# Hardware implementation of a scheduler for high performance switches with Quality of Service support

**Article** *in* Proceedings of SPIE - The International Society for Optical Engineering · May 2009

DOI: 10.1117/12.821522

CITATIONS
0

READS
52

**4 authors**, including:

Félix B. Tobajas
Universidad de Las Palmas de Gran Canaria
**81** PUBLICATIONS **212** CITATIONS
SEE PROFILE

Valentín de Armas Sosa
Universidad de Las Palmas de Gran Canaria
**63** PUBLICATIONS **151** CITATIONS
SEE PROFILE

Roberto Sarmiento
Universidad de Las Palmas de Gran Canaria
**202** PUBLICATIONS **1,075** CITATIONS
SEE PROFILE

# Hardware implementation of a scheduler for high performance switches with Quality of Service support

R. Arteaga, F. Tobajas, V. De Armas and R. Sarmiento

Institute for Applied Microelectronics (IUMA)
Departamento de Ingeniería Electrónica y Automática (DIEA)
University of Las Palmas de Gran Canaria, Campus de Tafira, 35017, Las Palmas de Gran Canaria.

## ABSTRACT

In this paper, the hardware implementation of a scheduler with QoS support is presented. The starting point is a Differentiated Service (DiffServ) network model. Each switch of this network classifies the packets in flows which are assigned to traffic classes depending of its requirements with an independent queue being available for each traffic class. Finally, the scheduler chooses the right queue in order to provide Quality of Service support. This scheduler considers the bandwidth distribution, introducing the time frame concept, and the packet delay, assigning a priority to each traffic class. The architecture of this algorithm is also presented in this paper describing their functionality and complexity. The architecture was described in Verilog HDL at RTL level. The complete system has been implemented in a Spartan-3 1000 FPGA device using ISE software from Xilinx, demonstrating it is a suitable design for high speed switches.

**Keywords:** Differentiated Services (DiffServ), Quality of Service (QoS), Traffic Scheduling, FPGA

## 1. INTRODUCTION

Nowadays, broadband services are growing incredibly due to technical improvements and social reasons. TV, Internet or Voice over IP (VoIP) are some representatives examples of recent demanded services. All of them generate a huge amount of traffic which needs routing and switching with special requirements. Specifically, these services introduce some traffic transmission parameters guaranteed by Quality of Service (QoS) mechanisms [1].

QoS is a networking term that specifies a guaranteed throughput level, which allows network providers to guarantee their customers that, i.e. end-to-end delay, will not exceed a specified level. The main QoS parameters also include loss rate, delay and jitter. However, the provision of a single class of QoS-controlled network service also requires the coordinated use of admission control, traffic access control, packet scheduling and buffer management. Only packet scheduling will be analyzed in this paper.

Since packets of many users may depart from the same outgoing interface, only schedulers that distinguish different flows or classes can be considered. So, packet scheduling specifies the queuing service discipline enforcing a set of rules in sharing the link bandwidth. In other words, packet scheduling prioritizes a user's traffic choosing the appropriate queue.

Generally, a packet scheduler should have some positive properties:

1) Low time complexity in order to select and forward a packet quickly;
2) Fairness among different flows;
3) Low worst-case delay and delay variation;
4) Simple enough architecture in order to be implemented efficiently.

The simplicity and time-complexity properties always collide with the fairness and delay-bound properties. Schedulers with short-term fairness and strict delay bound generally have high time complexity and are hard to be implemented.

VLSI Circuits and Systems IV, edited by Teresa Riesgo, Eduardo de la Torre, Leandro Soares Indrusiak,
Proc. of SPIE Vol. 7363, 73630B · © 2009 SPIE · CCC code: 0277-786X/09/$18 · doi: 10.1117/12.821522

O(1) time-complexity schemes are easy to be implemented, but they generally fail to provide short-term fairness and low scheduling delay bound.

In schedulers based on time-stamp, a virtual time clock is maintained to emulate the ideal Generalized Processor Sharing (GPS) [2]. It is an ideal scheduling policy in that it provides an exact max-min fair share allocation. GPS is fair in the sense that it allocates the whole outgoing capacity to all backlogged sessions in proportion to their minimum rate (bandwidth) requirements. Basically, the algorithm is based on an idealized fluid-flow model. That is, a GPS scheduler is able to serve all backlogged sessions instantaneously and the outgoing link capacity can be split infinitesimally and allocated to these sessions. However, in real systems only one session can be served at each time and packets can not be split into smaller units. An important class of so-called Packet Fair Queuing (PFQ) algorithms can then be defined which try to schedule the backlogged packets by approximating the GPS algorithm. Some examples are Weighted Fair Queuing (WFQ) [3], Virtual Clock (VC) [4] or Self-Clock Fair Queuing (SCFQ) [5].

The algorithms based on Round-Robin (RR) schemes are other possible solution. Usually, they are simpler to implement than time-stamp schedulers and have O(1) time complexity. However, RR schemes are also well known for their output burstiness and short-term unfairness. Deficit Round Robin (DRR) [6] and Smoothed Round Robin (SRR) [7] are typical round-robin schedulers.

However, the previous packet schedulers are not suitable to process variable length packets. If this kind of packets is considered, the packet switch internal architecture will be necessary to be deeply studied. Output Queuing (OQ) switches allow all incoming packets to arrive at output port achieving 100% throughput independently of their length [8][9]. However, memory speed must be increased limiting switch scalability. Input Queue (IQ), Virtual Output Queue (VOQ) or shared memory switches are focused on segmentation and re-assembly strategies [10] [11] and include switch matrix speed-up in order to increase the throughput. So, a variable length packet scheduler is an interesting challenge due to performance improvement [12] [13] in IQ architectures and hardware simplification in OQ architectures. In this paper, the chosen reference architecture will be an OQ switch in order to obtain the best throughput, make easier the QoS support (memory storage distribution) and avoid segmentation and re-assembly strategies with variable length packets. However, a memory speed solution is required and will be presented later.

In this paper, a packet scheduler implementation with QoS support is proposed. The rest of this paper is organized as follows. In Section 2, the reference architecture, OQ switch, is presented. In section 3 the most relevant details about Differentiated Service (DiffServ) model and service parameters to support QoS are given. The algorithm is described in section 4 and its architecture and synthesis are detailed in sections 5 and 6. Finally, in sections 7 and 8, the performance and conclusions are shown.

## 2. REFERENCE ARCHITECTURE

The reference architecture presented in this paper is a real Output Queuing switch. This architecture, named GMDS (Gigabit MultiDrop Switch) [14], is based on multidrop transmission, specifically in a high speed multidrop backplane avoiding the need of a switch fabric. In Fig. 1, an implementation of a 4 × 4 switch based on the GMDS architecture is presented.

Multidrop feature implies that every line card has a dedicated asynchronous uplink connected in point-to-multipoint to all the line cards in the switch. Consequently, inherent multicast/broadcast transmission and variable length packets are supported.

In Fig. 1, four line cards named as #0, #1, #2 and #3 are interconnected through this high speed multidrop backplane. Each line card has an input port (Ingress) and an output port (Egress). In the GMDS, the Ingress is transparent to packets, which are inserted in the multidrop backplane. Each Egress is listening to all Ingress connected to the backplane, decoding the destination address. Only packets with destination addressed related to each particular Egress are accepted and enqueued.

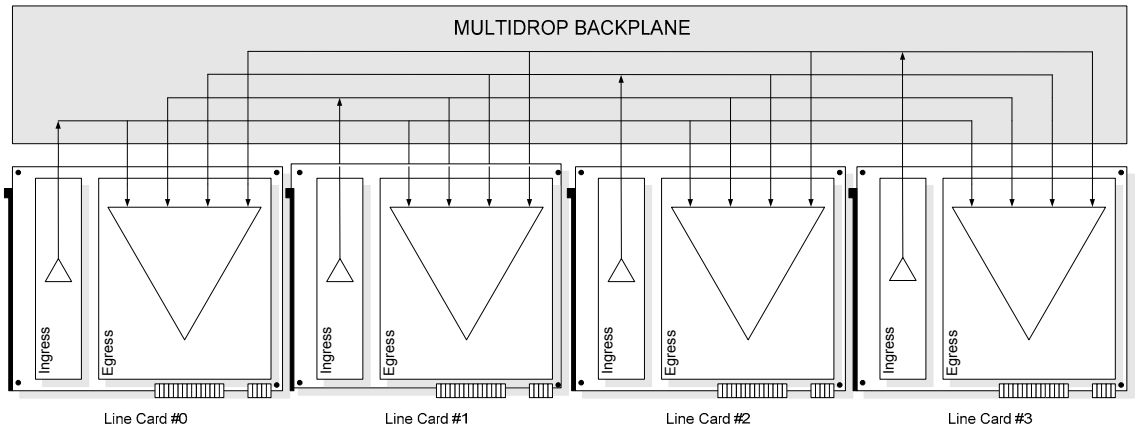

Fig. 1. A 4x4 GMDS general architecture.

The main features of this switch architecture are:

1. Non Switch Fabric architecture, increasing system reliability.
2. Real Output Queuing structure, avoiding HOL blocking problems of systems based on input queues.
3. Variable length packet support. This design has a dedicated link for each transmitter, so, inherently, variable length switching without speed-up requirement is supported.
4. Segmentation & reassembly of packets are not needed, improving overall efficiency, since functions like padding are not required.
5. Broadcast/Multicast natively supported.
6. Packets classification in multiple output queues based on traffic classes and sources according to Differentiated Services philosophy.
7. Enhanced QoS support, since scheduling can be performed over output enqueued traffic.
8. Accurate end-to-end flow control based on individual threshold configuration in each queue.

### 2.1 Detailed Description

The architecture of the GMDS is composed by two main elements: the multidrop backplane and the line cards. The line cards are responsible of frame reading/processing/enqueuing. The multidrop backplane, which replaces the switch fabric, distributes the packets among the line cards.

**Multidrop Backplane**

A gigabit multidrop backplane is presented in [15]. This backplane is based on broadband power splitters that accept an input signal and deliver multiple output signals with specific phase and amplitude characteristics, while maintaining a good impedance match at all ports. These features make maximum data bit-rate of a multidrop serial link to be limited principally by the PCB fabrication material, which theoretically allows achieving 10 Gbps using FR4 substrate.

**Line Cards**

The detailed architecture of each line card in a 4 × 4 GMDS is shown in Fig. 2.It is composed by the following modules: *Ingress Manager*, *Frame Filter*, *Queue Manager*, *Multiplexer*, *Status Manager*, *Traffic Scheduler* and *Memory System*.

*Frame Filter*
Each line card has a *Frame Filter* module dedicated to each multidrop downlink on the backplane. Each *Frame Filter* deserializes the multiframe, extracts each packet, and selects packets which destination address matches the line card id. Also a direct port assignation in the frame header can be performed, in order to support multicast. Selected packets are transferred to the *Queue Manager* module in order to be enqueued in the *Memory System*. The status & flow control information contained in the multiframe is decoded and relevant information to the line card is extracted.

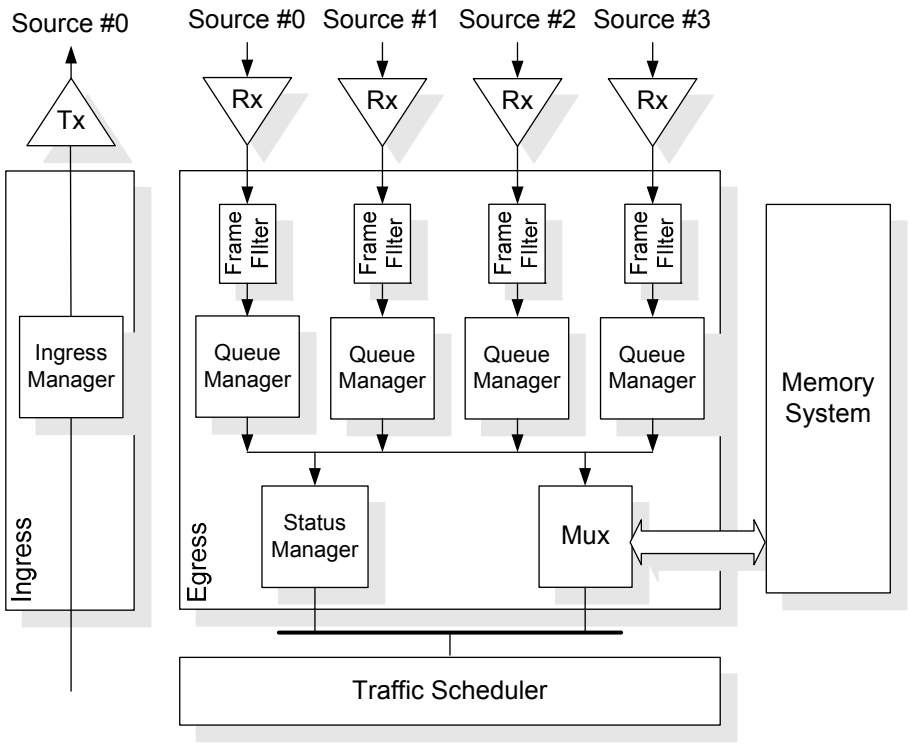


Fig. 2. 4x4 line card architecture.

The *Frame Filter* module also performs clock rate adaptation between remote *Ingress Manager* and local Egress clock, since to simplify clock distribution, each line card can operate based on their own local clocks.

*Queue Manager*

The *Queue Manager* module assigns a slot in the *Memory System* to each incoming packet. Packets are enqueued separately by source downlink and packet class. In the current implementation, the *Queue Manager* module uses fixed size slots of the maximum packet size. The *Queue Manager* module also reports to the *Status Manager* when a new packet is received, together with its class and source downlink. In addition, the *Queue Manager* module selects a slot to be dequeued when a specific source/class is requested to be read by the *Scheduler* module.

*Multiplexer*

In the GMDS, the memory bandwidth is splitted among the active *Frame Filters/Queue Managers* in the line card. However, the data rate and the memory rate will not necessarily match. The *Multiplexer* is a module that couples data flows from one/several *Queue Managers* to the *Memory System*.

The internal clock of the *Multiplexer* module runs at 3× the nominal frequency of the *Queue Manager* module, so in one *Queue Manager* cycle the multiplexing system is able to write from two assigned sources on a memory, and has an extra cycle reserved for a read operation from the *Scheduler*. When the number of ports is increased, a memory device has to be added per each pair of ports, together with a *Multiplexer* module. This approach is very important to guarantee the switch scalability avoiding memory speed bottleneck.

*Status Manager*

The *Status Manager* module collects information about packets incoming/outgoing of the Egress, keeping an accurate frame count on any queue. It reports about the empty/fillness levels of each single queue to the *Scheduler*. Besides, this module keeps some threshold levels per class to trigger flow control mechanisms. Since packets are queued per source/class, an end-to-end flow control can be supported.

*Traffic Scheduler*
Packet scheduling specifies the queue service discipline in GMDS. Since packets may depart from the same outgoing interface, packet scheduling also enforces a set of rules in sharing the output link bandwidth. Consequently, the *Scheduler* module defines the performance of the packet switch and provides the QoS features, being therefore a key module on the system.

## 3. SPECIFICATIONS

Traffic packets are classified depending on its importance in individual queues in the GMDS and, later, they are forwarded using a programmed scheduler per node. This approximation allows that traffic packets with similar service can cross multiple nodes receiving the same service in each one with the same configuration. This topology corresponds to the Differentiated Service (DiffServ) model [16], whose main advantages can be briefed in end-to-end signaling removing and efficient improvement because packet classification and scheduling are based on traffic classes instead of data flows.

The service differentiation allows including a lot of transmission necessities in a few data flows. The main objective of this approach is to provide scalable service discrimination without maintaining information about each individual data flow. So, DiffServ employ a small and well defined block structure capable of attending a large amount of services. Internally, a field in the packet head is defined. This field contains some bits with information about Types of Service (ToS). Also, ToS determines the packet service in each network node according to previously established Classes of Service (CoS): Expedited Forwarding (EF), Assured Forwarding (AF) and Best Effort (BE) forwarding.

In order to guarantee the different services, packets must be classified based on the ToS value. Consequently, each type of service has an independent queue available and packets with the same service parameters are stored in the same queue. The packets are chosen from the queues by the packet scheduler depending on the service parameters. Consequently, the proposed algorithm must consider all of them in order to guarantee the different Classes of Services and support QoS. These parameters, configured externally, are:

- **Weight**. The weight is a programmable ratio with regard to the whole bandwidth. This feature, along with variable length packet, makes the packet scheduler design more difficult. It must be taken into account that each traffic class has a fixed weight value.

- **Priorities**. Nowadays switches architectures commonly support Differentiated Services using priorities mechanism [17] [18]. This priority allows assuring a delay to each traffic class. The packet scheduler must also maintain the bandwidth distribution among traffic classes independently of the traffic priorities.

- **Loss probability**. When a queue is almost full or completely full, the flow control mechanism must send a message to the traffic source in order to stop or reduce the data transfer. The type of the message depends on the filling level controlled with thresholds. This mechanism is outside the objectives of this paper.

Finally, the **variable length packet** issue must be remarked. The proposed algorithm must share the bandwidth taking into account the weight. However, a classical approximation, as for example Weight Round Robin [19] or Deficit Round Robin [6][20], is impossible due to variable length packet and priorities combination. A credit based system is proposed to be implemented in order to control variable length packets in a similar way to Deficit Round Robin. Nevertheless, the algorithm must also include a priority control system. In the next section, both ideas are developed together.

## 4. PACKET SCHEDULER ALGORITHM

Before presenting the proposed packet scheduler algorithm, it is necessary to remark some details about its functionality. These details are the time frame-based approach and the work conserving service.

- **Time frame-based concept**. One important aspect in the packet scheduler is the bandwidth control. In the proposed algorithm, time is divided in frames. Each traffic class reserves a part of the frame depending of his weight value. However, the packet length must be taken into account. Consequently, the weight is transformed in a credit value to adequately consider the nature of the packets. When all the traffic classes have consumed its proportional part of the frame, the frame is restarted, that is, the credit value is restarted. In the middle of a frame, some traffic classes with credit can send packets, while other traffic classes have consumed all their credits and must wait until the next time frame.

- **Work conserving service**. There are two types of scheduling classification: non-work and work conserving service. In the first group, a packet is transmitted in the right instant based on an internal tag. So, it is possible that some packets are stored but the scheduler is idle. This policy assures the delay parameter but could reduce the throughput.

  In the second group, the algorithm sends a packet if there is a backlogged one. The packet scheduler is never idle if there is a queued packet. That is, non empty traffic classes can get the whole bandwidth even if empty traffic classes do not have consumed all their bandwidth. The packet scheduler restarts the frame if all the traffic classes have consumed its credit (or frame time) or if all the non empty traffic classes do not have credit (or frame time). So, after restarting the frame, non empty traffic classes can continue sending packets. In comparison to the non-work conserving algorithms, throughput is maximized but bandwidth distribution could not be assured in some specific situation such as low traffic environment.

In Fig. 3, the different steps to choose a queue and maintain the QoS parameters in the proposed scheduling algorithm are shown.

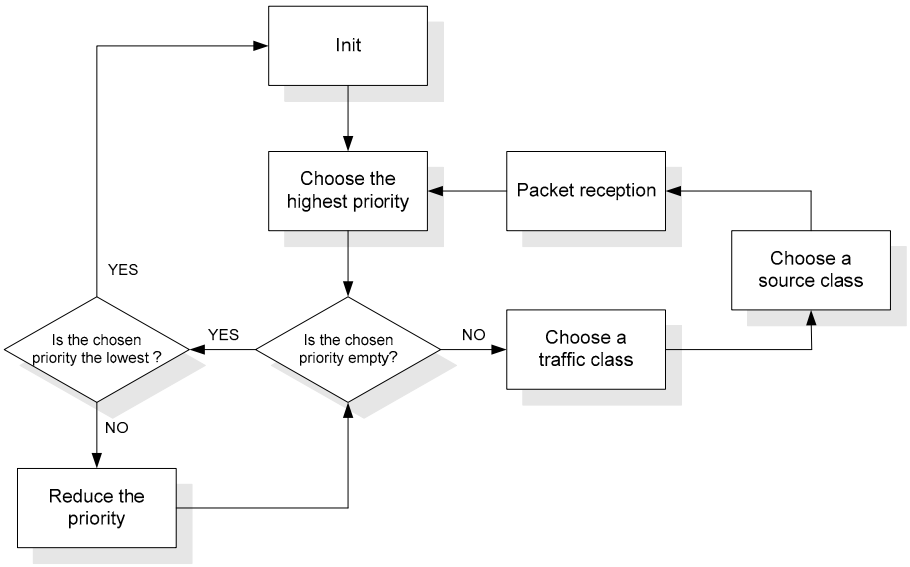


Fig. 3. Proposed packet scheduler algorithm.

According to this algorithm, the queue fillness is continuously polled by the packet scheduler. With this information, the packet scheduler must firstly choose a traffic class and, later, a traffic source, as shown in Fig. 3. Before the packet scheduler starts, the configuration process assigns a priority and a weight to each traffic class, that is, to each queue. Five steps based on hierarchical selection [21] are considered in the proposed scheduler.

1. **Priority.** The highest priority is selected. When this priority only has unavailable traffic classes, the priority is reduced. This functionality allows controlling the packet delay of any traffic class creating similar services to EF, AF and BE.

2. **Traffic class.** Each traffic class is assigned to a specific priority. When this priority is selected, a Smoothed Round Robin (SRR) [7] algorithm is performed among all non empty and available classes assigned to it.

   The traffic class selection can be also performed using a basic RR algorithm. However, traffic burst can be generated depending on weight and priorities configuration and packet lengths. For this reason, a SRR algorithm has been chosen instead of RR algorithm in order to improve fairness.

3. **Traffic Source.** When a traffic class is selected, all non empty traffic sources assigned to the selected traffic class are checked. Then, the traffic source which has sent the smallest amount of information of the previous selected traffic class is selected. This decision is justified due to variable length packet. For this reason, it is not possible to apply a RR philosophy and the scheduler must know the traffic sent by each source in order to be fair.

4. **Update information.** After choosing traffic class and source, the packet scheduler sends a request, including the selected traffic class and source, to the queues. Packet scheduler reads the packet length to update the information about the traffic class, assuring the weight and the flow protection. This mechanism is based on the idea of credits associated to all the traffic classes according to the weight assigned to the traffic classes. If a traffic class is empty, or has sent too much information in relation to its weight (no credit available), it is unavailable. The importance of this step is to guarantee the bandwidth control.

   At the same time, the packet length is used in the traffic source selection. Each traffic source has a register in charge of controlling the amount of information sent by each traffic class. Consequently, when a packet is sent, the respective register value is increased in the packet scheduler in order to choose the right traffic class in the next packet selection process.

5. **Init.** If it is not possible to choose a traffic class, the packet scheduler will be restarted. At this moment, the init process unlocks all the traffic classes making all of them available again, and restarts the credit values and the traffic source registers.

The configuration information of this packet scheduler algorithm consists on the assignation of a weight and a priority value to each traffic class. In the init step, the algorithm uses all these values for the next frame. So, a plug-and-play configuration is possible.

The unique information provided to the packet scheduler is the number of packets stored in each FIFO queue. So, the packet length is completely unknown by the packet scheduler. For this reason, firstly, a traffic class and source are selected and, secondly, the packet length is read and processed in the update information step.

## 5. PACKET SCHEDULER HARDWARE ARCHITECTURE

The packet scheduler algorithm description has been introduced in the previous section. In the current section, and before describing the packet scheduler implementation, a hardware specification must be presented to understand adequately the packet scheduler algorithm and its implementation: The interface between the network processor and the switch element is CSIX [22]. The most important characteristics of this interface are: variable length packet among 12 and 264 bytes and a ToS in the head packet with the traffic class value. However, the guaranteed maximum length is 256 bytes to assure an optimal memory distribution.

An initial approximation to the implementation of the proposed packet scheduler is shown in Fig. 4. This architecture is divided into three main modules.

1. **The Interface Module** is in charge of reading the status information from the packet switch, the number of packets in each queue, and the Configuration from user.

2. **The Traffic Class Selection Module** is in charge of traffic class selection. The internal structure of this module is shown in Fig. 5. In the selection process, some vectors and matrixes are required. The credit values are stored in the credit vector (1) of length T traffic classes. The status information is stored in the traffic class availability vector (2) of length T traffic classes in order to know if a traffic class is empty or non empty. Also, the configuration information (3) about matching among traffic classes and priorities is necessary. With all this information, a selection matrix (4) of P rows, being P the number of priorities, and T columns, is created. If a non empty traffic class is assigned to a priority, and its credit value is greater than zero, a true value is stored in the selection matrix. In the packet selection process, the first row -the highest priority- is checked, and a SRR selection process is performed among all the active traffic classes. When all the credit is consumed, priority is reduced and the SRR selection process is repeated in the second row. When a packet is read, the credit value is reduced depending of its length.

3. **The Traffic Source Selection Module** is composed of a matrix of T traffic class rows and N traffic sources columns. In each matrix position, the amount of sent traffic is stored. When a traffic class is selected, the non empty N sources assigned to this traffic priority are checked and the traffic class which has sent the smallest amount of traffic is selected.

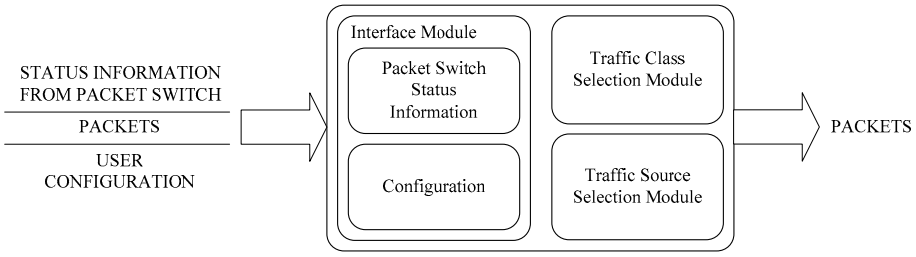


Fig. 4. General architecture.

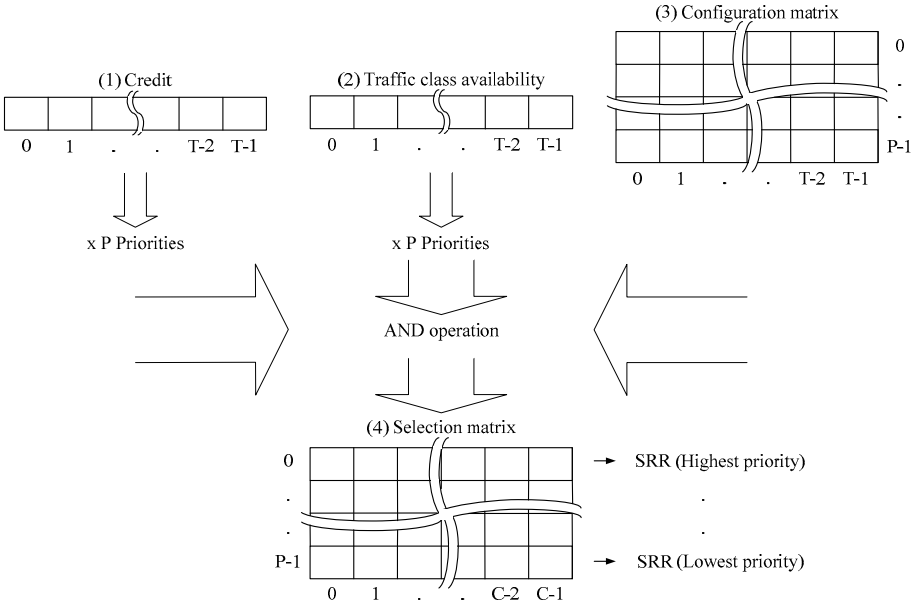


Fig. 5. Traffic class selection.

In addition to packet scheduler algorithm and implementation details described in the previous sections, it is necessary to remark and study some clock restrictions in the proposed packet scheduler implementation.

The worst case of operation in this architecture is the shortest packet length. In this case, this packet needs 3 clock cycles in order to complete the transmission in the CSIX format. The scheduler needs, at least, 3 clock cycles in order to read the packet length, choose the traffic class and choose the traffic source of the next packet. Finally, the *Memory System* needs 3 clock cycles in order to start the transmission of the next packet. Consequently, when a packet is extracted, in the first clock cycle the packet scheduler reads the packet length and, in the fourth cycle, it could ask for a new packet that will be extracted in the seventh cycle. If the packet is very short, the bus can be waiting, in the worst case, 3 clock cycles.

## 6. IMPLEMENTATION RESULTS

The proposed packet scheduler has been implemented in Verilog HDL at RTL level and synthesized in a Spartan-3 1000 FPGA (xc3s1000-4FT256) using Xilinx ISE software version 8.2.02i. The internal packet scheduler configuration is 16 traffic classes, 4 priorities and 4 different weight values. Queue size is limited to 7 bits, that is, 128 packets in each queue. The traffic classes are separated in 4 groups and each group has a priority and a weight. The packet scheduler also considers 4 traffic sources to each traffic class resulting 64 queues. This configuration is the worst hardware case because the hardware resources needed by the SRR algorithm are the largest. Synthesis results are shown in Table 1.

Table 1. Scheduler synthesis results.

| Logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| Number of Slices | 5254 | 7860 | 68 % |
| Number of slices Flip-Flops | 1595 | 15360 | 10 % |
| Number of 4 input LUT | 8050 | 15360 | 52 % |
| Number of Block RAMs | 2 | 24 | 8 % |
| Number of MULT 18x18s | 16 | 24 | 66 % |
| Number of GCLKs | 1 | 8 | 12 % |
| Equivalent gates | 271240 | | |

In some scenarios the SRR policy environments is not necessary in order to choose the traffic class. One example of this situation can be 16 traffic classes with the same weight and priority, that is, without QoS support. In this case, SRR algorithm can be replaced by RR. The main advantages of hardware modification are the hardware reduction, a 25 %, from 271240 to 219174 equivalent gates. So, this value can be associated with SRR improvement hardware cost. Based on the number of traffic classes, weight values and priorities, hardware cost can move between these two values.

In this paper, it is also relevant to compare the GMDS with the algorithm implementation hardware resources. The GMDS synthesis results are shown in Table 2. It must be remarked that the synthesis has been performed using the same software. However, the chosen FPGA device is a Virtex-II 4000. For this reason, only the number of equivalent gates is presented and the values correspond to the line card architecture shown in Fig. 2. The *Memory System* is implemented externally with commercial devices, MT55L512L32PT-6 (Micron).

Table 2. GMDS synthesis results.

| Module | Equivalent gates |
|---|---|
| 1 x *Ingress Manager* | 213884 |
| 4 x *Frame Filter* | 638585 |
| 4 x *Queue Manager* | 62361 |
| 1 x *Status Manager* | 101736 |
| 1 x *Multiplexer* | 9572 |

Some details can be also introduced to adequately understand these synthesis results. Internal FPGA memory is required in order to store packets addresses and, accept or reject the packets. Since this internal memory is included in the *Frame Filter* module, the number of equivalent gates is considerably increased comparing with the remaining modules.

Other important aspect is the proportion of GMDS to scheduler hardware resources. GMDS equivalent gates are 1004628 and scheduler equivalent gates are 271240, that is, the scheduler represents approximately a quarter of the GMDS hardware resources. Besides, this hardware results show the complexity of the Egress vs. the Ingress. Summarizing, the scheduler represents a large part in the system and it determines the performance of the switch.

## 7. SIMULATION RESULTS

The performance of the proposed packet scheduler was evaluated based on the simulation of Bernoulli arrivals and variable length packets among 12 and 264 bytes. Traffic classes, weight and priority configuration are shown in Table 3. These configuration values have been chosen to demonstrate the packet scheduler versatility in order to support different QoS parameters. The Y axis represents the Mean Queuing Delay in packets, which includes the queuing and output transmission delay, while the X axis represents the traffic load measured in the traffic generators. The packet scheduler is simulated in Verilog HDL using NC Compiler (Cadence).

Table 3. Packet scheduler configuration.

| Traffic Classes | Weight | Priority |
|---|---|---|
| 3, 7, 11, 15 | 12 % | ↑↑ (Highest) |
| 2, 6, 10, 14 | 8 % | ↑ |
| 1, 5, 9, 13 | 4 % | ↓ |
| 0, 4, 8, 12 | 1 % | ↓↓ (Lowest) |

Performance results are shown in Fig. 6. It must be remarked the different service depending on priorities. The highest priority has the lowest delay. At the same time, the weight also influences the delay, that is, the largest weight has more opportunities to be attended so it reduces the medium delay. Finally, this packet scheduler corresponds to an Output Queue switch, so the throughput should be very high. In this case, QoS support is maintained until a throughput of 96 %. At this point, the delay increases exponentially although packet scheduler continues switching. Besides, this figure shows the same treatment to traffic classes with the same service parameters.

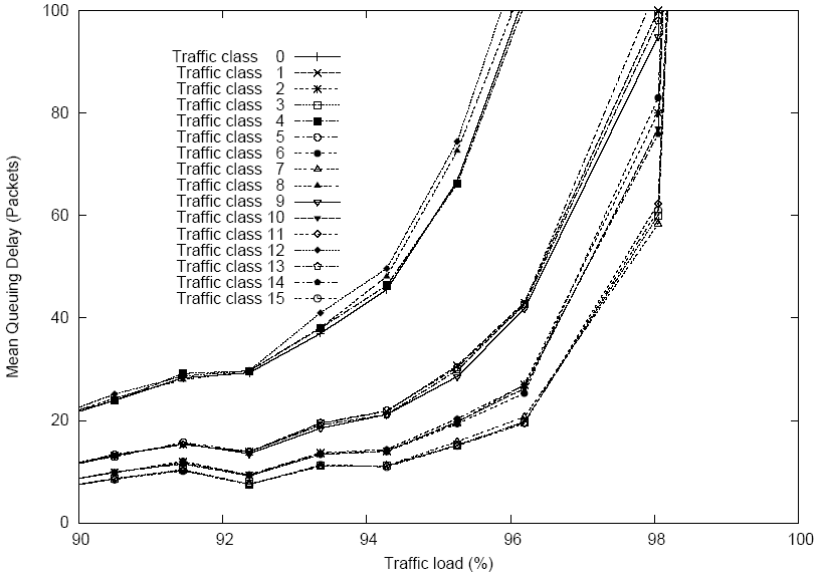


Fig. 6. Delay performance.

It is also verified that the packet scheduler maintains the bandwidth distribution independently of weights values, priorities configuration (flow protection), variable length packet, and traffic load. In Fig. 7, the constant lines of the bandwidth distribution can be observed.

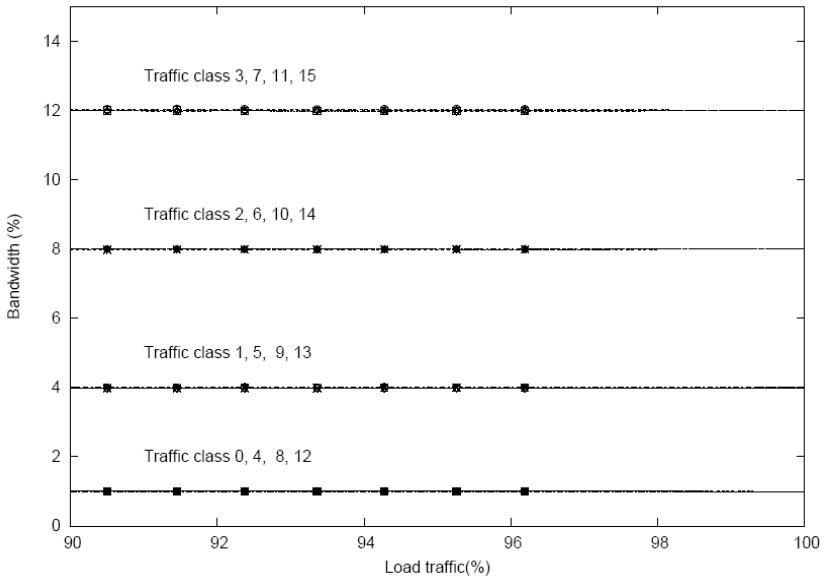


Fig. 7. Bandwidth distribution.

In order to demonstrate a suitable configuration for Real Time traffic (RT), traffic class 3 with the highest priority is chosen. Besides, weight control is deactivated for this traffic class and, consequently, the bandwidth of this traffic class is unlimited. Fig. 8 shows the mean queuing delay with real time traffic configuration. In this figure three important details can be observed. Firstly, traffic class groups in Fig. 6 and Fig. 8 show similar behavior. Secondly, maximum throughput is almost not reduced, and finally, traffic class 3 shows a constant delay suitable for RT traffic. However, it is necessary to add a weight control mechanism at the Ingress in order to assure a minimum protection in the system.

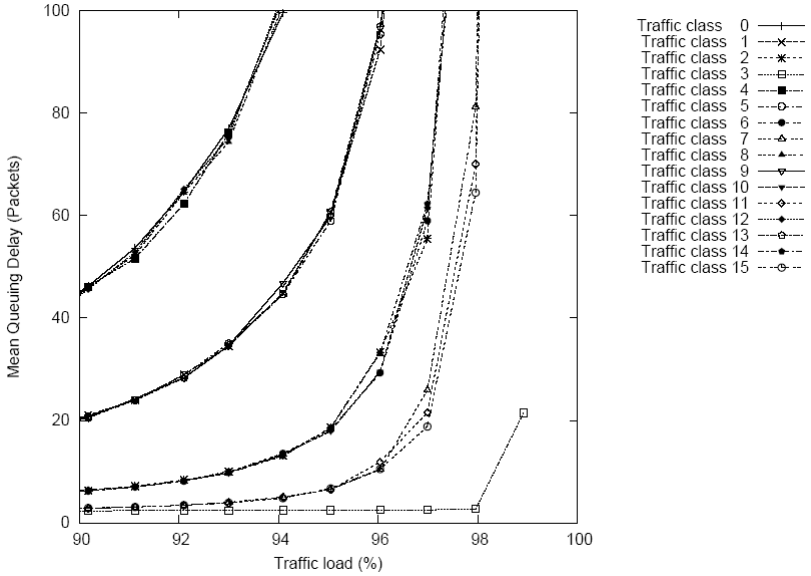


Fig. 8. Mean queuing delay with RT traffic configuration.

## 8. CONCLUSIONS

In this paper a packet scheduler implementation is presented. The main features of this proposal are versatility for current network and future services due to configuration possibilities (priority, bandwidth associated to any traffic class), low

hardware cost, high performance, inherent variable length packet switching due to multidrop backplane, inherent QoS support for OQ switches due to queue structures and flow protection independent of variable length packets and configuration. Besides, a traffic class with flow protection and constant delay can be also configured in order to service traffic with real time requisites.

## REFERENCES


[1] Xiao, X. and Ni, L. M., "Internet QoS: A Big Picture," IEEE Network 13(2), 8-18 (1999).

[2] Parekh, A. K., and Gallager, R. G., "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," IEEE/ACM Transactions on Networking 1(3), 344-357 (1993).

[3] Demers, A., Keshav, S., and Shenker, S., "Analysis and simulation of a fair queueing algorithm," Symposium Proceedings on Communications Architectures & Protocols (SIGCOMM'89) 1-12 (1989).

[4] Zhang, L., "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," ACM Transactions on Computer Systems (TOCS) archive 9(2), 101-124 (1991).

[5] Golestani, S., "A Self-Clocked Fair Queuing Scheme for Broadband Applications," Proceeding of the IEEE INFOCOM'94, 636-646 (1994).

[6] Shreedhar, M., and Varghese, G., "Efficient Fair Queuing using Deficit Round Robin," IEEE/ACM Transactions on Networking 4(3), 375-385 (1996).

[7] Guo, C., "SRR: An O(1) time complexity packet scheduler for flows in multi-service packet networks," IEEE/ACM Transactions on Networking 12(6), 1144-1155 (2004).

[8] Yoshigoe, K., and Christensen, K., "A parallel-polled virtual output queued switch with a buffered crossbar," IEEE Workshop on High Performance Switching and Routing (HPSR'01), 271-275 (2001).

[9] Moon, S., and Sung, D. K., "High-performance variable-length packet scheduling algorithm for IP traffic," Global Telecommunications Conference (GLOBECOM '01) 4, 2666-2670 (2001).

[10] Christensen, K., Yoshigoe, K., Roginsky, A., and Gunther, N., "Performance of packet-to-cell segmentation schemes in input buffered packet switches," Proceedings of IEEE International Conference on Communications (ICC'04), 2, 1097-1102 (2004).

[11] Katevenis, M., and Passas, G., "Variable-Size Multipacket Segments in Buffered Crossbar (CICQ) Architectures," Conference on Communications (ICC 2005), 2, 999-1004 (2005).

[12] Marsan, M. A., Bianco, A., Giaccone, P., Leonardi, E., and Neri, F., "Packet scheduling in input-queued cell-based switches," Proceedings of IEEE Conference on Computer Communications (INFOCOM'01), 1085-1094 (2001).

[13] Ganjali, Y., Keshavarzian, A., and Shah, D., "Cell Switching Versus Packet Switching in Input-Queued Switches," IEEE/ACM Transactions on Networking (TON'05), 13(4), 782-789 (2005).

[14] Arteaga, R., Tobajas, F., Esper-Chaín, R., De Armas, V., and Sarmiento, R., "GMDS: Hardware Implementation of Novel Real Output Queuing Switch," Proceedings of Design, Automation & Test in Europe (DATE'08), 1450-1455 (2008).

[15] Esper-Chaín, R., Tobajas, F., Tubío, O., Arteaga, R., de Armas, V., and Sarmiento, R., "A gigabit multidrop serial link for high-speed digital systems based on assymetrical power splitters", IEEE Transactions on Circuits and Systems II: Express Briefs, 52(1), 5-9 (2005).

[16] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W., "An Architecture for differentiated services", RFC 2475 Internet Engineering Task Force (IETF), (1998).

[17] Chrysos, N., and Katevenis, M., "Multiple Priorities in a Two-Lane Buffered Crossbar," GLOBECOM'04, 2, 1180-1186 (2004).

[18] Lee, T. H., and Kuo, Y. C., "Packet-based scheduling algorithm for CIOQ switches with multiple traffic classes," Computer Communications, 28(12), 1410-1415 (2005).

[19] Katevenis, M., Sidiropoulos, S., and Courcoubetis, C., "Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip," IEEE Journal on Selected Areas in Communications, 9(8), 1265-1279 (1991).

[20] Yamakoshi, K., Nakai, K., Oki, E., and Yamanaka, N., "Dynamic Deficit Round Robin scheduling scheme for variable length packets," Electronic Letters, 38(3), 148-149 (2002).

[21] Zhang, Y., and Harrison, P.G., "Performance of a Priority-Weighted Round Robin Mechanism for Differentiated Service Networks," Proceedings of 16th International Conference on Computer Communication and Networks (ICCCN 2007), 1198-1203 (2007).

[22] CSIX-L1: Common Switch Interface Specification - L1. May 2000.