an input can have significantly longer time to perform the output selection.

The scheduler should perform a selection so that overall pipeline delay is negligible part of the packet delay. The packet delay depends on the cell duration and the granularity of bandwidth reservations. It is equal to the policing interval (frame) in unicast switches with input buffers controlled by the SGS [1], [6]. Policing interval equals $F \cdot T_c$ where F is the number of cells per policing interval, and T_c is the cell duration. The minimum bandwidth that can be allocated to some input-output pair is one cell per policing interval. This corresponds to the rate granularity of $G = B/F$, where B is the port bit-rate. The granularity should be sufficiently low so that not much bandwidth is wasted when many ports exchange small amounts of traffic. In the worst case, one port exchanges negligible traffic with $N-1$ out of N ports, and these flows are assigned one cell per policing interval. The total bandwidth wasted is $(N-1) \cdot G = (N-1) \cdot B/F$. The wasted portion of port bit-rate is negligible if $(N-1) \cdot B/F << B$, i.e. if $F >> N$ (or $F \geq 10N$). 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 the high worst case efficiency, i.e. $N << F$. 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.

The packet delay significantly increases in multicast packet switches with input buffers. Only one currently known practical algorithm that provides a non-blocking multicast switch with input buffers utilizes packet forwarding. Namely, a multicast packet is transmitted in each policing interval only to the limited number of destination outputs which forward the packet to the limited number of remaining destination outputs in the following policing interval [10]. In this case, the delay is multiplied by $\log_2 N$. In order to keep the delay acceptable for most sensitive applications in high-capacity switches, the cell duration should be lower than 100ns [10].

In this paper, we propose the design of a scheduler that is based on the SGS algorithm, implement the proposed design, and examine the performance of the implemented design. The scheduler comprises the linked list memory, the queue manager and the output selector. These particular scheduler components will be first described. Then, we present two design options for the SGS scheduler and discuss their performance. These designs are implemented using field programmable gate arrays (FPGA) of Altera Cyclone family. Altera Cyclone

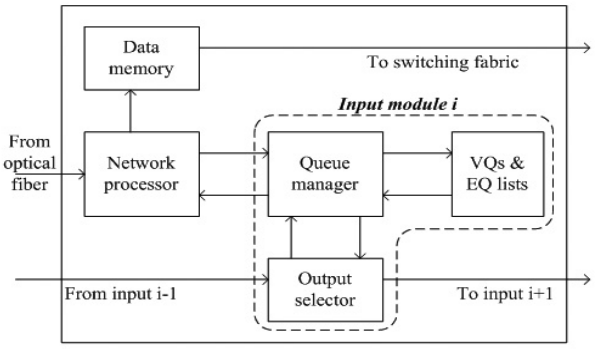

Fig. 1. The internal architecture of an input port i

family of FPGAs is a low-cost family, which maintains high performance [11]. Considered designs differ in terms of the placement of the controller elements, the processing order, and the speed of input/output registers. The number of input modules that can fit one FPGA device for various switch sizes is examined. Also, the output selection time T_s is calculated for various switch sizes and design parameters.

II. SGS Controller Design

When a packet arrives to the switch, its IP destination address is read, the switch output to which the packet should be sent is determined, and the packet is divided into smaller fixed length cells and stored into the appropriate virtual output queue (VOQ). In each input buffer, there are N VOQs which comprise cells bound for particular outputs. The internal architecture of the input port is given in Fig. 1. The input port comprises several components: the network processor (which processes the packets and determines their destinations from the headers, i.e. the output ports of the switch to which the packets should be forwarded, and divides the packets into cells), the data memory (which stores the incoming packets/cells until they are scheduled and sent through the switching fabric), the linked list memory (which stores the data memory addresses of cells in VOQs), the queue manager (which performs operations on virtual queues), and the output selector (which calculates the schedule for the incoming cells and stores this information in the output memory until the cells are read). We have implemented the input module comprising queue manager, the linked list memory and the output selector on Altera Cyclone EP1C20F400C6 FPGA, and these components will be presented in more details in the following subsections. Multiple input modules can be placed on a single FPGA device.

The EP1C20F400C6 device that we used for the implementation consists of 20060 logic elements (LEs) grouped into logic array blocks (LABs), 64 M4K memory blocks containing 4608 memory bits each (294912 bits total), 301 IOs on the periphery, two PLLs, and some configuration logic connected through a programmable routing fabric [11]. Cyclone logic element comprises a four input look-up table (LUT), a register, and dedicated arithmetic circuitry. It can operate in three different modes simultaneously. LUT and register can operate in parallel, LUT can drive the register input, and register can drive one of LUT inputs. M4K memory blocks can be organized to have widths from 1 to 36 bits, and operate in five different modes: single-port memory, simple dual-port memory, true dual-port memory, embedded shift register and ROM. M4K memory block has a maximum clock frequency of 200 MHz, while the maximum EP1C20F400C6 device clock frequency is 400 MHz. Groups of 10 LEs (one LAB) share control signals and a local routing structure. Varying packing levels of effort can be specified in Quartus II software, where the packing level determines the trade-off between the speed of the design and the area utilization of the device.

A. Linked List Memory

It has been shown that switches with input buffers experience a head-of-line (HOL) blocking, which decreases the throughput of the switch [12]. This problem can be overcome by using virtual output queues (VOQs) [13]. 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 (referred to as VQs & EQ lists in Fig. 1) stores the addresses of cells in different VOQs, and addresses of empty locations. Each location in the linked list memory contains the address of the next location in the empty queue linked list (EQL) or the virtual queue linked list (VQL) to which the location belongs. The linked list memory's size is defined by the number of cells that can be stored in the data memory of the switch. 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 have implemented the linked list memory in the M4K memory blocks of the Altera Cyclone FPGA. The memory exchanges control and data signals with the queue manager component (implemented on the same FPGA). M4K is set to work as a simple dual port memory, which can perform simultaneous read and write operations.

B. Queue Manager

Queue manager performs operations when the cell arrives to the queue manager, when the cell is scheduled by the output selector, and when the cell departs the switch. The queue manager stores pointers to VQLs. There are three pointers to each VQL, i.e. 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. These pointers are updated each time one of the operations is performed. Queue manager needs to carry out multiple operations 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 the cell bound for that output arrives to the switch. The cell is stored to the

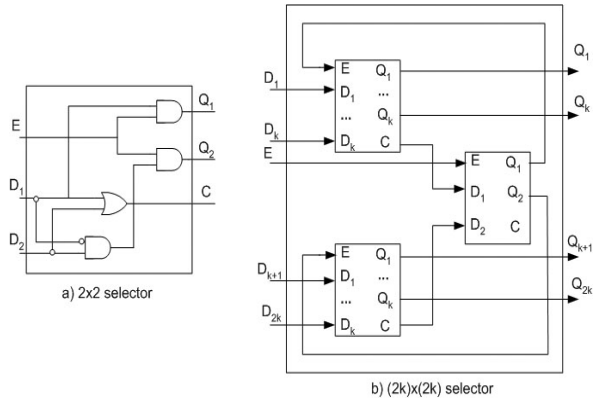

Fig. 2. The output selector structure

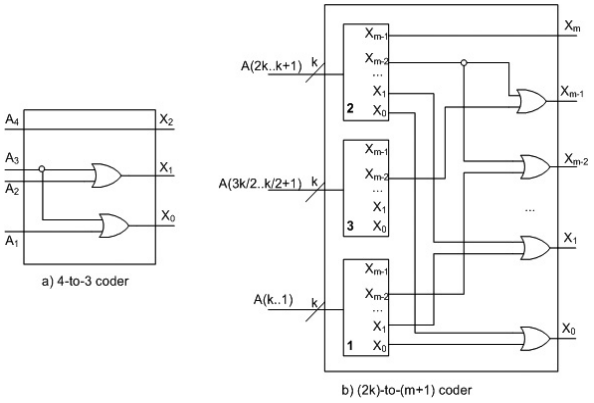

Fig. 3. The coder structure

address from this memory location. Similarly, when the cell departs from the switch, the memory location at the beginning of the corresponding VQL is added to the end of EQL. When a cell is scheduled, the corresponding VQL is updated. Queue manager obtains the scheduled cell from the output selector. If the pointer to the first unscheduled cell from the same VQL points to the last cell in the 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 implemented in VHDL, and can be scaled easily for different switch sizes.

C. Output Selector

The output selector schedules a cell of the associated input module by selecting the first available output for which the given input has a cell to send from the set of requested outputs. This way, the scheduling procedure is performed quickly but not fair, because the earlier output ports are chosen first. The fairness can be provided by rotating priorities in each frame.

The structure of the output selector is shown in Fig. 2. D bits contain the information about cells of a particular input that participate in the selection process. D_j bit is set to '1' only if the input has unscheduled cells for the j-th output port and the j-th output port was not selected by previous input ports. The number of D bits corresponds to the number of output ports. As a result of the scheduling process, Q bits contain the information about the scheduled output port. Only one Q bit can be set to '1' in one time slot, i.e. Q_j is set to '1' if 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 E bit is the enable signal for the output selector, and its outputs are valid only when the E bit is set to '1'. The C bit is set to '1' only if there is at least one of the D bits set to '1'. Otherwise, if there are no cells that are to be scheduled, this bit equals '0'. The E and C bits enable building of larger structures.

The output selector is implemented recursively. The basic output selector structure for the switch with two ports is shown in Fig. 2a. In the 2×2 output selector, $Q_1 = 1$ if $D_1 = 1$, and $Q_2 = 1$ if $D_1 = 0$ and $D_2 = 1$. The output selector for the switch with $2k$ output ports can be constructed recursively using the basic structure, as shown in Fig. 2b. The output selector structure for $2k$ output ports consists of two output selectors for k ports, and one basic output selector for two ports. Output selectors for $(2k) \times (2k)$ switches are built by dividing a set of $2k$ D bits into two subsets of k bits, and each subset is forwarded to a smaller output selector for k output ports. If any cell is scheduled in one of the subsets, then the C bit in the corresponding structure is set to '1'. The third 2-port output selector determines which k-port structure will be selected and sets the corresponding Q bit to '1', which serves as an enable signal for the selected k-port output selector.

The output Q bits of the output selector are forwarded to the coder, which structure is given in Fig. 3. The coder determines the position of a logical one in the vector with one-hot coding, and thus the number of the VOQ which has been scheduled for the observed time slot. When $Q_j = 1$, the coder returns j coded binary. It forwards the scheduled VOQ number to the queue manager. The coder is also implemented recursively. The simple structure of a 4-to-3 coder is given in Fig. 3a. The output X bits represent the VOQ number, where $X = 001$ for $A_1 = 1$, $X = 010$ for $A_2 = 1$, $X = 011$ for $A_3 = 1$, and $X = 100$ for $A_4 = 1$. The larger structure of a $2k$-to-$(m+1)$ coder can be obtained recursively (where $m = log_2 k + 1$) by using three k-to-m coders, as shown in Fig. 3b. First k bits are connected to the first coder, and second k bits are connected to the second coder. If the selected output is among the first k outputs the resulting coder has the same outputs as the first coder. Otherwise, if the selected output is among the second k outputs, the resulting coder has the same outputs as the second coder plus 2^m. So, the first $m-1$ bits of the resulting coder output X are equal to either outputs of the first or the second coder, when the selected output is among first or second k outputs. The third coder determines if the selected output is among the first $k-1$ outputs or second $k+1$ outputs, and sets X_m to 0 in the first case, or to 1 in the second case. Finally, the last bit X_m equals the last output of the second coder.

Both the output selector and the coder use the minimum number of logic circuits and produce the minimum delay. The correct functioning of both the output selector and the coder

can be verified by mathematical induction.

The output selector for the input i in time slot k calculates the schedule and reserves the output for the time slot $k+N+3-i$ (the first input port for the time slot $k+N+2$, the second for the time slot $k+N+1$, etc.) 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 uses M4K memory blocks functioning as an embedded shift register. The additional delay of three time slots is introduced (since the minimum tap distance of the shift register is three), but the implementation is more efficient and enables lower minimum time slot values, because the output memory reserves no LEs, whose increase in number decreases the speed of the switch.

III. Implementation Options

We implemented two different design options of the scheduler that implements the SGS algorithm, and examined their performance. In any case, the queue manager module must perform multiple operations within one time slot. The number of cycles in which these operations are performed is at least three, because three read and write operations need to be performed on the linked list memory. Pointers to beginnings, unscheduled cells and endings of VQLs may be stored either in registers (LEs) or memory blocks of the Altera FPGA. Finally, the speed at which input data are read to the input registers and output data are read from the output registers may be increased in order to increase the chip throughput and consequently the number of input modules on a single FPGA device.

Both implementation options have been verified by simulation, which includes accurate routing delays between LEs, pins and M4K blocks.

A. Case A

The state machine for the queue manager operation comprises four states: swait, write, schedule, and read. The state transitions are unconditional. The queue manager performs each of the operations in its corresponding state. In one additional state (swait) no pointer updates are performed. However, this additional time slot is needed to ensure the correct functioning of the controller, since the M4K memory inherently inserts one additional cycle between the address setup cycle, and the cycle in which the data is read. Therefore, the number of cycles per time slot is four.

The pointers to VQLs are implemented using LEs of the Altera Cyclone. For smaller designs, accessing pointers implemented in registers may require the shorter time than accessing pointers in memory. However, the number of LEs of the FPGA may limit the number of input modules implemented on a single chip.

B. Case B

The state machine for the queue manager operation 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. Also, the timing of schedule and read operations is defined by the output selector and the output memory. The output selector is allowed two cycles to choose an output, because the vector containing the scheduled cell's VOQ number and the vector with the requested outputs are updated in the same cycle.

To increase the scalability of the queue manager, the pointers have been implemented in the M4K blocks of the Cyclone FPGA. Since the M4K memory read operation requires one additional cycle between the address setup cycle and the cycle in which the memory outputs are read, we set the pointer memory to work with twice the speed of the linked list memory. Separate state machine with six states regulates the pointer operations. This way, the timing of pointer updates does not affect the timing of the linked list updates. Pointer updates start at the predefined times, and consist of writing to and reading from pointer memories. For example, in the first cycle the address of the pointer to the beginning of EQL is written to the pointer to the end of the VQL of an arriving cell, in the second cycle the same address is written to the pointer to the beginning of the VQL of the arriving cell if the VQL was empty before the arrival, etc.

Finally, to increase the number of modules which can fit the FPGA, the pins' speed is increased two times. Input control bits (which denote available outputs) are read from pins to the input registers at the speed which is two times higher than the speed of the case A. Namely, the first half of the input bits are read at falling and the second part at the rising edge of the clock with the time slot period. The first half of the output control bits (which denote remaining available outputs) is forwarded as soon as they are calculated, and the second half is read at the falling edge of this clock. The data memory addresses to be written to and read from, which are exchanged between network processor and the implemented controller, are similarly sped up. This way, the number of pins required to ensure the given throughput of control information is decreased around two times.

IV. The Performance Analysis

In this section we will discuss the performance of two implementation options presented in the previous section. The performance will be measured in terms of the design scalability, i.e. the number of input modules that can fit one FPGA device. The maximum number of input modules that can fit one device may be limited by different factors: the number of available LEs, the number of memory blocks, and the number of pins. Second important performance measure that will be examined is the speed of the design, i.e. the minimum achievable output selection time, T_s.

A. Case A

In the presented design, the queue manager performs operations in four cycles of a time slot and the pointers to VQLs are implemented using the LEs.

Tables I-III present the FPGA resource utilization and timing characteristics for different number of input ports N, and for different number of input modules N_P which could have

been fit onto a single device. The minimum output selection time includes the time needed for processing the pointers in the queue manager, and can be calculated from the maximum clock frequency as $T_{MIN} = 4/f_{MAX}$. The tables also show in the column "lim" which factor limits the number of input modules that fit one FPGA device: the number of LEs, the number of memory blocks, or the pin number. The number of pins used by the design can be calculated as:

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

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 reset signal. The number of input modules is limited by memory to 32 since there are 64 M4K blocks, and every input 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 link list memory and the output memory can be calculated as:

$$mem(N, F) = \lceil F\lceil log_2 F \rceil / 4096 \rceil + 1 \quad (2)$$

since the link list memory has $F log_2 F$ locations and the output memory in none of the observed cases reserves more than one M4K block. The linked list memory reserves $\lceil F\lceil log_2 F \rceil / 4096 \rceil$ M4K blocks, since the Quartus software for Altera FPGA devices limits the use of M4K block to 4096 bits.

TABLE I

RESOURCE UTILIZATION AND TIMING CHARACTERISTICS, $F = N$

N	N_P	LE	LM [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_{MIN} [ns]
16	16	12901	3.3	243	/	106	37.7
32	10	17354	5.5	247	LE	96	42
64	5	17255	6.5	236	LE	83	48.4
128	1	7798	3	283	Pins	72	55.6

Table I contains the FPGA resource utilization and timing characteristics when the frame length is minimal. It can be seen from Table I that the speed of the FPGA design decreases as the number of ports increases (e.g. the minimum time slot rises from 37.7 ns for $N = 16$ and $N_P = 16$, to 55.6 ns for $N = 128$ and $N_P = 1$). For $N = 128$, the maximum clock frequency is limited to 72 MHz. This clock frequency corresponds to the faster clock that feeds the queue manager. Therefore, the minimum output selection time is limited to 55.6 ns.

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 II and III contain the FPGA resource utilization, timing characteristic and limiting factors when the frame size is eight and sixteen times greater than the number of ports, respectively. It can be observed from Table II and Table III that the maximum clock frequency of the design somewhat decreases with the increase of the frame length because it causes the increase of the number of required LEs. However, the minimum output selection time does not increase significantly with the frame length F, as expected, and will remain well below 100ns for the finest granularities as desired.

TABLE II

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

N	N_P	LE	LM [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_{MIN} [ns]
16	13	13065	12.3	282	Pins	103	38.9
32	8	16092	17.8	243	LE	91	43.8
64	4	15809	20	231	LE	79	50.5
128	1	9698	11.2	287	Pins	70	56.9

TABLE III

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

N	N_P	LE	LM [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_{MIN} [ns]
16	12	14609	25.2	287	Pins	104	38.6
32	7	16280	33.5	235	LE	91	43.9
64	3	14421	32	212	LE	81.3	49.2
128	1	9733	23.5	289	Pins	68	58.8

B. Case B

In the presented design, the queue manager performs operations in six cycles of a time slot, the pointers to VQLs are placed in the FPGA memory, and the pins are two times faster than in case A.

In this case, the minimum output selection time can be calculated from the maximum clock frequency as $T_{MIN} = 6/f_{MAX}$, since the time slot comprises six cycles of the faster clock that manages pointer updates. The upper limit of the number of modules which can fit the FPGA may be limited by the number of pins, the number of M4K blocks and the number of LEs on the FPGA. When the pins are sped up the number of pins can be calculated as:

$$pins(N, F) = (2\lceil \lceil log_2 F \rceil / 2 \rceil + log_2 N + 1)N_P + N + 4, \quad (3)$$

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 clock is used.

The number of input modules is limited by memory to 12 since there are 64 M4K blocks, and every input 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, as described by equation:

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

because in the cases observed no pointer memories nor output memories reserve more than one M4K block.

The resource utilization, timing characteristics and limiting factors for the minimal frame length are given in Table IV, and for $F = 8N$ and $F = 16N$, in tables V and VI respectively. It can be observed from the tables IV-VI that the minimum time slot duration slowly increases with the switch size, and

with the frame length as before. For $N = 16$ and $F = 16N$, the maximum number of modules which can fit the FPGA is $N_P = 12$, and this number decreases to $N_P = 6$ for $N = 128$ and $F = 16N$. Speed of the pins that carry control data has been increased twice (for all the cases but the $N = 16$), to increase the number of modules placed on the chip. For $N = 16$, the increase in speed of the pins was not necessary since the upper limit was defined by the number of available memory blocks. The minimum time slot is slightly longer and the number of modules which can fit the FPGA in this case is smaller than in the case A for smaller switch sizes. However, as the size of the controlled packet switch increases, the minimum time slot duration becomes shorter, and the number of input modules that can fit the FPGA in case B is significantly larger than in case A. That is, in case B nine modules can fit one chip for $F = 16N$ and $N = 64$, or six modules for $N = 128$; while in case A three modules fit the chip for $N = 64$ or one module for $N = 128$. The minimum time slot is limited to T_s=56.6ns in case B, which is still well below 100ns.

TABLE IV

RESOURCE UTILIZATION AND TIMING CHARACTERISTICS, $F = N$

N	N_P	LE	LM [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_{MIN} [ns]
16	12	4643	7.6	216	Mem	130	46.1
32	12	6829	18.4	180	Mem	124	48.1
64	12	10489	42.9	248	Mem	110	54.4
128	10	14421	81.9	292	Pins	107	55.7

TABLE V

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

N	N_P	LE	LM [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_{MIN} [ns]
16	12	5211	19.4	264	Mem	125	48
32	12	7533	44.9	204	Mem	117	51.1
64	12	11172	101.6	272	Mem	109	55
128	9	13821	170	294	Mem	106	56.6

TABLE VI

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

N	N_P	LE	LM [Kbits]	$pins$	lim	f_{MAX} [MHz]	T_{MIN} [ns]
16	12	5635	34.4	288	Mem	124	48.3
32	12	7912	77.9	228	Mem	115	52.2
64	9	8699	130.4	221	Mem	112	53.5
128	6	9375	191.9	252	Mem	109	54.9

In Table VII we calculate the maximum number of input modules that would fit one FPGA device, using formulas (1), (3), and (4). 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".

V. CONCLUSION

In this paper we proposed the design of the scheduler based on the SGS algorithm, and presented its implementation. It was shown that placing pointers to virtual queue lists into the FPGA memory and speeding the pins significantly improves the design scalability, and the processing speeds in the larger switches. The preferred design option is highly scalable: up to 10 input control modules of an 128×128 switch 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. In the case of ports' bit-rate of 10Gb/s, the total capacity of the controlled switch would be 128·10Gb/s=1.28Tb/s. The output selection time remains below 60 ns in discussed high-capacity packet switches. Consequently, the implemented scheduler provides delay guarantees for sensitive applications, and the fine granularity of reservations.

TABLE VII

MAXIMUM NUMBER OF INPUT MODULES N_P PER CHIP

	$F = N$			$F = 8N$			$F = 16N$		
N	ps	nops	mem	ps	nops	mem	ps	nops	mem
16	16	16	12	16	13	12	16	12	12
32	22	12	12	18	10	12	16	9	12
64	15	8	12	13	6	12	13	6	9
128	10	1	12	9	1	9	8	1	7

Acknowledgments We thank Miloš Blagojević for his help with this paper.

REFERENCES

[1] A. Smiljanić, "Flexible bandwidth allocation in terabit packet switches," *Proceedings of IEEE Conference on High Performance Switching and Routing (Best Paper Award)*, June 2000, pp. 233-241.

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

[3] E. Oki, R. Rojas-Cessa, H. J. Chao, "A pipeline-based approach for maximal-sized matching scheduling in input-buffered switches," *IEEE Communication Letters*, vol. 5, no. 6, June 2001, pp. 263-265.

[4] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transcations on Networking*, vol. 7, April 1999, pp. 188-200.

[5] Y. Tamir, and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, 1993, pp. 13-27.

[6] A. Smiljanić, "Flexible bandwidth allocation in high-capacity packet switches,"" *IEEE/ACM Transactions on Networking*, April 2002, pp. 287-293.

[7] A. Smiljanić, "Bandwidth Reservations by Maximal Matching Algorithms," *IEEE Communication Letters*, March 2004, pp. 177-179.

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

[9] E. Leonardi, M. Mellia, A. Marsan, and F. Neri, "Stability of maximal matching scheduling in input-queued cell switches," *Proceedings of IEEE International Conference on Communication 2000*, pp. 1758-1763.

[10] A. Smiljanić, "Scheduling of multicast traffic in high-capacity packet switches," *IEEE Communication Magazine (Best Paper Award in IEICE/IEEE HPSR 2002)*, November 2002, pp. 72-77.

[11] P. Leventis, et al., "Cyclone^{TM}: A low-cost, high-performance FPGA," *Proceedings of the IEEE CICC 2003*, September 2003, pp. 49-52.

[12] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space division switch," *IEEE Trans. Communications*, 35(12)(1987), pp. 1347-1356.

[13] Y. Tamir, G. Frazier, "High Performance multi-queue buffers for VLSI communication switches," *Proc. of 15th Ann. Symp. on Comp. Arch.*, June 1988, pp. 343-354.