Design of the Multicast Controller for the High-Capacity Internet Router

Miloš Blagojević[1], Aleksandra Smiljanić[1,2], Miloš Petrović[1]
[1]*Belgrade University, Belgrade, Serbia*
[2]*Stony Brook University, New York, USA*
milosdb@etf.bg.ac.yu, aleks@ieee.org, pmilos@ieee.org

***Abstract*—High-capacity routers that circulate multicast packets have unique properties. Their architecture is non-blocking even for the multicast traffic. It also provides delay guarantees. This paper presents the detailed implementation of the controller that circulates multicast packets through a router toward their destination ports. The controller was implemented on the FPGA device and its functioning was tested. Its scalability and speed were assessed, and shown to be satisfactory.**

I. Introduction

Multicasting has been recognized as a problem in high-capacity Internet routers [3], [6], [7], [13], [14], [15]. In this paper we consider multicast traffic that has one source and multiple destinations. A single-hop high-capacity router is typically based on input buffers and a cross-bar [4], [8], [12]. Packets are split into cells. In each time slot, the cross-bar is configured so that each input is connected to at most one output, and each output is connected to at most one input. Alternatively, the cross-bar might have a multicast capability meaning that one input might simultaneously send packets to multiple outputs. In both cases, scheduling of multicast packets is problematic. In the case of purely unicast cross-bar, an input has to replicate each multicast packet, and send it separately to all output ports. If the input sources many popular sessions, it might become clogged easily. This is because the input might have to send packets of a popular session to as many as N ports. In the case of the multicast cross-bar, a dilemma arises. Should a multicast packet be sent to the outputs when all of them are free for reception, or should it be sent only to the available outputs? In the first case, the packet might wait unacceptably long; in the second case (that allows so called fan-out splitting), the packet might again be replicated and sent to all outputs separately. In [6], a discouraging result was reported. Even if the optimal scheduling algorithm is applied to the router based on the multicast cross-bar, the speedup required to pass an admissible traffic increases with the number of router ports. An admissible traffic is the one that does not overload output router ports.

In [9], the multicast solution implemented in the Cisco Gigabit Switched Router (GSR) was also presented. In GSR, there are two types of queues: VOQs for storing unicast cells, and a separate multicast queue for multicast cells. The cross-bar has multicast capability, so each multicast cell may be sent to multiple outputs within one time slot. Cells are served according to the eSLIP algorithm. Multicast cells have priority. In some time slot, a multicast cell is transmitted only to the outputs which it won in a contest. Generally, a multicast cell will be sent to all outputs within multiple time slots. There are many concerns about this approach. Since multicast cells are stored in FIFO queues, well known head-of-line blocking might occur. Also, in the worst case, one cell could be sent to only one output in each time slot, and again the transmitting port would become clogged. The properties of this approach are not analitically proven, and therefore its performance for general traffic pattern is uncertain.

On the other hand, the high-capacity router could handle the multicast traffic well when the packet circulation through a unicast fabric is utilized [13], [14], [15]. Let us say that all ports receiving the packets of a multicast session belong to the multicast set corresponding to that session. Let us assume that the input port sends the packets only to a limited number of ports in the multicast set. Each of those ports forwards the packets to a limited number of ports in the multicast set that did not receive them yet, and so on. This procedure continues until all the outputs in the multicast set receive the packets in question. In this way, the transmission load of the input port is spread over all the ports in the multicast set. The speedup required to pass an admissible traffic depends on the number of ports to which each port forwards multicast packets that it receives, i.e. P. It was proven that all the admissible multicast traffic patterns may pass the cross-bar with the speedup of $P + 2$ if the maximal matching algorithm is used for packet scheduling [13], [14]. Here, the maximal matching algorithm is the one that does not leave an input-output pair unmatched if there is a packet from the input to the output in question [1]. One scalable maximal matching algorithm that could be used is the sequential greedy scheduling (SGS) [12]. Not only that the throughput could be guaranteed through this architecture, but also the packet delay. Let us assume that all sessions passing the router police their traffic, sending less than the negotiated amount of data within each policing interval of duration D. Then, packets of each session are guaranteed the delay below $D \cdot \lceil \log_P N \rceil$, where N is the number of ports and $P \geq 2$ [13], [14]. This delay is acceptable for $P \geq 2$, since the packet does not pass through a large number of core routers.

A high-capacity router, which circulates multicast packets as described, is non-blocking even for the multicast traffic. Regardless of the traffic pattern, and popularity of various

multicast sessions, all the traffic will pass the router as long as the outputs are not overloaded. In addition, the delay is guaranteed to the sessions through the router. In this paper, we present the implementation of the router controller that performs the multicast packet circulation. This controller builds and maintains the router internal trees according to which the packets will be forwarded. In Section 2, we describe the implemented protocol which builds and maintains trees. Section 3 presents the testing environment in which the functioning of the implemented controller has been verified. In Section 4, the performance of the FPGA design is analyzed in terms of its speed and scalability. Resources that the design consumes have been calculated in this section, and the design parameters have been determined based on the calculated formulas. Finally, timing analysis has been performed for the chosen parameters.

II. The Multicast Controller Implementation

In this section, we will present implementation of the multicast control protocol for high capacity routers [13]. Multicast packets are forwarded through the router according to the tree comprising the ports that receive packets of the multicast session in question. Implemented protocol maintains the tree structure, adding new ports to the tree or removing existing ports from the tree. In response to changes in the multicast group membership, the ports exchange control messages and update the forwarding tree accordingly. The tree is updated so that it is always the shallowest, thus providing the lowest possible delay. This is achieved by adding a new port at the end of the tree shortest path, and replacing a port on leave with the port at the end of the tree's longest path.

In our scheme, a tree is associated with each multicast session. This tree comprises nodes that correspond to the ports of the multicast set. Those ports receive the packets of the session in question. Multicast traffic arrives at the root port and it is further circulated through the router according to corresponding multicast tree. Each port in the tree sends the multicast packet to its children ports in the tree. As it was previously shown [13], the forwarding fanout of $P = 2$ provides satisfactory delay through the router. This fanout value was used in our implementation, meaning that each port forwards the multicast packets to at most two ports in the multicast set. For each multicast session passing the router, the higher-layer protocols determine the multicast set of ports, and the root port. The internal router controller that we are implementing receives the requestsfor adding and removing ports of the multicast set from the higher-layer protocols such as SM-PIM. Based on these requests, our protocol generates the internal control messages that are exchanged between the router ports in order to update the multicast forwarding tree.

Multicast sessions are defined by their IP multicast addresses. Each multicast session has the local session identifier - SID. The SID is associated with the port, and it is actually the memory address of the location where the multicast session information is placed. This tree memory entry stores the information about the parent (previous) node in the tree, the children (next) nodes in the tree, and the branch fanouts. The branch fanout represents the number of nodes that could be reached through that branch. There is no need to perform the lookup based on the long IP multicast address at each node during the message journey through the tree. Instead, each node keeps the information about the SIDs of its parent and children nodes in the tree memory, and replace its own SID in the control message destination field with the SID of the next port to which the message is forwarded. In Figure 1, the format of the tree memory entry is shown, where F_right and F_left denote fanout values and ID is port identification number(from 1 to N). The tree memory of one port comprises one entry for each multicast session whose packets it forwards.

F_right	F_left	Parent SID | ID	Child left SID | ID	Child right SID | ID

Fig. 1. Tree memory entry

To keep the tree memories updated when the corresponding multicast set changes, the ports exchange internal control messages through the cross-bar. There are eight different types of internal control messages:

1. Allocate memory for a new port
2. Find where to add the new port and add it
3. Find a replacement port for the port on leave
4. Send the multicast entry to the replacement port and release memory
5. Change child of the replacement port
6. Change parent
7. Change child of the parent of the leaving port
8. Forward data packet to the children ports

Processing of data packets is beyond the scope of this paper, and so handling of the messages of type 8 was not implemented. All messages have the same length and the similar structure. As it can be seen from Figure 2, the internal control message consists of a field denoting the message type, two fields related to the destination port, denoting its SID and ID, two fields related to the port which processes the message, also denoting its SID and ID, and one additional field with the information that needs to be exchanged when a port is leaving the tree.

Type	Destination Port SID	Des. port ID	Port SID	Port ID	Remove message info

Fig. 2. Local control message format

When the request for adding a port to the given multicast set arrives to the router, the higher layer-control protocol processes it, generates the local control message of type 1 and sends that message to the port which should be added. This message contains information about the router session ID, RSID=[root port ID, root port SID], for the corresponding multicast session, so that the new port can find the appropriate tree based on this information. The destination port SID is omitted from this message type, because that information is not available until the tree memory entry is allocated for this

multicast session at this port. The next available memory location is assigned to the new tree entry. The address of the assigned memory location represents the local multicast session identifier (SID) for that port. The next step is to find a place in the corresponding tree where the new port will be added. The port to be added generates the message of type 2, and sends it to the root port. From there, the message is forwarded through the tree, choosing a branch with the smaller branch fanout number at each port. If fanouts have the same value, the message will be forwarded through the left child port. Fanouts of the branches on the control message path are incremented by 1. When a port with the branch fanout equal to zero (the port without at least one child), is reached, the search is over and the new port is added as a child of this port.

Removing the port from specific multicast set involves more intensive message exchange. As before, a request for removing certain port will be first processed by the higher-layer control protocol, the message of type 3 will be generated and sent to the root port. First, a replacement port needs to be found. This port will replace within the tree the port that has to be removed. In order to find the replacement node, the control message is forwarded through the tree to the most distant port from the root. Forwarding rule is the opposite from before, this time a port will always forward the message through a branch with the higher fanout value. If the fanouts have the same values, the right child will be selected. Fanouts of the branches on the control message path are decremented by 1. Forwarding is stopped when the childless port is reached. That port is the replacement port and it needs to update its local tree memory entry, so that it reflects its new position in the tree (position of the leaving port). So, the replacement port generates the message of type 4, and requests the corresponding tree memory entry from the leaving port. The replacement port does not know the multicast SID value of the port on leave. For this reason, RSID is included in message of type 4. So, when the leaving port receives this type of message, it can perform the lookup and find its SID corresponding to the given RSID. Based on the obtained SID, the requested tree memory entry is found and sent to the replacement port. Three messages are used for sending the requested data. The first two messages are of type 5 (one for left and other for right child) which carries the child information (ID, SID and the branch fanout), while the third one is the message of type 6 which carries the parent information (ID and SID). When all requested data is sent, the leaving node should release this multicast tree memory entry. After receiving all requested information (two type 5 and one type 6 messages), the replacement port modifies the corresponding tree memory entry. In the last step, the replacement port announces itself to the new neighbours (its new parent and children ports). So, the replacement port send to its new children ports the messages of type 6 with the request to change the parent, and to its new parent port the message of type 7, with the request to replace the child port specified in the message. When these messages are received, the neighbouring ports update their tree memory entries with the new parent, and the child port, respectively.

III. Testing of the Implemented Multicast Controller

The functional verification requires a complete controller environment including the cross-bar and the cross-bar scheduler, as shown in Figure 3. The functioning of the higher-layer control protocols was emulated by the central processor (CP). Implemented multicast control module has to have two FIFO buffers, one at the input and other at the output, so it can handle situations when multiple control messages arrive, or have to be sent from controller module, at same time slot. Complete testing environment and multicast control modules are implemented together on the Altera FPGA. Testing is performed through timing simulations using Quartus II software.

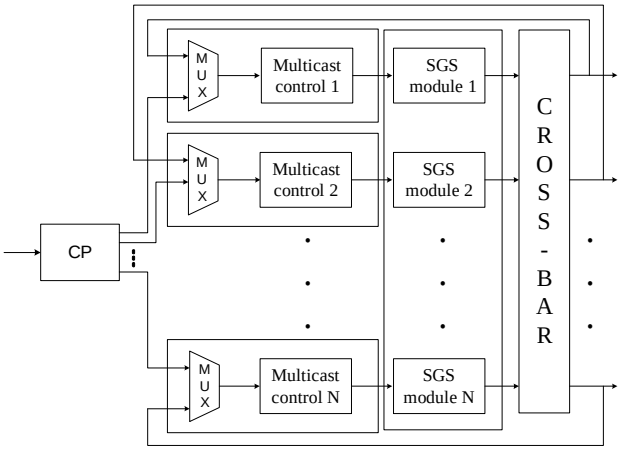

Fig. 3. Block scheme of complete test design

The multicast controller of the 8x8 router was implemented on a single FPGA chip (Figure 3). Eight multicast control modules are connected to the scheduler modules and the cross-bar. The multicast control module receives the control messages, processes them, updates the tree memory accordingly and generates new control messages that it sends to other control modules. The scheduler determines the cross-bar configuration in each time slot based on the information about the outstanding control messages and their destinations. The scheduler configures the cross-bar according to the sequential greedy scheduling (SGS) algorithm [10], [11], [12]. In SGS, inputs sequentially one after another choose outputs to which they will be connected. SGS is a scalable maximal matching algorithm. Internal control packets generated by the multicast control modules are stored in the memory of the scheduler modules. Then, these control packets are scheduled for transmission through the cross-bar according to the SGS algorithm.

Requests for adding or removing of certain ports came to the central processor (CP) which transforms those requests to the proper internal control message formats. Regardless

of the request type, RSID has to be obtained from CP for further request processing. RSID is used because it uniquely defines each multicast session that passes the router, but it is shorter than the IP multicast address. Consequently, using RSIDs instead of multicast IP addresses within the router reduces the amount of the exchanged control information, and the size of the port lookup tables that find SIDs for the given multicast sessions.

Each multicast control module (Figure 4) can get requests from two sources: from the higher-layer control protocols (i.e. the CP block) and from other multicast control modules through the cross-bar. This means that one module can receive two control messages in the same time slot. These requests are stored in a FIFO buffer and handed to the controller module through a multiplexer. Also, a FIFO buffer is necessary at the controller output because each module can generate up to three messages in one time slot. Beside the input and the output FIFO memories, the multicast control module has two additional memories: the tree memory and the lookup memory (which translates the RSID address to the local SID, as already described).

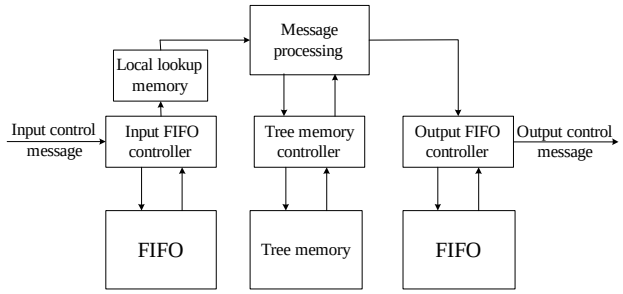

Fig. 4. The multicast control module (with implemented memories)

Described design was implemented on the Altera FPGA. The software for testing the given design was developed similarly as in [11]. Quartus II software uses the waveform file (with the extension .wvf) as the input file for the simulation tool. All information about input signals is stored in this file using simple description programming language. Our software generates random sequence of request messages, and translates that sequence to the waveform file data. Created waveform file is used as the input stimuli vector. In the presented implementation, information about created tree structures is not available at the output pins, but it is contained in the implemented memory. Information about the memory content is placed in the memory information file (with the extension .mif). Assuming generated requests (to add or remove certain ports), the expected simulation results were found. After the simulation is completed, the data stored in the .mif file is compared with the expected simulation outcome in order to verify correct functioning of implemented design.

The testing environment including the multicast controller for the 8x8 router shown in Figure 3 was implemented on the Cyclone II EP2C70F896C6 device. In the tested scenarios, the number of multicast sessions (trees) that can pass the router was assumed to be up to $N_s = 40$. Probability that some request will arrive at certain time instance was assumed to be $p_r = 0.5$. The type of the request (add/remove), the port ID in the request and its multicast session are chosen according uniform distributions (events have equal probabilities of 0.5 and $1/N$ and $1/N_s$ respectively). Around 50 simulations were performed. Duration of each simulation was 350 time slots. Different random sequences of requests were generated and correct functioning of the implemented design was verified for each of them.

IV. Performance Analysis of the Multicast Controller Design

Performance of the implemented multicast control module can be measured by its speed (processing time) and scalability. Particular performance measures that will be used are: the minimum time slot duration, the maximal number of input modules that can fit one device and the maximal number of multicast session per port. The required chip resources are calculated in terms of the design parameters. From those formulas and the chip available resources, the feasible design parameters are found. Limiting resources might be either the number of available logic elements (LEs), the number of pins, or the number of available memory blocks (M4K) on the Altera FPGA chip [2], [5]. New testing environment has been created for the chosen parameters, and the timing simulation has been performed.

After we verified that our design works properly, the next step is to perform more detailed timing analysis and estimate the processing speed for the multicast controllers of the routers of various sizes. Timing analysis was performed for different number of test modules implemented on the selected FPGA device. Each test module consists of the multicast control module and adjoining scheduler module. In the timing analysis, the cross-bar was removed from the design, because it is typically on a separate chip in the high-capacity routers. Also, the local lookup memories of the multicast control modules were reduced, because they should be implemented on the external memories. Because, the multicast control module and the SGS control module are associated with the router port, it is natural to join them into a single port control module. So, the analyzed design comprises both multicast and SGS control modules. Assuming different numbers of ports, N, and numbers of test modules per chip, N_p, the maximal number of the multicast sessions per port, N_s, was calculated and the appropriate timing analysis was performed.

The number of the available pins possibly limits the number of test modules that can fit one chip. The number of necessary pins depends on the number of ports, the number of modules per chip and the message length. At the input side each test module has to reserve pins for the incoming control message (L bits wide), one request notification bit, and $(L+\log_2 N+1)$ pins for the CP block control input (comprising the control message and its destination port). At the output it needs pins for the scheduled control message (L bits wide) and the cross-bar configuration message ($\log_2 N$+1 wide). In addition, N

pins are required for the SGS control message [10], and 5 pins are required for clock, reset and initialization signals. Total number of necessary pins for testing design with N_p modules per chip, can be obtained as:

$$\begin{aligned} N_{pin}(N, N_p, N_s) &= (2N_p + 1) \cdot L + \\ &+ (N_p + 1) \cdot (\log_2 N + 1) + \\ &+ N + N_p + 5. \end{aligned} \quad (1)$$

From Figure 2 the message length can be calculated as:

$$L = \lceil \log_2 N_{type} \rceil + 2 \cdot \lceil \log_2 N_s \rceil + 3 \cdot (\lceil \log_2 N \rceil + 1), \quad (2)$$

where N_{type} denotes the number of different message types, in our case $N_{type} = 8$.

Another limiting factor might be the number of M4K blocks on Altera's FPGA required for the implementation of all memories. The multicast control module consists of two FIFO memories, simplified lookup memory and the tree memory, while the SGS module includes one linked list memory, three memories for pointers, one output memory and one memory for storing the queued control messages. Most of the M4K blocks are used for the tree memory and the memory with control messages, while all other implemented memories do not contribute significantly to the total number of used M4K blocks. The tree memory should be large enough to place one entry for each multicast session passing that port. Formula for the number of M4K blocks that the SGS module consumes was calculated in [10]. The number of blocks required for the memory with control messages should be added to this formula.

All memories are implemented using the Cyclone II embedded memory structure. That memory consists of columns of the M4K memory blocks that can be configured to provide various memory functions such as RAM, FIFO and ROM. We could not find in the Altera's documentation the formula for the number of M4K blocks required by the given design. Calculation of this formula was intricate. Based on various cases, we derived a general formula for the number of required M4K blocks in terms of the number of memory entries, N_a, and the single entry width, N_w. According to [2], the content of each M4K blocks can be arranged in N_xxN_y structure, where allowed configurations are: 4Kx1, 2Kx2, 1Kx4, 512x8, 256x16, 128x32 (with the total of 4096 bits), and 512x9, 256x18, and 128x36 (with the total of 4608 bits). Quartus fitter configures M4K blocks according to the requested memory dimensions. For storing N_a entries where each entry is N_w bits wide, the M4K configuration is determined by the requested address space, while the entry width, N_w, affects the number of used M4K block. So, if $N_a \leq 128$ then $N_x = 128$.Also, it should be noticed that for $128 \leq N_a \leq 512$ the M4K blocks of up to 4608 bits are used, while for $512 < N_a \leq 4096$ only the 4096 bits large M4K blocks are used. If $N_a > 4096$, the 4Kx1 M4K blocks are used, and some additional logic has to be implemented in order to provide the unique address space of the requested size. Based on these observations, the number of used M4K blocks (N_m) can be calculated as:

$$N_m(N_a, N_w) = \lceil N_w / K(N_a) \rceil, \quad (3)$$

where function $K(N_a)$ is given by:

$$K(N_a) = \begin{cases} 36, & N_a \leq 128 \\ \frac{4608}{2^{\lceil \log_2 N_a \rceil}}, & 128 < N_a \leq 512 \\ \frac{4096}{2^{\lceil \log_2 N_a \rceil}}, & 512 < N_a \leq 4096 \\ \lceil 4096/N_a \rceil, & 4096 < N_a \end{cases} \quad (4)$$

In order to obtain the total number of used M4K blocks, we will, first, calculate the number M4K blocks used for each implemented memory structure. Most of M4K blocks are used for the tree memory implementation. According to equation (3), the number of M4K blocks required by the tree memory is $M_{tree} = N_m(N_s, M_w)$, where M_w is the number of memory bits used for one entry in the tree memory shown in Figure 1. From this figure, M_w can be calculated as:

$$M_w = 3 \cdot \lceil \log_2 N \rceil + 3 \cdot \lceil \log_2 N_s \rceil + 2 \cdot \lceil \log_2 N_F \rceil, \quad (5)$$

where N_F is the largest branch fanout value, in our case $N_F = N/2$. $N_m(F, L)$ is the number of M4K memory blocks used for implementation of the memory with control messages. That memory should be large enough to store F control messages, where F represents the number of cells per policing interval [10], [12], [13]. In our implementation $F = 8N$. The number of the M4K blocks required for the SGS implementation was calculated in [10] to be:

$$mem(N, F) = \lceil F \lceil \log_2 F \rceil / 4096 \rceil + 4. \quad (6)$$

The lookup memory was implemented so that it can fit a single M4K block. Each of two FIFO memories can store up to 128 messages. The input FIFO memory stores only internal control messages (L bits wide), while the output FIFO stores internal control messages and destination IDs for the cross-bar configuration ($L + \log_2 N + 1$ bits wide). So, the numbers of used M4K blocks for the FIFO memories are $N_m(128, L)$ and $N_m(128, L + \log_2 N + 1)$, respectively. From previous , the total number of used M4K modules can be obtained as:

$$\begin{aligned} N_{M4K}(N, N_p, N_s) &= N_p \cdot [N_m(N_s, M_w) \\ &+ N_m(128, L + \log_2 N + 1) \\ &+ N_m(128, L) + N_m(F, L) \\ &+ mem(N, F) + 1]. \end{aligned} \quad (7)$$

Our design was implemented on the Cyclone II device which has 250 M4K memory blocks and 622 general-purpose I/O pins. For the specified switch dimension N and the number of modules per chip N_p, the maximal number of sessions passing one module N_{sm} should satisfy:

$$\begin{aligned} &N_{M4K}(N, N_p, N_s) \leq 250, \\ &N_{pins}(N, N_p, N_s) \leq 622, \\ &N_s = 2^k, k \in \mathbf{N} \end{aligned} \quad (8)$$

For specified N and N_p, and calculated maximal N_s, the design parameters were determined (all memory dimensions, input and output signal sizes) and the design was implemented. Results of the timing analysis for the implemented design are

presented in the table I. Column "lim" presents the limiting factors for the number of sessions per module. The number of pins or the number of memory blocks might be the limiting factor, depending on the inequality that is the first one reached in (8). The utilization of logical elements remains below 25% for all tested parameters. So, for the selected device, the number of logic elements is not a limiting factor. Low utilization of the chip logic contributes to the satisfactory design speed. Implemented multicast control module assumes

TABLE I
RESOURCE UTILIZATION AND TIMING CHARACTERISTICS

N	N_P	N_S	LM [M4K]	$pins$	LE [kbits]	lim	T_{MIN} [ns]
8	1	8192	126	133	2.0	M4K	54.34
	2	4096	128	202	3.4	M4K	50.98
	4	2048	144	334	6.6	M4K	54.15
	8	1024	168	584	12.6	Pins	52.39
16	1	8192	136	149	2.1	M4K	53.05
	2	4096	142	223	3.7	M4K	52.58
	4	2048	152	365	7.0	M4K	54.84
	8	512	120	601	12.7	Pins	53.57
32	1	8192	147	173	2.3	M4K	52.85
	2	4096	154	253	4.1	M4K	53.81
	4	2048	176	404	8.0	M4K	53.62
	8	128	96	591	13.4	Pins	55.76
64	1	8192	159	213	2.6	M4K	56.71
	2	4096	168	297	4.8	M4K	55.92
	4	2048	192	459	9.3	M4K	57.03
	8	16	104	565	14.4	Pins	55.76
128	1	8192	178	213	3.1	M4K	54.20
	2	4096	194	297	5.9	M4K	56.39
	4	2048	236	492	10.8	M4K	55.66

multiple memory access operations (read data from the tree memory, process those data, and based on the obtained result write new data to the tree memory). Design was implemented so that those operations are performed efficiently during six cycles of the cell time slot. Six clock cycles were suitable because the SGS module uses similar timing structure, where six memory operations are implemented in one timeslot, too. So, the minimal time slot duration is obtained from the maximal achieved clock frequency f_{MAX} as $T_S = 6/f_{MAX}$. Maximal value for T_S in performed simulations was 57.03ns as it can be seen from table I. Since the delay through the router is less than $D \log_2 N = FT_S \log_2 N = 8NT_S \log_2 N$, the proposed design provides a tolerable delay to the sensitive applications.

V. CONCLUSION

In this paper, we presented the design of a multicast controller for a high-capacity non-blocking router. Regardless of the multicast traffic pattern, all the traffic will pass the router as long as the outputs are not overloaded. Implemented controller builds and maintains internal trees within the router for each multicast session. Multicast packets of the sessions are forwarded according to the appropriate trees. Each tree is kept shallowest by the implemented protocol.

Implemented controller is distributed. Each port stores the limited local information about the tree corresponding to each multicast session whose packets this port receives. Each port has the multicast control module, which receives the control messages, updates its tree memory according to these messages, and generates the control messages for the next port down the tree in question. For testing purposes, a complete control path within the 8x8 router has been implemented including the multicast controller, scheduler, and the cross-bar through which the control messages are exchanged. Testing has given a satisfactory result: the multicast controller has been shown to correctly build and maintain the trees.

After the functional verification, the performance of the multicast controller has been examined. Control modules have been implemented on the Altera Cyclone II FPGA. It has been examined how many multicast sessions can be served by a module, if different numbers and sizes of modules are assumed. It was shown that whenever the number of control modules is four or less, the number of supported multicast sessions per module is more than 2000. We find such scalability satisfactory. Also, the processing time of the multicast controller is less than 57ns, and would incur the delay which is tolerable by the sensitive applications.

Acknowledgement: We thank to Vlada Petrović and Marija Antić for their help.

REFERENCES

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

[2] Altera Corporation, "Cyclone II Device Handbook Volume 1," *http://www.altera.com/*

[3] A. Bianco, P. Giaccone, C. Piglione, S. Sessa, "Practical algorithms for multicast support in input-queued switches," *Proceedings of the HPSR 2006*, Poznan, Poland, June 2006.

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

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

[6] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri, "Optimal multicast scheduling in input-queued switches," *Proceedings of the IEEE ICC 2001*, Helsinki, Finland, June 2001.

[7] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri, "Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput," *IEEE/ACM Transactions on Networking,* Vol.3, No.11, pp.465-477, June 2003.

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

[9] N. McKeown, "Fast Switched Backplane for a Gigabit Switched Router," *white paper, Cisco Systems,* San Jose, Calif., 1997.

[10] M. Petrović, A. Smiljanić, "Design of the Scheduler for the High-Capacity Non-Blocking Packet Switch," *Proceedings of the IEEE HPSR 2006*, Poznan, Poland, June 2006.

[11] M. Petrović, M. Blagojević, V. Joković, A. Smiljanić, "Design, implementation, and testing of the controller for the terabit packet switch," *Proceedings of the IEEE ICCCAS 2006*, Guilin, PR China, June 2006.

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

[13] 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.

[14] A. Smiljanić, "Flexible multicasting in high-capacity packet switches," *IEEE Communication Letters,* August 2002, pp. 349-351.

[15] J. S. Turner, "An optimal nonblocking multicast virtual circuit switch," *Proceedings of INFOCOM 1994,* vol. 1, pp. 298-305.