Scalable Hardware Earliest-Deadline-First Scheduler for ATM Switching Networks

Byung Kook Kim
Department of Electrical Engineering
Korea Advanced Institute
of Science and Technology
373-1 Kusongdong
Taejon 305-701 Korea
bkkim@ee.kaist.ac.kr

Kang G. Shin
Real-Time Computing Laboratory
Department of Electrical Engineering
and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122
kgshin@eecs.umich.edu

Abstract

A fast, scalable hardware earliest-deadline-first(EDF) link scheduler for ATM switching network is developed. This EDF scheduler is a fast hardware solution suitable for real-time scheduler on nodes in ATM switching networks up to 2.5 Gbps switching speed (scheduling within 0.17 μs), capable of performing simultaneous input and output operations within two clock cycles (mostly in one clock cycle). The designed hardware is efficient since the architecture employs the minimum size EDF priority queue, combined with variable-size FIFO queues for channels implemented with a two-port memory buffer. Early traffic can be simply checked and delayed. Also, it is scalable with respect to the number of channels C and the total number of buffers N. Moreover, deadline folding technique eliminates the need to extend the deadline resolution. Simulation studies and layout design demonstrate the efficiency and utility of the proposed architecture.

Keywords: Real-time communication, real-time channel, ATM switch, EDF scheduler.

1. Introduction

The increasing demand of real-time network services has generated considerable interest in the development of real-time communication protocols. Such real-time communication requires quality-of-service(QoS) guarantees, such as bounded end-to-end delay, bounded cell-loss rate, and guaranteed bandwidth from the network. Real-time communication in packet-switched networks is characterized by applications having diverse traffic patterns and at the same time require certain quality-of-service(QoS) guarantees.

As a low layer of B-ISDN, Asynchronous Transmission Mode (ATM) provides fast switching of fixed size packets (cells with 53 bytes). An ATM network is composed of a set of switching nodes (ATM switches), connected by many communication links with various transmission speeds (155 Mbps, 622 Mbps, and 2.5 Gbps, etc). Expansibility of switching nodes and communication links makes the ATM network carry far more communication traffics compared to other communication networks such as FDDI network. Each node in the ATM switching network provides a switching function from incoming links to outgoing links. Each link usually contains various channels for end-to-end connections according to users' requirements.

The concept of *real-time channel* proposed by Ferrari and Verma [7] and refined by Kandlur et al. [11] is a unidirectional virtual circuit which, once established, is guaranteed to meet user-specified performance requirements as long as the user does not violate his *a priori* specified traffic-generation characteristics with parameters of *maximum message size* S_{max}, *maximum message rate* R_{max}, and *maximum burst size* B_{max}. Two distinct phases are required to realize the concept of real-time channel: off-line channel establishment and run-time message scheduling. During the channel establishment phase, the system has to select a route between the source and destination of the channel along which sufficient resources can be reserved to meet the user-specified delay and buffer requirements. Once established, traffic shaping and link scheduling algorithms ensure that the QoS requirements are satisfied for all connections passing through the node.

Various channel establishment algorithms are suggested [7][11]. Firoiu suggested an efficient flow admission control for EDF (Earliest Deadline First) schedulers with simple computation by introducing the notion of flex classes [6].

Recently, many service disciplines for scheduler are suggested. Simplest methods such as Stop-and-Go [9] and Hierarchical Round Robin [10] allocate the link's bandwidth, but introduces dependencies between delay and bandwidth allocation. Various algorithms are devised to distribute the end-to-end delay requirement into each node on the channel and utilizing the sorted priority queue: Generalized Processor Sharing (GPS) [14] utilizes an idealized fluid model, and reserves link bandwidth. Weighted Fair Queueing(WFQ) [5] uses the packet as the unit for assignment. Worst-case Fair Weighted Fair Queueing (WF^2Q) [1] is an improved version of WFQ which gives performance almost identical to GPS.

The insertion operation in the sorted priority queue has complexity of $O(logM)$, where M is the size of the queue. This makes high speed implementation of the algorithm very difficult. Hence, a hardware priority queue is needed to transmit packets at link's full speed. For example, in a 2.5 Gbps ATM network, an ATM cell can be transmitted every 0.17 μsecs. The priority queue must then determine the next highest priority cell every 0.17 μsecs, while being able to accept new cells from any incoming links. A hardware solution can operate at the link's required speed, and also it is possible to overlap enqueue and dequeue operations with packet transmissions.

Various hardware schedulers have been proposed in the literature: A binary tree comparator architecture utilizes N storage registers for buffering cells, and a binary comparator tree [15][16]. When $N = 2^n$, the architecture requires total of $(N - 1)$ comparators, and n comparison times are required to determine the highest priority (or the minimum deadline). It is conceptually straightforward, but disadvantages are decreased speed as N becomes large, FIFO ordering is not maintained among cells with the same priority, and it is not scalable. A set of FIFO queues and a priority encoder can be utilized to form a FIFO priority queue [2][3]. Each new entry is demultiplexed according to the priority, and stored in the FIFO of corresponding to its priority. A priority encoder is used to dequeue the highest priority entry — entry from the first nonempty FIFO. Main drawback is that it is not scalable to both the priority levels P and the number of buffers N. The shift register architecture consists of an array of blocks that stores the entries in sorted order [3][4]. Each block stores a single entry and communicates with blocks immediately to its left and right. Each block contains a comparator which compares priorities of the existing entry and the new entry — all blocks operate simultaneously, hence enqueue or dequeue operation can be performed in just one clock. The advantages are that it is scalable and fast. One disadvantage is a bus loading problem which adds to the hardware costs (buffers) and decreases the maximum operating speed of the queue.

The systolic array priority queue [12] has a series of systolic blocks, with enqueue/dequeue operations being performed sequentially starting from the highest priority block. The sequential behavior does not slow down the speed if pipeline operation is used. It does not have the bus loading problem. The main drawback is that twice the storage and twice the clock cycles are required compared to the shift register architecture. Moon and Shin [13] suggested a hybrid approach using modified systolic blocks — chips with a set of shift registers connected in systolic array architecture. This hybrid approach does not have the bus loading problem. Compared to the systolic array architecture, it requires less hardware with the same speed. Compared to the shift register architecture, the speed is slowed down by half. They treated cells in a link in the same way — the concept of channel is not used. They also proposed a priority queue which handles multiple links. Since each enqueue/dequeue operation requires 7 clock cycles each, which is not fast enough for high-speed ATM switch. Moreover, the available bandwidth should be divided into multiple links, again slows down the speed for each link. Zheng proposed a two-stage architecture [19][20]. In his approach, a memory with fixed size N is used as FIFO queues for all the C channels. Each channel has the same size FIFO queue of N/C cells, which results in a fixed size FIFO implementation. The enqueue or dequeue operation requires a max. of 12 clock cycles, which is not fast enough for a 2.5 Gbps ATM switch.

In this paper, we developed a hardware EDF scheduler, which is fast, efficient, and scalable. It has the minimum size EDF queue, and hence minimum bus-loading problem. The scheduler can perform simultaneous enqueueing/dequeueing within two clock cycles, and can also handle early traffic. Deadline folding technique enables finite, minimum bit representation of deadlines and parallel comparison of deadlines.

This paper is organized as follows. In Section 2, the problem is formulated and a new fast, efficient hardware EDF link scheduler is proposed. Section 3 shows the performance of scalability, and deadline folding technique. Evaluations with simulation studies and IC layout design in Section 4 shows the power and performance of the proposed hardware scheduler. This paper concludes with remarks in Section 5.

2. New Scalable Hardware EDF Scheduler

In this section, we formulate the hardware scheduler problem, and suggest a fast, scalable, two-stage hardware EDF link scheduler architecture. Our approach utilizes the shift register architecture with the minimum size, and a variable size FIFO for channels using a two-port memory. The resulting hardware is efficient in hardware, with minimum bus-loading on the new deadline input, and is fast enough for a 2.5 Gbps ATM switch.

2.1. Problem Statement

To provide the real-time channel communication in the ATM switching network, a real-time ATM switch structure is required which is composed of receivers for incoming links; a router; traffic shapers, link schedulers, and transmitters for outgoing links. The router performs VPI(Virtual Path Identifier) and VCI(Virtual Channel Identifier) translation from incoming cells using a look-up table, and generates new values of VPI and VCI as well as outgoing link number, channel number, and priority. The channel number will be used for channel identification in the link scheduler. The priority can be used to distinguish real-time traffic and non-real-time traffic. For non-real-time traffic, best-effort delivery using a separate large non-real-time FIFO usually suffices for vast number of VCI's. For real-time traffic with limited number of channels, traffic shaper algorithm can be utilized to assign the deadline for each packet such as jitter-EDD [17], Rate-Controlled Service(RCS) discipline [18], and per node traffic shaping [8]. After traffic shaper, a fast scheduler is ultimately required.

The problem for hardware EDF scheduler can be stated as follows:

Problem: Design a fast hardware EDF link scheduler for ATM switches with link speeds up to 2.5 Gbps which can schedule a cell every 0.17 μs. The scheduler should be scalable with respect to deadline resolution D, number of channels C, and number of cell buffers N.

2.2. Approach

The link scheduler and buffer should perform EDF scheduling and store all waiting cells. Various number of cells for each channel can be waiting due to various traffic patterns of channels. Since cells for a message in each channel have a sequence, i.e., the arrival sequence is the same as the delivery sequence, and usually $C << N$, we can use a FIFO cell storage for each channel, and an EDF queue with size of only C cells. Hence, a conceptual multi-channel EDF queue can be described as shown in Fig. 1.

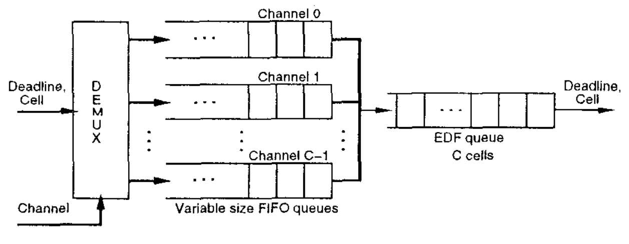

Figure 1. Conceptual model of link scheduler

Since storing the entire cell inside the FIFO requires a large size FIFO, it is more economical to store the cells in a cell buffer memory and instead store the cell's address in the FIFO [3][20]. Hence, the cell address and the deadline are stored in the FIFO for each channel as shown in Fig. 1.

Each channel has different traffic patterns — rates and delays — and hence different buffer requirements. Hence, buffer space requirements and the size of the FIFO for each channel i, N_i, can vary. Providing FIFOs with sufficient size (max. buffer requirement for each channel), N_{max}, for all channels (total CN_{max}) is inefficient: A shared FIFO for all channels with size $N = \sum_{i=0}^{C-1} N_i$ is more efficient. Hence, it is more desirable to have variable-size FIFO queue for each channel.

Also, the hardware solution should provide scalability to cope with various user requirements such as expansion of the deadline resolution, the number of channels, and the number of buffer requirements, etc.

2.3. Architecture

The architecture of our EDF link scheduler and buffer is illustrated in Fig. 2. The lower module C is the cell buffer for real-time cells. We used a two-port memory to enable simultaneous write and read operation. Since the departure sequence of cells for all channels is different from the arriving sequence due to allowable delay variation, an idle address FIFO is utilized to store and provide available addresses(pointers) in the cell buffer. This idle address FIFO should be initialized to contain all the available addresses in the cell buffer. Its output represents an available storage address in the cell buffer, which is called *cell-address* hereafter. When the scheduler provides a read address for the cell to be transmitted, a read operation at this cell-address in the cell buffer provides the cell output, and this read cell-address is returned to the idle address FIFO for future use.

The middle module B represents the storage corresponding to the variable size FIFO queues for all channels. For each cell, deadline and address of cell buffer are stored in the two-port memory data buffer, which enables simultaneous write and read operation. Available addresses in this data buffer, called *buffer-address* hereafter, are provided by an idle address FIFO. Since each FIFO queue is implemented as a linked list in the memory, we need to store the next buffer-address NA_i for each cell. The head and tail buffer-addresses of the linked list for each channel are stored in register RA_i and WA_i in the upper block A. Note that modules B and C can be implemented using off-the-shelf two-port memories and FIFOs. Hence, the upper module A is the only portion to be custom designed.

The upper module A contains the EDF queue and overall control logic for the link scheduler. The EDF queue is composed of C identical blocks, each composed with shift registers, comparators, and control logic. Each block stores the deadline, the channel number, and the cell-address. Also,

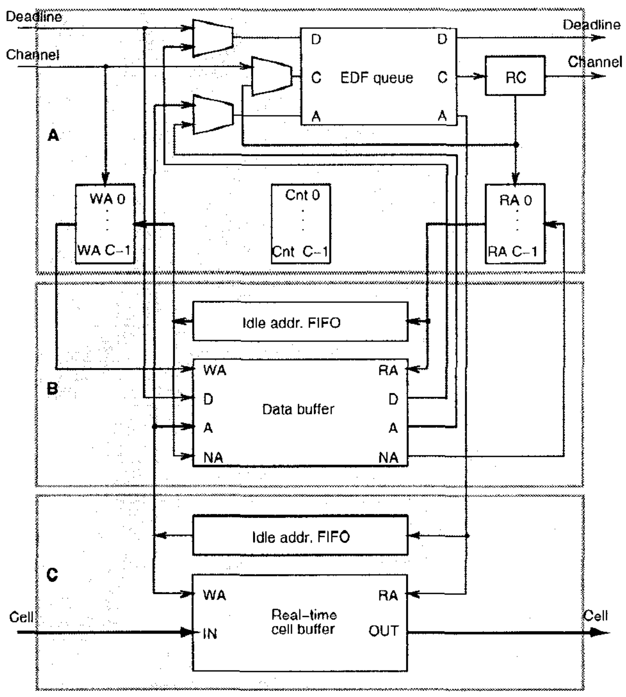

Figure 2. Structure of link scheduler and buffer

there are comparators in each block to compare deadlines of the enqueing cell and the resident cell. The number of cells stored for each channel, either in the data buffer or the EDF queue, are stored in the counter array Cnt_i, $i = 0, \cdots, C-1$. The corresponding Cnt_i for the current cell is incremented when enqueueing or buffering, and is decremented when dequeueing.

When an incoming cell for channel i arrives at the link scheduler, it is stored either in the data buffer or the EDF queue. If there are no cells for the channel of the incoming cell (F: $Cnt_i = 0$), it is stored in the EDF queue with the IQ operation: All cells having deadlines greater than the deadline of incoming cell are shifted left, and the corresponding rightmost cell is loaded with the incoming cell. When a cell with the channel number i is already stored in the EDF queue (F': $Cnt_i > 0$), the incoming cell's deadline, cell-address, next buffer-address is stored in the data buffer with the IB operation. The next buffer-address is supplied from the idle buffer address FIFO, and is also stored in NA_i, to be used by the next cell for channel i. Hence, the incoming cell's channel, deadline, and cell-address should be fed to both the data buffer and the EDF queue. Either input operation requires just one clock cycle.

When the output operation is requested, and if it is the last cell for this channel $j = RC$ (L: $Cnt_j = 1$), then the cell is dequeued from the EDF queue with the QO operation: When there are any cells in the data buffer for the channel (L': $Cnt_j > 1$), buffer-to-queue enqueueing and dequeueing operations are performed simultaneously with the BQ&QO operation.

The EDF queue in module A has structures similar to [4], but our link scheduler has much more features:

- *Minimum size queue:* Only one EDF queue with C blocks is required, compared to $(C + N)$ blocks in [4]: Only one cell for each channel is stored in the EDF queue. Other entries are stored in the variable-size FIFO queue of module B. Its control is more complex, but the overall hardware requires less chip space: A data buffer is much simpler than one block in the EDF queue. In addition, the bus-loading problem can be minimized: Reduced from $(C + N)$ to C.

- *Simultaneous enqueueing/dequeueing:* Simultaneous enqueueing/dequeueing can be performed in one clock cycle. Most simultaneous input/output operation can be performed in one clock cycle, with one exception of two clock cycles when input and output both requires enqueeing operation, as shown in the state diagram in Fig. 3. This performance improvement is possible due to the fact that we adopted the two-port memory where simultaneous writing and reading is possible, and our controller enables simultaneous enqueue/dequeue operation.

- *Early traffic handling:* When the deadline of the head cell on the EDF queue is too far (much larger than the current-time), the dequeueing operation can be simply delayed.

- *Deadline folding:* The deadline increases as the current-time increases. We implemented the deadline folding technique to limit the deadline within finite bit size, and performing parallel comparisons using this technique.

- *Scalability:* We suggested a scalable architecture where expansion with respect to C and N is simple and straightforward.

Features of our EDF scheduler will be explained in the subsequent Sections.

2.4. Early Traffic and Non-Real-Time Communication

We can handle the early traffic by adding hardware as shown in Fig. 4. Basically, a cell is transmitted when its deadline is due (same as the current-time). However, if several cells for a message have the same logical arrival time

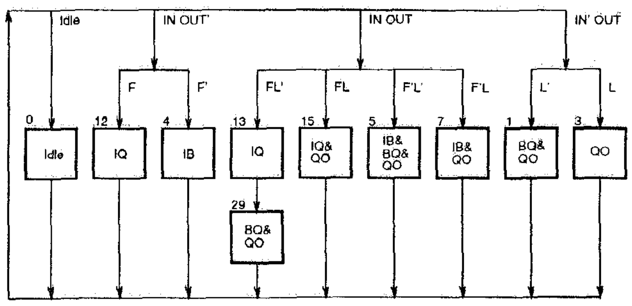

Figure 3. State diagram of link scheduler

and hence the same deadline, it is desirable to transmit cells whose deadline is within the near future. Also, we may transmit some cells somewhat earlier when the link is not busy. Hence, at time t, we transmit a cell with deadline less than $t + H$, where H refers to the *horizon*. Cells with deadline margins (deadline - current-time) larger than H are not transmitted, since bottlenecks on downstream nodes can occur if cells are transmitted too early.

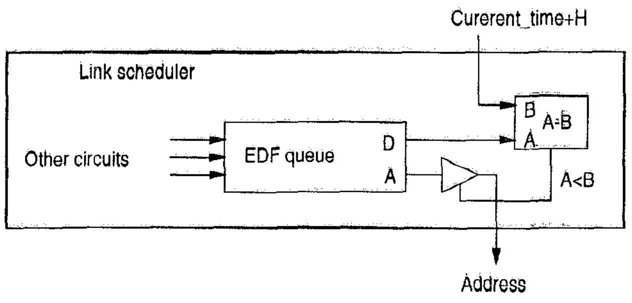

Figure 4. Early traffic handling circuit

Using an additional comparator at the deadline output of the queue, and comparing the deadline of the head cell A with B=(current-time + H), the dequeueing and tri-state buffer on the cell-address output are enabled only when $A < B$. Otherwise, dequeueing is not allowed. Although this method uses a fixed horizon for all channels and hence less flexible, it is simple to implement in hardware, and the buffer requirement is simpler.

Note that, our definition of horizon H is different from the definition H' suggested by Kandlur et al. [11]. They defined the horizon as earliness from the logical arrival time, not from the deadline. They used 3 queues: Queue 1 for real-time traffic with the logical arrival time $\leq$ current-time, queue 2 for non-real-time traffic, and queue 3 for real-time traffic with the logical arrival time $>$ current-time. When the queues 1 and 2 are empty, and the logical arrival time of the head cell in queue 3 is less than (current-time + H'), this cell is transmitted. Continuous examination of the queue 3 is required to transfer packets which have $l(m_i) <$ current-time, to the queue 1. Also, buffer space required for each channel i is given by:

$$\lceil (H' + d^i_{previous\_link} + d^i_{current\_link}) / I^i_{min} \rceil S^i_{max}.$$

Using our definition of horizon, only two queues are required — real-time queue and non-real-time queue — and examination for transferring (queue 3 $\rightarrow$ queue 1 in [11]) is not required. Also, our scheme requires smaller buffer space for each channel:

$$\lceil (H + d^i_{current\_link}) / I^i_{min} \rceil S^i_{max}.$$

In our early traffic handler, only one global earliness parameter can be set for all channels, whereas the delay jitter control which can be set for each channel [17]. More advanced traffic shaping algorithm [18][8] can be applied prior to our scheduler.

When there is no real-time traffic, non-real-time traffic can be transmitted. Non-real-time traffic is simply stored in a FIFO cell buffer. It is read whenever the real-time traffic is idle (empty real-time queue or all real-time cells' deadlines beyond the horizon).

3. Scalability

Scalability is essential for wide applicability of the implemented hardware. For the link scheduler, it is desirable to have scalability for the deadline resolution D, the number of channels C, and the number of cell storage N. Scalability w.r.t. D is solved using a deadline folding technique, and scalability w.r.t. C and N is solved by including additional logic for multiple-chip operation.

3.1. Deadline folding

If L bits are used for deadline representation, then the deadline will range between 0 and $2^L - 1$. Since deadlines need to be stored in the EDF queue, the number of bits L should be finite. Also, corresponding bits of registers and comparators should be used for each entry. However, the upper range required becomes ∞ as time goes by ($t \rightarrow \infty$). Hence, the number of bits L should be finite, and should be as small as possible.

We propose a deadline folding technique to solve this problem. Deadline folding uses finite L bits for deadline representation, utilizing modulo arithmetic. It resolves deadline comparisons in the transition region: Some deadlines are near the maximum ($2^L - 1$), and the other are near zero.

Consider the deadline vs. time graph as shown in Fig. 4. Here, U(t) represents the upper bound of cell deadlines stored in the EDF queue and the data buffer, and L(t) represents the lower bound of cell deadlines. Since we are using modulo arithmetic, L(t) can be represented as

$$L(t) = t \,\%\, D$$

where $D = 2^L$, and % represents the modulo operation. Let the maximum allowable delay (deadline - current-time) for all channels be d_{max}:

$$d_{max} = max\{d(m_0), d(m_1), \cdots, d(m_{C-1})\} - t$$

Any cell arriving at time t cannot obtain a deadline larger than $t + d_{max}$: Hence, U(t) can be represented as:

$$U(t) = (t + d_{max}) \,\%\, D \qquad (1)$$

Since any cell should be dequeued before the deadline, deadline $d(t)$ for any cell at time t should be in the range:

$$L(t) \leq d(t) < U(t)$$

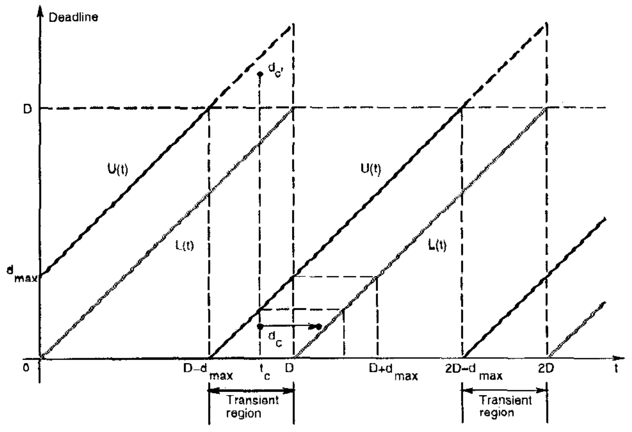

Figure 5. Deadline folding

Consider the transient region, where U(t) has changed to 0 but L(t) has not. A new cell c of message j for channel i arrives at t_c, and is assigned a deadline $d_{c'}$, which is supposed to be larger than D. For proper comparison with other cells' deadlines, the deadline should be $d_{c'}$; but due to modulo arithmetic, the deadline becomes $d_c = d_{c'} - D$. We can handle this by using the most significant bit (MSB) in the subtracter comparator as long as $d_{max} < D/2$.

For simplicity of explanation, suppose D is 8 bit, and $d_{max} < 128$. In the transient state, suppose the EDF queue has 4 deadline entries of $q_1 = E0_H, q_2 = F0_H, q_3 = 10_H, q_4 = 20_H$, and a new deadline entry of $e = F0_H$. Computation of $e - q_i$ with the 8-bit subtracter hardware result in the following:

$F0_H - E0_H = 10_H$ with no borrow, M=0
$F0_H - F0_H = 00_H$ with no borrow, M=0
$F0_H - 10_H = E0_H$ with no borrow, M=1 and
$F0_H - 20_H = D0_H$ with no borrow, M=1,

where M denotes the MSB of the result. In this case, for proper enqueue operation, q_1 and q_2 should remain, q_3 is loaded with e, and q_4 is shifted left (gets q_3 output). Hence, the borrow is useless in this case: Instead, we can use the MSB M. Let the MSB from the right(left) queue entry be $M_r(M_l)$. Then we can perform a proper operation using M and M_r:

When M=1 and $M_r = 1$ then shift left.
When M=1 and $M_r = 0$ then load the new entry.
When M=0 and $M_r = 0$ then no operation.

For simultaneous input/output operation, we can perform a proper operation using M and M_l:

When M=1 and $M_l = 1$ then no operation.
When M=0 and $M_l = 1$ then load the new entry.
When M=0 and $M_l = 0$ then shift right.

In summary, by using deadline folding technique with representing deadlines using one more bit than required for d_{max} representation:

$$L = log_2 D = \lceil log_2\, d_{max} \rceil + 1$$

and the most significant bit instead of the borrow bit, we can handle the deadline with finite, smallest number of bits. This scheme with the most significant bit is easily hardware implementable. The smaller the L, the smaller the shift register and subtracter that are required for each entry in the EDF queue.

A clock roll-over scheme by [16] uses a similar idea for logical arrival time determination with finite resolution. They determined the logical arrival time is early/late by determining $t - l(m_i)$ <Time-range/2. Neither detailed hardware explanation nor parallel expansion are explained there. Our scheme suggests a detailed, parallel implementation: Utilization of subtracters' MSBs instead of carry bits.

3.2. Channel Scalability

For any hardware solution, scalability (expandability) is very important for a wide range of applications.

The required size N of the cell buffer should be larger than or equal to the sum of buffer requirements for all channels:

$$N = \sum_{i=0}^{i=C-1} \lceil (H + d^i)/I^i_{min} \rceil S^i_{max}$$

which is dependent on the user application. We have to provide the cell-address entry A in the EDF queue with $\lceil log_2 N \rceil$ bits. Since the A entry in the EDF queue requires shift left/right registers, which is simpler than the D entry, we may provide sufficient bits for the A entry, say, 20 bits, to handle 1M cell buffers. The number of data buffers required is $N - C$, which approaches N if $C << N$.

An EDF queue with 256 cells is easily implementable with current hardware technology. When we require more channels for the link scheduler (many channels with slower traffics), we can combine several link scheduler modules. Fig. 6 shows an EDF scheduler expansion with 4 modules. Only one 2-to-4 decoder chip is required for external hardware. The first module at the top compares(subtracts) the deadline output D with the external deadline input DI (all 1's in this case). The lower value of the deadlines are selected by the multiplexer MUX and fed to the next module as DO output. The next module performs the same way, and the output of the final module will be the minimum of all deadline outputs of modules min(D). The output address lines are first fed to tri-state buffers and then connected together. Only one address output corresponding to the minimum deadline will be enabled via tri-state buffer with EI/EO logic. The EI input is fed to the lowest module as "1". If the $A < B$ output of the subtracter equals 0, this means the deadline form this module is not the minimum, the output EO becomes 1 by right AND gate, which means this module cannot output the address. If the $A < B$ output is 1, which means this module has the minimum deadline among modules from the top to itself. The EO output will be 0, and the tri-state buffer of this module is enabled, and its address A is the address with the minimum deadline A(min(D)). When the EI input equals 0 (lower modules already declared to have the minimum D), the module has no chance to output its address. Note that not the borrow bit but the MSB is used in the subtracter comparator, since we used deadline folding.

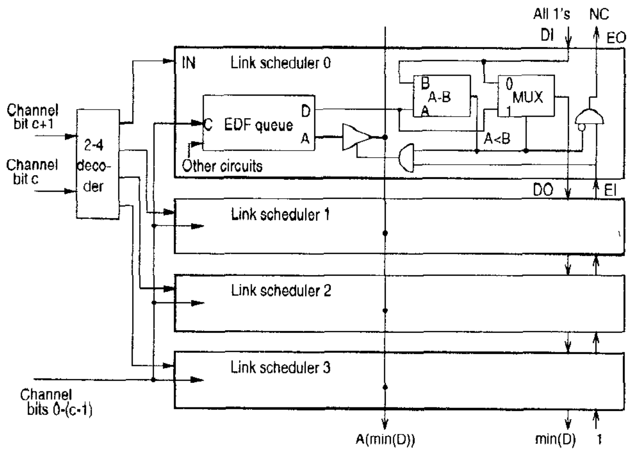

Figure 6. Channel extension with 4 link schedulers

By setting DI of the top module as *current-time*$+H$, we can get the minimum deadlines smaller than *current-time*$+H$, thereby handling the early traffic. The hardware for early traffic handling shown in Fig. 4 and the hardware for scalability in Fig. 6 have subtracter and tr-state buffer in common, which can be shared.

4. Evaluations

4.1. Simulation Results

To verify our design of real-time hardware scheduler chip, the architecture was simulated using the Verilog hardware description language, and the IC layout was designed using the Epoch compiler.

We wrote functional level descriptions and structural level descriptions in Verilog that describes link scheduler's the internal registers and controllers of the link scheduler. We input a set of packets to the link scheduler and checked if the internal state transients and sequences of the output packets were correct. A typical run with input packets is shown in Fig. 7. We used 16 bits of D, 8 bits of C, and 12 bits of N in this simulation.

In each line, T denotes the time — one clock period equals 100T. At the negative transition of the system clock (when T input signals are sampled: At the positive transition of the clock (T state transition occurs. Until $T < 10300$, the chip is initialized: CNT registers are cleared; WA, RA, and the buffer address FIFO are initialized. The data buffer read/write, queue input/output operations are performed in parallel within one clock cycle. We can verify that except when $103700 < T < 103900$, all input, output, and simultaneous input and output operations are performed in just one clock cycle. $103700 < T < 103900$ corresponds to the first-input&not-last-output, which requires two clock cycles — for IQ operation and then BQ&QO operation.

The deadline folding circuit also performs well. We assigned one bit of D as the bit indicating invalid entry (a deadline for nonoccupied cell). With the remaining 15 bits, the maximum modulo deadline is $2^{15} = 32768$. When T=104300, a new cell with deadline 2000 is input, which is a result of the modulo operation, and should be later than the previous cell's deadline of 32000. Our link scheduler uses the MSBs of subtracters, and can handle this situation neatly without much hardware overhead.

4.2. Layout Design

We also performed IC layout design using the Epoch compiler. The Verilog file was read to generate the netlist. The final design with 32 buffers and 16 EDF queues (but with the same D, C, and N widths as above) revealed to be a 105 pin IC with 6647 standard cells, which requires 110K transistor counts. Using the 0.8 micron CMOS technology, it requires 298 square mils of chip area, which is quite small. Note that, our design complexity is linear to the number of

```
T=1 RESET
T=551 Begin initialization. state=30 icnt= 0
T=102951 Initialized. state=30 icnt=1024
T=103151 state=12 in=1 inch= 7 cnti= 0 out=0 outch=63
T=103201 Queue input dl= 1024 ch= 7 adr= 11
T=103251 state= 4 in=1 inch= 7 cnti= 1 out=0 outch= 7
T=103301 dbuf write dl= 2048 adr= 12 nadr= 64 at 7
T=103351 state= 1 in=0 inch= 7 cnti= 2 out=1 outch= 7
T=103401 dbuf read dl= 2048 adr= 12 nadr= 64 at 7
T=103401 Queue Input dl= 2048 ch= 7 adr= 12
T=103401 Queue Output dl= 1024 ch= 7 adr= 11
T=103451 state= 3 in=0 inch= 7 cnti= 1 out=1 outch= 7
T=103501 Queue output dl= 2048 ch= 7 adr= 12
T=103551 state=12 in=1 inch= 1 cnti= 0 out=0 outch=63
T=103601 Queue input dl= 4096 ch= 1 adr= 1
T=103651 state= 4 in=1 inch= 1 cnti= 1 out=0 outch= 1
T=103701 dbuf write dl= 8192 adr= 2 nadr= 65 at 1
T=103751 state=13 in=1 inch= 2 cnti= 0 out=1 outch= 1
T=103801 Queue input dl=15000 ch= 2 adr= 3
T=103851 state=29 in=1 inch= 2 cnti= 1 out=1 outch= 1
T=103901 dbuf read dl= 8192 adr= 2 nadr= 65 at 1
T=103901 Queue Input dl= 8192 ch= 1 adr= 2
T=103901 Queue Output dl= 4096 ch= 1 adr= 1
T=103951 state=15 in=1 inch= 3 cnti= 0 out=1 outch= 1
T=104001 Queue Input dl=14000 ch= 3 adr= 4
T=104001 Queue Output dl= 8192 ch= 1 adr= 2
T=104051 state= 4 in=1 inch= 3 cnti= 1 out=0 outch= 3
T=104101 dbuf write dl=20000 adr= 5 nadr= 66 at 3
T=104151 state= 5 in=1 inch= 3 cnti= 2 out=1 outch= 3
T=104201 dbuf write dl=25000 adr= 6 nadr= 67 at 66
T=104201 dbuf read dl=20000 adr= 5 nadr= 66 at 3
T=104201 Queue Input dl=20000 ch= 3 adr= 5
T=104201 Queue Output dl=14000 ch= 3 adr= 4
T=104251 state= 7 in=1 inch= 3 cnti= 2 out=1 outch= 2
T=104301 dbuf write dl=32000 adr= 7 nadr= 68 at 67
T=104301 Queue output dl=15000 ch= 2 adr= 3
T=104351 state=12 in=1 inch= 5 cnti= 0 out=0 outch= 3
T=104401 Queue input dl= 2000 ch= 5 adr= 7
T=104451 state= 1 in=0 inch= 5 cnti= 1 out=1 outch= 3
T=104501 dbuf read dl=25000 adr= 6 nadr= 67 at 66
T=104501 Queue Input dl=25000 ch= 3 adr= 6
T=104501 Queue Output dl=20000 ch= 3 adr= 5
T=104551 state=12 in=1 inch= 6 cnti= 0 out=0 outch= 3
T=104601 Queue input dl= 1000 ch= 6 adr= 7
@@@@@ Done.
L135 "lstest.v": $finish at simulation time 104690
710183 simulation events + 638675 accelerated events +
244551 timing check event s
```

Figure 7. EDF link scheduler operation

buffers N and size of queues C, and hence can be scaled up easily.

By using 12.5 MHz or faster clock for the link scheduler chip, we can guarantee enqueue/dequeue operation within 0.16 μs (2 clocks), which satisfies the 0.17μs constraint for 2.5 Gbps operation of the ATM switch.

5. Concluding Remarks

A fast, scalable hardware earliest-deadline-first (EDF) scheduler for ATM switching network is developed. The major feature of this scheduler is a fast hardware solution for EDF link scheduling suitable for real-time channel implementation on ATM switching networks with switching speeds of up to 2.5 Gbps. The advantages are as follows: 1) Speed - simultaneous input and output can be performed within two clock cycles (mostly within one clock cycle). 2) Hardware efficiency - the minimum size EDF priority queue and minimum bus-loading problem, combined with variable-size FIFO queues for channels implemented with two-port memory buffer. 3) Early traffic handling - enables the dequeueing of entries with deadlines up to the horizon. 4) Deadline folding - finite, minimum bit representation of deadlines, resulting in simpler and faster hardware. 5) Scalability with respect to the number of channels C and to the number of buffers N. Simulation studies and layout design demonstrate the efficiency and power of the proposed architecture.

References

[1] J. Bennett, Hui Zhang, "WF^{2}Q: Worst-case fair weighted fair queueing," *Proceedings of the INFOCOM'96*, Washington DC, Dec. 1996.

[2] Randy Brown, "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220-1227, Oct. 1988.

[3] Jonathan Chao, "A novel architecture for queue management in the ATM network," *IEEE Journal on Selected Areas in Communication*, vol. 9, no. 7, pp. 1110-1118, Sep. 1991.

[4] Jonathan Chao and N. Uzun, "A VLSI sequencer chip for ATM traffic shaper and queue management," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1634-1643, Nov. 1992.

[5] A. Demers, S. Keshav, and S. Shenkar, "Analysis and simulation of a fair queueing algorithm," *Journal of Internetworking Research and Experience*, pp. 3-26, Oct. 1990.

[6] V. Firoiu, J. Kurose, D. Towsley, "Efficient admission control for EDF schedulers," *Proceeding of the INFOCOM'97*, Nagoya, Japan, April 1997.

[7] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Selected Areas on Communications*, vol. 8, pp. 368-379, Apr. 1990.

[8] L. Georgiadis, R. Guerin, V. Peris, and K. Sivarajan, "Efficient network QoS provisioning based on per node traffic shaping," *IEEE/ACM Trans. on Networking*, vol. 4, no. 4, pp. 482-501, Aug. 1996.

[9] S. J. Golestani, "A stop-and-go queueing frmaework for congestion management," *Proceedings of ACM SIGCOMM'90*, pp. 8-18, Philadelphia, Pennsylvania, Sep. 1990.

[10] C. R. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," *IEEE Global Telecommunications Conference*, pp. 300.3.1-9, San Diego, California, Dec. 1990.

[11] D. Kandlur, Kang G. Shin, "Real-time communication in multi-hop networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044-1056, Oct. 1994.

[12] P. Lavovie and Y. Savaria, "A systolic architecture for fast stack sequential decoders," *IEEE Transaction on Communications*, vol. 42, no. 2/3/4, pp. 324-334, Feb./Mar./April 1994.

[13] S. W. Moon, J. Rexford, and Kang G. Shin, "Scalable hardware priority queue architectures for high-speed packet switches," *Submitted for publication*.

[14] A. K. J. Parekh, R. G. Gallager, "A generalized processor sharing approach to flow control - the single node case," *Proceedings of the INFOCOM'92*, 1992.

[15] D. Picker and R. Fellman, "A VLSI priority packet queue with inheritance and overwrite," *IEEE Transactions on Very large Scale Integration Systems*, vol. 3, no. 2, pp. 245-252, June 1995.

[16] J. Rexford, J. Hall, and Kang G. Shin, "A router architecture for real-time point-to-point networks," *Proceedings of International Symposium on Computer Architecture*, pp. 237-246, May 1996.

[17] D. Verma, H. Zhang, and D. Ferrari, "Guaranteeing delay jitter bounds in packet switching networks," *Proceedings of Tricomm'91*, pp. 35-46, Chapel Hill, North Carolina, April 1991.

[18] H. Zhang, D. Ferrari, "Rate-controlled service disciplines," *Journal of High Speed Networks*, vol. 3, no. 4, 1994.

[19] Q. Zheng, Kang G. Shin, Real-time communication in local area ring networks," *Conference on Local Computer Networks*, pp. 416-425, Sep. 1992.

[20] Q. Zheng, "Real-time fault-tolerant communication in computer networks," Ph. D. Thesis, University of Michigan, 1993.