A Self-Timed Real-Time Sorting Network

Kenneth Y. Yun, *Member, IEEE*, Kevin W. James, *Student Member, IEEE*, Robert H. Fairlie-Cuninghame, Supratik Chakraborty, *Member, IEEE*, and Rene L. Cruz, *Senior Member, IEEE*

Abstract—
High speed networks are expected to carry traffic classes with diverse Quality of Service (QoS) guarantees. For efficient utilization of resources, sophisticated scheduling protocols are needed; however, these must be implemented without sacrificing the maximum possible bandwidth. This paper presents the architecture and implementation of a *self-timed real-time sorting network* to be used in packet switches that support a diverse mix of traffic. The sorting network receives packets with appropriately assigned priorities, and schedules the packets for departure in a highest-priority-first manner. The circuit implementation uses zero-overhead, self-timed, self-precharging domino logic to minimize the circuit latency. An experimental sorting network chip has been designed using the techniques described in this paper to support 10 Gb/s links with ATM size packets.

Keywords— **Sorting network, EDF scheduling, quality of service, self-timed implementation.**

I. Introduction

The explosive growth in network utilization in the past few years has triggered the development of a global high bandwidth network infrastructure. A challenging problem in the design of today's high speed networks is allowing prioritization of network access so that diverse quality of service (QoS) guarantees can be provided to users with different requirements. Central to this concept are the issues of: (i) assigning priorities to information packets, and (ii) sorting packets by their priorities at intermediate switches to determine the allocation of network resources. This paper addresses the second issue only (although there remains interesting new research in the priority assignment problem, despite some recent progress in the area [1], [2]).

Specifically, this paper describes a novel architecture for a high-throughput, low-latency 2×2 switch with *priority-based* conflict resolution and distributed buffering. The switch sorts conflicting packets — packets destined for the same output port — according to priority levels. The packet with the highest priority is transmitted, and lower priority packets are stored for future transmission. Sorting is done in real-time, so that the network can be used in real-time networked applications, such as video-conferencing.

Related work. Sorting networks using fast optical switches have been implemented in [3], [4]. However, these do not support priority-based sorting for QoS. In addition, viability of the optical switching techniques for immediate volume production is unclear. Priority-based hardware sorting algorithms have been presented elsewhere [5], [6]. These may be viewed as degenerate cases of our algorithm. Most of the related work comes from the ATM switching community. The work of [7] describes an ATM switch that supports sorting for QoS, but at a relatively low speed (500 Kb/s). A very fast electronic switch is discussed in [8], but priority-based sorting is not supported in it. Different ATM switch architectures with prioritized servicing are described in [9] and [10]. Unfortunately, these and other existing solutions do not support both high speed operation (e.g., 1 to 10 Gb/s) and sorting for QoS.

This work addresses key issues in practical packet switched networks: high throughput with low latency, priority-based sorting, and extensible architecture yielding large storage capacity. We describe the architecture in detail and explain how the buffer capacity of the network can be extended, without incurring excessive latency, by stacking multiple layers of the network. We suggest how slower (and presumably cheaper) sorting networks/systems can be used to extend the buffer capacity of our network, which is critical for tolerating bursts of conflicting packets. We then describe our implementation of the sorting network using zero-overhead, self-timed, self-precharging domino logic. Our implementation technique enables extremely fast operation of the network with zero control overhead. Finally, we provide a brief overview of a sorting network chip designed using the techniques described in this paper to support 10 Gb/s links with ATM size packets.

II. Architecture

This section describes the architecture of a 2×2 real-time packet sorting network. The sorting network receives up to two packets in every packet slot and transmits up to two packets at the end of each packet slot. We assume that there are two packet types, A and B, and two associated output ports, A and B. Type A packets are destined for output port A, and type B packets for output port B. Contention arises if two input packets of the same type arrive at the same time, in which case the sorting network stores one of the packets in a buffer. The criterion used to determine which one is to be buffered and which one is to be transmitted is based on the *priority* stamped on each input packet. In each packet slot, our sorting network outputs on port A the type A packet with the highest priority of all type A packets, both incoming and stored. Similarly, the type B packet with the highest priority is outputted on port B.

A. Basic Switching Element

The basic switching element, henceforth called 2×2 *crossbar*, used in our sorting network is illustrated in Fig. 1. It has two input ports (X and Y) and two output ports (A and B). If a single packet arrives, then it leaves the crossbar on its destined output port. If two packets of different types arrive in the same slot, then each packet is routed to its destined port. However, if two packets of the same type arrive in the same slot, the crossbar

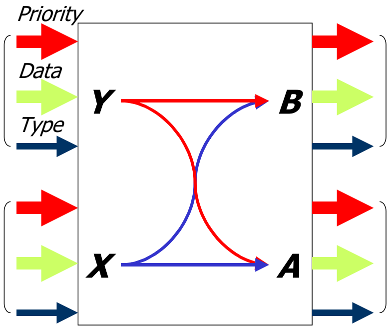
This research was supported in part by a NSF CAREER Award MIP-9625034 and by a gift from Intel Corp.

K. Y. Yun, K. W. James, and R. L. Cruz are with ECE Dept., University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0407. Email: kyy@ucsd.edu, kwjames@ucsd.edu, cruz@ece.ucsd.edu.

R. H. Fairlie-Cuninghame is with Nuera Communications, 10445 Pacific Center Court, San Diego, CA 92121. Email: rfairlie@cwc.ucsd.edu.

S. Chakraborty is with Fujitsu Laboratories of America, Sunnyvale, CA 94086-3922. Email: supratik@fla.fujitsu.com.

routes the packet with higher priority to its destined port and the packet with lower priority to the other port.



Cross only if		
Port X	Port Y	Priority Level
type B	type A	–
type B	none	–
none	type A	–
type A	type A	port X < port Y
type B	type B	port X > port Y

Fig. 1. Basic switching element: 2×2 crossbar.

B. TC Stage

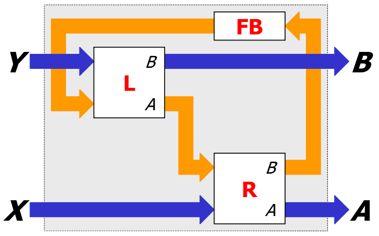
A TC ("Track Changer") stage is constructed from two 2×2 crossbars and a latch as depicted in Fig. 2. The TC stage functions similarly to the crossbar but with an additional buffering capability. *Assuming that incoming packets have been sorted by a crossbar*, the TC stage routes the packets with the highest priorities to their destined ports and stores the remainder in the buffer, unless all packets, both incoming packets and the stored one, are of the same type.

For example, if the packet stored in the buffer is type A and the incoming packets are of different type (type A on input port X and type B on input port Y), then crossbar L routes the type B packet to output port B and the type A packet stored in the buffer to crossbar R. Crossbar R compares the priority levels of the two type A packets, routes the one with higher priority to output port A, and routes the one with lower priority to the buffer.

However, if all three packets are of type A, then crossbar L has no choice but to route a type A packet to output port B. Because the incoming packets are assumed to have been sorted by a crossbar, the type A packet entering input port Y must have lower priority than the one entering input port X. The priority of this packet is compared to that of the type A packet stored in the buffer. Since the packet with the lower of the two priorities is routed to port B, the one leaving on port B must have the lowest priority of all three. In contrast, the one with the highest priority leaves on its destined port, port A, and the remainder (with the medium priority) is stored in the buffer.

C. TC Chain

A TC chain of length N is constructed by cascading a 2×2 crossbar and N TC stages in series. The TC chain implemented



Packets	Port A	Port B	Buffer
$3A$	A_{high}	A_{low}	A_{medium}
$3B$	B_{low}	B_{high}	B_{medium}
$2A, B$	A_{high}	B	A_{low}
$A, 2B$	A	B_{high}	B_{low}
A, B	A	B	–

Fig. 2. TC stage and its function. We assume that packets have been sorted by a crossbar before entering the TC stage. $3A$ means that both the incoming packets and the packet stored in the buffer are all type A; $2A, B$ means that two are type A and one is type B. A_{high}, A_{medium}, and A_{low} are type A packets with *high*, *medium*, and *low* priority levels respectively.

as a cascaded optical delay line *with no priority based sorting* was first introduced in [3]. In this paper, we extend that work to an electronic implementation with priority based sorting.

A two-stage TC chain is shown in Fig. 3. The crossbar labeled 0 sorts the incoming packets and routes them to stage 1 of the TC chain. When an incoming packet arrives, crossbar L_1 sorts it against the packet stored in FB_1 and routes a packet each to R_1 and L_2. R_1 and L_2, in turn, sort packets and route them to the next stage.

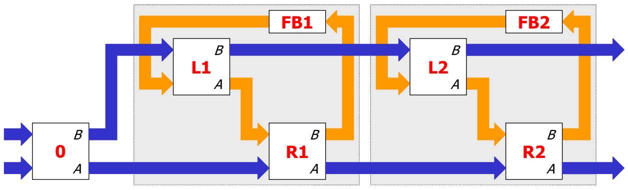


Fig. 3. 2-stage TC chain.

It is not difficult to verify that a TC chain constructed this way has the following properties.

- Type A (B) packet leaving on output port A (B) has higher priority than all other type A (B) packets stored in the chain.
- Type A (B) packet leaves on output port B (A) if and only if
  - both incoming packets are of type A (B) and
  - the TC chain is filled only with type A (B) packets with higher priorities.

In other words, the packet leaving on port A is either the type A packet with the highest priority or the type B packet with the lowest priority.

A drawback of this architecture is that the latency of a TC chain is proportional to the number of stages, N, in the chain. That is, type and/or priority comparison must be made in every stage serially before determining which packets are to leave the chain in a given packet slot. Therefore, increasing the buffer capacity also increases the sorting latency.

D. TC Stack

In general, we need large storage capacity without incurring long sorting latency. To increase the buffer capacity without increasing the sorting latency, we employ the architecture depicted in Fig. 4. Incoming packets enter the bottom TC layer. Overflow packets from layer L are "pushed" onto layer $L+1$. In a given packet slot, packets "popped" from layer $L+1$ (descending from $L+1$ to L) have the highest priority of the incoming packets to layer $L+1$ and all the packets stored in layers $L+1, L+2, \ldots$. We cap the sorting latency at the delay through a single TC layer by insuring that the bottom layer contains (at least) one packet of each type with higher priority than the packet of the same type descending from overflow layers.

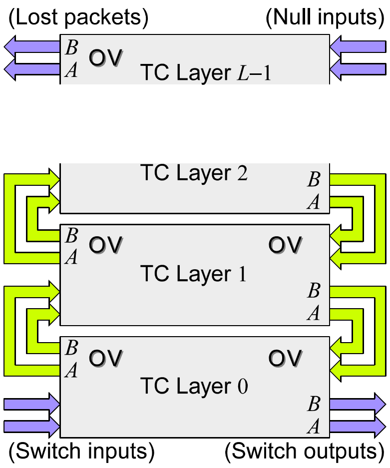


Fig. 4. TC stack.

Each TC layer is implemented as shown in Fig. 5. *Mixed* buffer is the main sorting network implemented as an N-stage TC chain. A/B-buffers are shorter M-stage TC chains ($N \geq M$). Because A-buffer and B-buffer are completely symmetrical as shown in Fig. 5, without loss of generality, we limit our discussion to A-buffer only. If it is assumed that packets entering overflow input port X (see Fig. 5) are only of type A, then A-buffer never receives two type B packets in a packet slot. Therefore, if a type B packet enters input port X of A-buffer, it leaves A-buffer via overflow port B in the same packet slot. This means that no type B packet can be stored in A-buffer, which is the reason it is named as such. It is also easy to verify that a type A packet leaves via overflow port B (see Fig. 5) only if A-buffer is filled with M type A packets with higher priorities and both the packets entering A-buffer are of type A.

Suppose a packet $\mathcal{P}$ of type A overflows from layer L and requires a minimum of K packet slot delays to descend back to L. We define the minimum *sojourn time* of $\mathcal{P}$ to be $K \times$ (packet slot delay). In order to insure that no new type A packet with lower priority than $\mathcal{P}$'s "leaks out" from layer L via output port A before $\mathcal{P}$ returns to this layer, there must be at least K type A packets with higher priority stored in layer L. We enforce this by choosing $M \geq K$ and $N \geq K$ in the TC stack. To understand why this works, consider the scenarios that can lead to a type A packet leaving layer L through an overflow port:

- If it left via overflow port A, then it must have entered B-buffer from *Mixed* buffer. *Mixed* buffer must therefore have been filled with N type A packets with higher priorities. As long as $N \geq K$, there is no "leakage" problem.
- If it left via overflow port B, then it must have departed from A-buffer and A-buffer must have been filled with M type A packets with higher priorities. Thus there is no "leakage" problem.

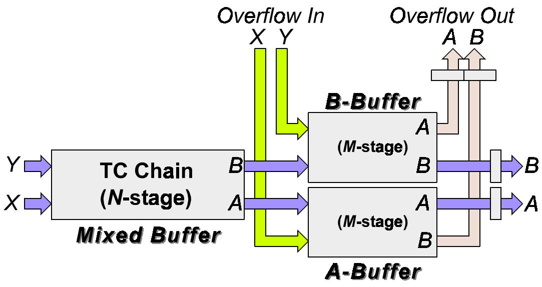


Fig. 5. TC layer.

Observation. If the minimum sojourn time is large, i.e., overflow layers are slow, then the size of A/B-buffers can be adjusted to compensate for it. In fact, we just need to increase M to match the speed of the slow overflow layers. Thus it is conceivable to use a *heterogeneous storage architecture*, in which overflow layers are implemented with a completely different architecture: e.g., an SRAM-based architecture.

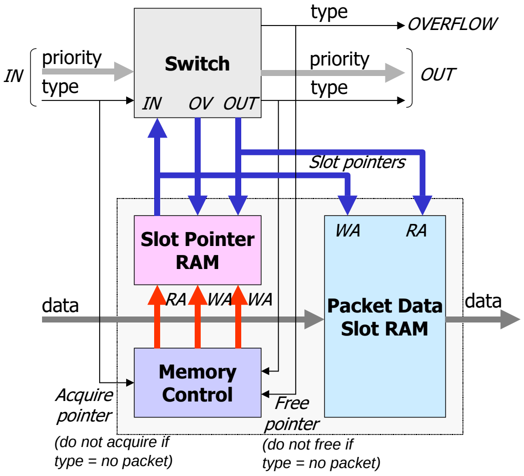


Fig. 6. Packet pointer switching system architecture.

E. System Architecture Using Packet Pointers

So far, we have discussed the sorting of packets within our network. Although it is technically feasible to sort the entire packets within the network, the cost for doing so may be prohibitive. In order to reduce the cost, we can store packets in an external memory, and sort pointers to fixed-size packet "slots" as shown in Fig. 6, instead of storing and sorting the entire packets in the sorting network. Slots are suitable for ATM cells or for fragments of larger variable length packets. One possible implementation uses a circular list of pointers to free packet slots. As depicted in Fig. 7, slots are assigned to new packets from the head of the list, and pointers to freed slots are appended to the tail of the list following packet transmission. Clearly, both allocating and returning a slot are $\mathcal{O}(1)$ operations. Note that the slots in use are not linked in any way: the order in which slots are freed (packets transmitted or lost) is determined entirely by the order in which slot pointers (heretofore "packets") exit the switch.

A typical switch operation proceeds as follows:

1. Two new packets arrive at the switch.
2. Pointers p_u and p_v, which point to memory slots m_u and m_v, are read from the free-slot list.
3. The new packets are written into memory slots m_u and m_v. Concurrently, the priorities and types of the packets and the pointers (p_u and p_v) are pushed into the switch.
4. Pointer p_x, which corresponds to the highest priority type A packet, is outputted on port A, and pointer p_y, which corresponds to the highest priority type B packet, is outputted on port B.
5. Packets from memory slots m_x and m_y are transmitted. p_x and p_y are written at the tail of the free-slot list.

Because of the large number of memory accesses, it is prudent to separate the packet memory from the pointer memory. In any cycle, there may be up to two writes to and two reads from the packet memory, which implies an additional two reads and two writes for the list maintenance. Furthermore, it is possible to have a packet overflowing from the switch, and the slot for this overflow packet must also be reclaimed. Therefore, up to two reads and three writes are required for the list maintenance in any cycle.

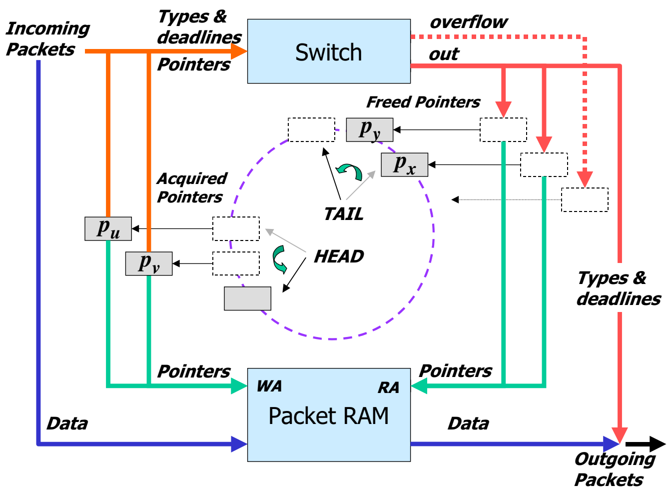


Fig. 7. Packet pointer list maintenance.

III. Implementation

Our implementation of the TC chain is a self-timed, self-precharging domino logic design. Self-timed, self-precharging design avoids the need for a precise clock distribution network and allows "zero-overhead" evaluation [11]. Control logic is simple and efficient as well.

A. Background Review: Zero-Overhead Domino Logic

Fig. 8 shows a domino multiplexor circuit (used in our datapath). Its operation consists of alternating precharge and evaluation phases. Before the logic can be evaluated for an input pattern, it must first be precharged, i.e., $\overline{prech}$ must be asserted (low), so that node X is precharged high and output Y driven low. After precharging, *eval* is asserted. Y is evaluated high, if a and s^{T} or b and s^{F} become high. Otherwise, Y remains low. Note that node X, once it is discharged, remains low until $\overline{prech}$ is re-asserted, because of the lack of a complementary pMOS stack. There are two implications: (1) inputs must be set up before *eval* is asserted or *monotonically rise*, i.e., remain low or rise once and not fall again until X is evaluated to an intended value; (2) once X is evaluated, inputs may fall (be precharged).

To ensure that inputs to all domino circuits rise monotonically during evaluation, we employ *dual-rail* logic for some computational blocks. Some logic functions, such as the multiplexor shown in Fig. 8, require both true and complemented forms of inputs. This means that the input monotonicity requirement described above cannot be satisfied if the logic is implemented *conventionally*: e.g., the complement of s is generated by inverting s. In dual-rail domino logic, however, outputs are encoded on two wires: one wire for logic '1' (*true* rail) and the other for logic '0' (*false* rail). Both wires are driven low during precharging and exactly one wire becomes high when the evaluation is completed. s^{T} and s^{F} in Fig. 8 represent true and false rails of s.

Domino circuits have significant speed advantages over static CMOS circuits because of the reduced fanin (the pMOS stack is replaced by a single precharge pMOS). Output inverters can also be biased for fast rising output transitions, which reduces the forward latency at the expense of longer precharge delay.

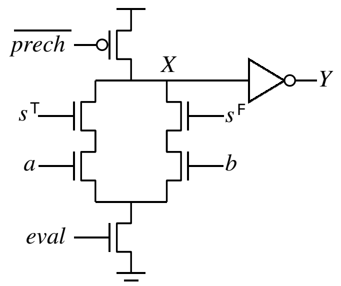


Fig. 8. A domino multiplexor: $Y = as^{\mathsf{T}} + bs^{\mathsf{F}}$.

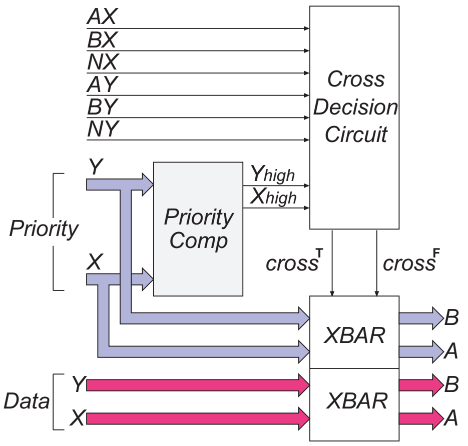
In our design, each crossbar (implemented as a domino circuit) starts in evaluation mode (precharged during power-up) and operates immediately on arriving packets. Its outputs trigger its successors to evaluate immediately. It then prepares itself for the next pair of packets based on the feedback from its predecessors and successors. That is, it precharges itself[1] when its outputs have been absorbed by its successors (latches and crossbars) and puts itself back in evaluation mode when one of its predecessors is precharged. Note that evaluation may be *enabled* well in advance of the arrival of the next pair of packets

[1] Our domino implementation style is called "self-precharging" because each domino stage is precharged based on the feedback from its successor stages without the aid of an external clock.

— the only constraint is that predecessors must be precharged, so that new inputs can be distinguished from old ones. As such, *no control overhead* delay is incurred upon arrival of new packets. This "zero-overhead" technique [11] substantially accelerates the flow of packets through the TC chain by hiding not only the controller latency, but also the delays to enable evaluation transistors.

B. Datapath

The TC layer datapath consists of three TC chains (*Mixed* buffer, A/B-buffers). Each TC stage consists of two crossbars and a feedback latch. The crossbars are further divided into decision units and data-routing "slices." Each decision unit consists of a dual-rail domino borrow-skip subtracter and a dual-rail cross decision circuit, as shown in Fig. 9. The decision to cross or not to cross is based on packet types and the result of priority comparison. When packets are of the same type, priorities are compared to determine the routing. When packets are of different types, each packet is immediately routed to its destined port. 16-bit unsigned priorities are used; therefore, 32 signals are required for dual-rail signaling. Packet types are one-hot encoded: 3 signals indicate type A, B or *no packet*. Each data slice routes 8 bits. Four data slices correspond to 32 bits of packet pointers. In contrast to the priority and type bits, no logical operations are performed on data bits; thus less costly single-rail domino logic suffices for the data crossbars.



AX	type A packet on X port
BX	type B packet on X port
NX	no packet on X port
AY	type A packet on Y port
BY	type B packet on Y port
NY	no packet on Y port

Fig. 9. Crossbar block diagram.

C. Control

The routing decision is made at the crossbar, as shown in Fig. 9, and propagated from one data slice to the next in a daisy-chained fashion to prevent broadcasting to all constituent data slices. The control flow from crossbar to crossbar through the TC chain is considerably more complex. An example detailing the process is described below.

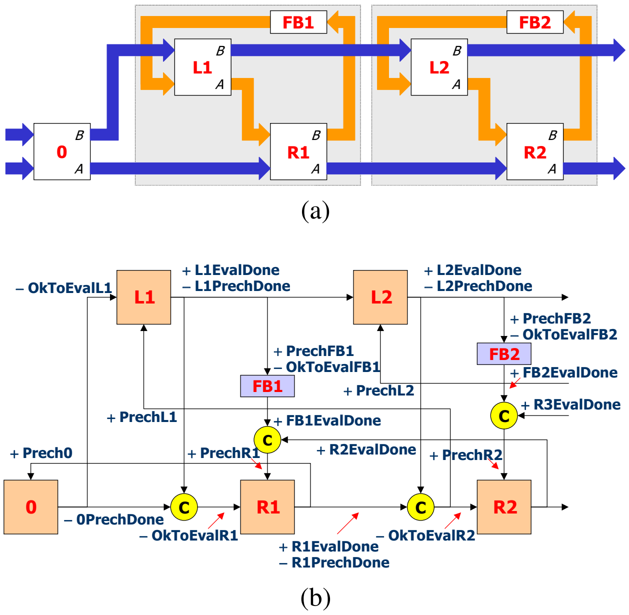


Fig. 10. A two-stage TC chain and its control: $+\mathsf{L1EvalDone}$ denotes that a rising transition on the wire that it labels indicates that L_1 has finished evaluating. $-\mathsf{OkToEvalL1}$ denotes that a falling transition on the wire that it labels indicates that it is "okay" to put L_1 in evaluation mode. Shaded circles labeled "C" are C-elements, whose outputs rise when both inputs rise, fall when both inputs fall, and maintain the same values otherwise.

C.1 Control Flow Example

Evaluation of datapath elements is done in a true *domino* fashion — the arrival of each new packet starts a domino effect, unencumbered by control. Control is only used for setting up the datapath elements for the next cycle, such as precharging crossbars and latching data in the feedback buffers. Each datapath element (except feedback latches) operates as follows:

- Start in evaluation mode;
- Evaluate as soon as the data from its predecessors arrive;
- Precharge only after all successors have finished evaluation;
- Set up for the next evaluation only after one of its predecessors has finished precharging.

Consider stage 1 of the two-stage TC chain shown in Fig. 10. The chain, as depicted in Fig. 11, operates as follows:

1. when L_1 receives a packet (data, priority, and type information), it starts evaluating immediately.
2. when L_1 is done evaluating, R_1 and L_2 immediately start evaluating. Since L_1 no longer requires its inputs, FB_1 starts precharging.
3. R_1 and L_2 have finished evaluating. L_1 starts precharging, because its successors are done.
4. L_1 has completed precharging, enabling FB_1 to evaluate. L_1 is ready for the next evaluation, assuming that its predecessor has already been precharged.
5. FB_1 is done evaluating. R_1 therefore starts precharging.
6. R_1 is precharged and enabled for the next evaluation.

Note that packets flow through the TC chain with a minimal synchronization delay — synchronization is needed only at R crossbars.

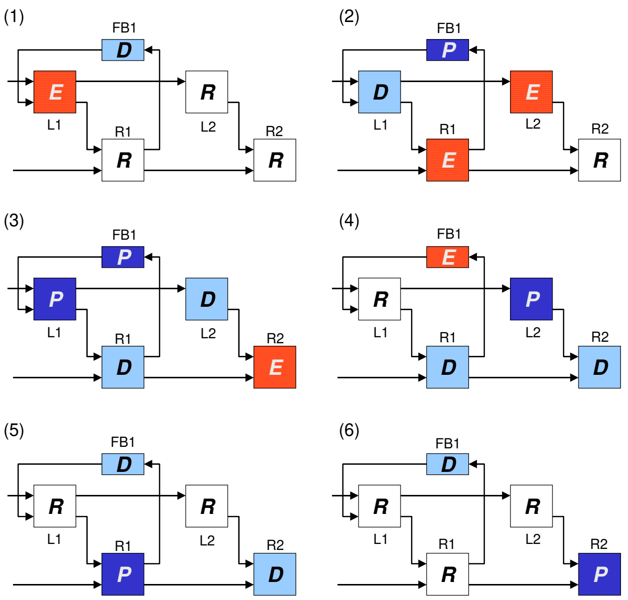


Fig. 11. Control flow example for a two-stage TC chain: R means "ready to evaluate," E means "evaluating," D means "done evaluating," and P means "precharging."

C.2 Extended Burst-Mode Controllers

The datapath control consists of a set of *extended burst-mode* (XBM) controllers [12], [13] and C-elements (used to AND two events). In Fig. 10(b), shaded boxes are XBM controllers and shaded circles are C-elements.

An extended burst-mode specification [12] consists of a finite number of states, a set of labeled state transitions connecting pairs of states, and a start state. Fig. 12(a) and Fig. 12(b) show the XBM specifications of the *crossbar control*, which has 3 inputs (SuccDone, PredDone, Done) and 2 outputs (Eval, Prech), and of the *feedback latch control*. Fig. 12(c) shows an implementation of the crossbar controller specified in Fig. 12(a).

Signals ending with + or − are *terminating* signals. For example, in the transition from state 0 to 1, all the inputs are terminating signals, which means that the state transition and the associated output transitions occur when all of the input transitions have occurred. That is, the controller lowers Eval and transitions to state 1 only after SuccDone has fallen and PredDone and Done have risen.

Signals ending with an asterisk, such as PredDone$*$ in state 1, are *directed don't cares*. If a state transition is labeled with $a*$, the following state transition in the specification must be labeled with $a*$ or with $a+$ or $a-$. A sequence of state transitions labeled with $a*$ followed by a transition labeled with $a+$ (or $a-$) means that a rises (or falls) *exactly once* somewhere during the sequence of state transitions. For example, $S_1 \to S_2$ is labeled with PredDone$*$, and $S_2 \to S_0$ with PredDone$-$. This means that PredDone is allowed to fall during $S_1 \to S_2$, but must have fallen before $S_2 \to S_0$ can occur. Inputs not mentioned in a state transition are not allowed to change during that state transition. For example, Done may not change in state 1.

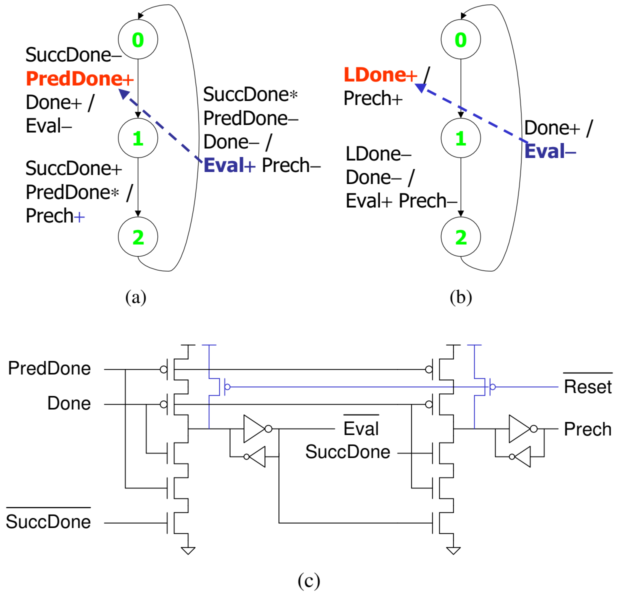


Fig. 12. (a) Crossbar control; (b) Feedback latch control; (c) Crossbar controller circuit.

C.3 Throughput Constraints

Recall that, during evaluation, the TC chain behaves like combinational logic. Precharging of crossbars and storing new data in the feedback buffers are done in the background, which poses no problem as long as the next packet arrives after a sufficient time. Here we consider the problem of how soon the next packet can arrive; i.e., the constraints on the throughput imposed by the time required for processing background tasks.

The crossbar must be in evaluation mode before new packets from its predecessors arrive, in order to prevent slowing down the input packets. This constraint coincides with one of the correctness constraints for the crossbar control FSM, shown as a dashed arrow in Fig. 12(a): i.e., Eval+ must precede PredDone+. This is a constraint on the maximum throughput.

Similarly, the feedback latch must be ready for precharging, i.e., it must have disabled evaluation, before its successor (L crossbar) completes its evaluation. The reason for this is that L crossbar prompts the feedback latch to begin precharging when it finishes evaluating. This constraint also coincides with one of the correctness constraints for the feedback latch control FSM, shown as a dashed arrow in Fig. 12(b), i.e., Eval$-$ must precede LDone+. This is another constraint on the maximum throughput.

D. Implementing Long TC Chains

The total storage capacity of a switch is the product of the capacity of each TC layer and the number of layers. The latency through each stage limits the maximum number of TC stages a packet can traverse[2] in a single layer, which, in turn, dictates the storage capacity of a layer. Although the storage can be increased by adding layers, there is an overhead for doing so,

[2]For clarity of exposition, we treat the flow of packets in the TC chain as though a single packet traversed all the way through the entire chain in every clock cycle. In reality, however, any packet may interact with a stored packet in a buffer and "bump" the stored packet down the chain in its place.

in terms of buffer efficiency. Because A/B-buffers can hold only one type of packets, the buffer efficiency of a multi-layer implementation is reduced as the capacity of the *mixed*-buffer becomes smaller. Thus it is more efficient to implement fewer higher capacity layers.

So far, we have rigidly adhered to the transmission ordering requirement: "The packets with the highest priority among all the packets in the switch at time $t + \delta$ $(0 < \delta < 1)$ shall be transmitted at time $t + 1$." If this requirement is relaxed slightly, i.e., the packets to be transmitted at time $t + n$ $(n > 1)$ are selected from a pool of packets in the switch at time $t + \delta$ *minus* the packets already selected for transmission at times $t + 1, \ldots, t + n - 1$, where n is the number of pipe stages, then we can extend the lengths of TC chains by implementing a multi-cycle pipeline. With this change in the transmission ordering requirement, pipelined implementations may transmit packets with lower priorities at time $t + n$ than some packets that have entered the switch after time t. However, as the layers are unlikely to be extended to incur a latency of more than a few clock cycles in this manner, this modification involves only a minor concession.

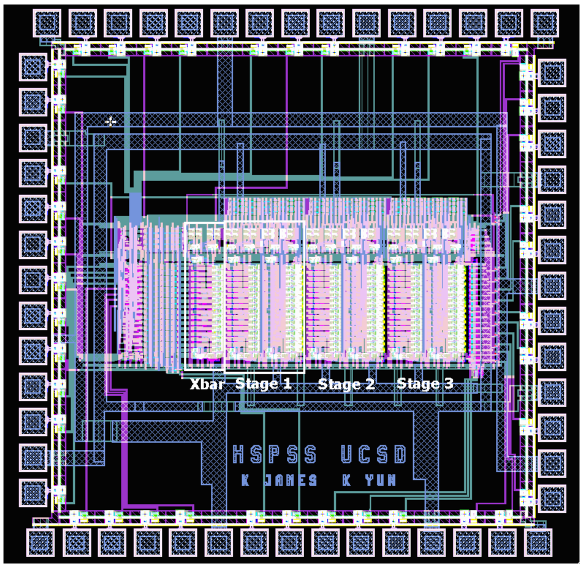


Fig. 13. 3-stage TC chain layout: 4-metal $0.35\mu m$ HP CMOS process.

IV. Practical Example: ATM Packet Switch

One application of the sorting network is an ATM packet switch supporting *deadline-based quality of service* (QoS). A chip design, which demonstrates the *core* functionality described in the previous sections, has been completed. The output packet sequencing is prioritized based on the deadline stamped on the packet. The switch is designed to support 10 Gb/s *per port*, i.e., a bit time of 0.1 ns. Since an ATM cell (packet) contains 53 bytes (424 bits), the resulting packet time is 42.4 ns. A full-custom layout, shown in Fig. 13, has been completed using a 4-metal 0.35μm ($\lambda = 0.2\mu$m) HP CMOS process. The test chip is organized as a single-layer, three-stage TC-chain with 8-bit data and 16-bit deadlines.

Simulation results are based on the backannotated schematic

Crossbar Latency	Subtract	Decide	Route
2.8ns (worst)	1.6ns	0.4ns	0.8ns
1.3ns (best)	–	0.5ns	0.8ns
Chip Latency	Input	Core	Output
18.3ns	1.1ns	13.7ns	3.5ns
Internal Cycle	Eval	Latch	Prech
13.1ns	2.8ns	5.4ns	4.9ns

TABLE I
SIMULATION RESULTS.

that incorporates parasitic capacitances extracted from the layout, and are summarized in Table I. The worst-case latency of the crossbar is 2.8 ns (1.6 ns for the worst-case subtraction delay, 0.4 ns for the cross decision circuit, and 0.8 ns for the data multiplexor and the buffer). The best-case latency comes from the cases that require only the type comparison, not the priority comparison.[3]

The worst-case latency through an N-stage TC chain is the sum of the latencies of $N + 2$ crossbars. As can be generalized from Fig. 3, the worst-case delay path is the path: $0 \rightarrow \mathsf{L}_1 \rightarrow \cdots \rightarrow \mathsf{L}_i \rightarrow \mathsf{R}_i \rightarrow \cdots \rightarrow \mathsf{R}_N$. Since the worst-case latency through a crossbar is 2.8 ns, there is enough time for 13 crossbar delays in 42.4 ns, with 4.6 ns reserved for input and output pads and other housekeeping functions. Therefore, as many as $N = 11$ TC Stages could be placed in a single TC Layer (for multiple layers, $N + M + 4 = 13$ or $N + M = 9$). Of course, the TC chain can be implemented to have zero control overhead using the *four-phase overlapped clocking* scheme devised by Harris and Horowitz [14]. However, the implementation of feedback latches would make *strictly* synchronous solution challenging.

The internal cycle time of 13.1 ns is measured from the start of an evaluation to the enabling of the subsequent evaluation. It is clear that the internal stage cycle time is not a limiting factor for a 10 Gb/s operation (a packet time = 42.4ns). Note that a substantial fraction (5.4ns) of the cycle time is due to the feedback latch. Before the R crossbar can complete its cycle (i.e., proceed to precharge), its output must be stored in the latch. However, before the (dynamic) latch can store the output of the R crossbar, it must be put in evaluation mode, which cannot occur until the L crossbar has completed precharging. Finally, the precharging delay (4.9ns) is substantially longer than the evaluation delay (2.8ns) because our domino circuits are biased for forward latency optimization.

In order to satisfy the throughput constraint depicted in Fig. 12(ab) and discussed in Section III-C.3, we need to make sure that all the crossbars are back in evaluation mode before new packets arrive. The last crossbar in the TC chain would be affected the most by this constraint, because it cannot be precharged until its output leaves the switch (or latched in the adjoining pipeline register). Therefore, the minimum delay through the first $N - 1$ TC stages (or N crossbars) must be greater than the internal cycle time minus the maximum evaluation delay (which would be the maximum additional time re-

[3] The decision time for the best case is slightly longer than for the worst-case because the circuit was designed to optimize for the worst-case.

quired to put the crossbar back in evaluation-mode after the end of the packet cycle). Clearly, this is not a problem at all since $11 \times 1.3\text{ns} >> 12.3\text{ns} - 2.8\text{ns}$.

Constructing the sorting network requires the sorting chip itself, as well as external memories and control, as discussed above. We now examine the viability of the packet pointer architecture operating at 10 Gb/s using commercially available bulk CMOS memory. Commercial synchronous SRAM's are available in speeds up to 250MHz. As depicted in Fig. 14, two read and three write operations can be performed in six clock cycles. Therefore, 150MHz SRAM's can be used to maintain the free slot list and to store packets, assuming 53-byte ports. Therefore, an implementation consisting of off-the-shelf SRAM's and a simple VLSI memory controller is definitely feasible.

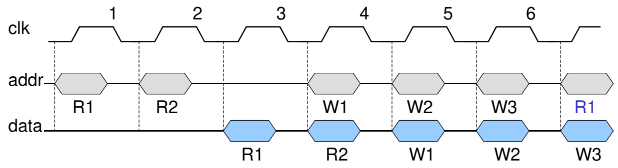


Fig. 14. Generic late-write SRAM timing.

The sorting network, of course, forms only a part of the complete 2×2 switch. The deadline assignment is arguably the most significant external function. In fact, the assignment of a suitable priority (or deadline for EDF scheduling) to each packet is *crucial* in the pursuit of meaningful QoS guarantees. It can be accomplished using a general approach, such as using *service curves* [1] or using a simpler approach outlined below.

The deadline of a packet is defined as the sum of its intrinsic priority (the higher the priority, the smaller the number) and a constantly advancing (and cyclic) baseline: $Deadline = t_s + t_\delta$, where t_s is the baseline and t_δ is the allowable delay based on priority. The baseline portion insures that the deadline increases as time elapses, so that not all similar priority packets are assigned similar deadlines over time. For example, old high-priority packets should have precedence over new high-priority packets. This style of deadline assignment also prevents starvation of low-priority traffic, because old low-priority packets *eventually* gain precedence over new packets.

Finally, we can form larger switches, e.g., 8×8 switch [15], [16], using 2×2 switches. Typically, these are configured as multi-stage distributed-buffer networks: e.g., Omega, Banyan, Flip, and Baseline [15].

V. Conclusion

This paper addressed key issues in practical packet switched networks: high throughput with low latency, priority-based sorting, and extensible architecture yielding large storage capacity. We described the basic architecture based on TC stages and suggested how slower, cheaper sorting networks can be used to extend the buffer capacity of our network, without sacrificing the sorting latency. We described our implementation of the sorting network to be used in an ATM switching core. We demonstrated our zero-overhead, self-timed, self-precharging domino logic design that achieves zero control overhead. Finally, we are investigating a hierarchical storage architecture to extend our current design.

Acknowledgment

We would like to thank Jason Hernandez of IBM, Austin Research Lab. for his comments on the use of pointers to increase buffer capacity.

References

[1] R. L. Cruz, "Quality of service guarantees in virtual circuit switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1048–1056, Aug. 1995.

[2] G. Chen and I. Stavrakakis, "ATM traffic management with diversified loss and delay requirements," in *Proceedings of IEEE INFOCOM*, 1996, pp. 1037–1044.

[3] R. L. Cruz and J. T. Tsai, "COD: Alternative architectures for high speed packet switching," *IEEE/ACM Transactions on Networking*, vol. 4, no. 1, pp. 11–21, Feb. 1996.

[4] R. Kannan, K. Y. Lee, and H. F. Jordan, "A high-speed hardware efficient optical sorting network," in *Proceedings of Asia-Pacific Conference on Communications*, 1995, pp. 873–877.

[5] M. R. Hashemi and A. Leon-Garcia, "The single-queue switch: a building block for switches with programmable scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 5, pp. 785–794, June 1997.

[6] S. W. Moore and B. T. Graham, "Tagged up/down sorter – A hardware priority queue," *The Computer Journal*, vol. 38, no. 9, pp. 695–703, Sept. 1995.

[7] A. R. Krappel, "CAN (Controller Area Network) controller for real-time use," *Elektronik*, vol. 46, no. 10, pp. 88–92, May 1997.

[8] S. E. Butner and R. Chivukula, "On the limits of ATM switching," *IEEE Network Magazine*, vol. 10, no. 6, pp. 26–31, November/December 1996.

[9] K. L. E. Law and A. Leon-Garcia, "A large scalable ATM multicast switch," *IEEE Journal on Selected Areas in Communication*, vol. 15, no. 5, pp. 844–854, June 1997.

[10] R. Palmer, "An experimental ATM switch for BISDN studies," *International Journal of Digital and Analog Communication*, vol. 3, no. 4, pp. 341–349, 1990.

[11] T. E. Williams, "Latency and throughput tradeoffs in self-timed asynchronous pipelines and rings," Tech. Rep. CSL-TR-90-431, Stanford University, Aug. 1990.

[12] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: part I (specification and hazard-free implementations)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 18, no. 2, pp. 101–117, Feb. 1999.

[13] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: part II (automatic synthesis)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 18, no. 2, pp. 118–132, Feb. 1999.

[14] D. Harris and M. A. Horowitz, "Skew-tolerant domino circuits," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 11, pp. 1702–1711, Nov. 1997.

[15] J. Walrand and P. Varaiya, *High Performance Communication Networks*, Morgan Kaufmann Publishers, Inc., 1996.

[16] J. Turner and N. Yamanaka, "Architectural choices in large scale ATM switches," Report WUCS 97-21, Washington University, Department of Computer Science, May 1997.

Kenneth Y. Yun (Member, IEEE) received the Ph.D. degree in Electrical Engineering from Stanford University, Stanford, CA and the M.S. degree in Electrical Engineering and Computer Science from Massachusetts Institute of Technology, Cambridge. He is currently an assistant professor in the Electrical and Computer Engineering Department at University of California, San Diego. He had held design engineering positions at TRW and Hitachi for 6 years. His current research interests include high-speed networking, asynchronous circuits and systems, and synchronization issues. He has been working with Intel Corp. as a primary consultant on the Asynchronous Instruction Decoder Project.

Dr. Yun has organized the Fourth International Conference on Advanced Research in Asynchronous Circuits and Systems as a program co-chair. He is a recipient of the 1996-99 National Science Foundation CAREER award and

1996 Hellman Faculty Fellowship, and a co-recipient of the Charles E. Molnar award for a paper that best bridges theory and practice of asynchronous circuits and systems. He has received a best paper award at the 1998 International Conference on Computer Design.

K. W. James (Student Member, IEEE) received the B.S. degree in Electrical Engineering from University of California at Berkeley in 1993. From 1993 to 1996, he was with Advanced Micro Devices, Sunnyvale, CA. In 1997 he received the M.S. degree in Electrical Engineering from University of California, San Diego, where he is now a Ph.D. candidate. His current research focus is on high-speed packet switching architectures that support diverse quality-of-service guarantees in heterogeneous traffic environments such as the Internet.

Robert H. Fairlie-Cuninghame received the B.S. degree in Computer Science and Applied Mathematics in 1996 and the B.E.(Hons) degree in Electrical Engineering (Information Systems Engineering) in 1998 from the University of Sydney, Australia. In 1997, he wrote an honors thesis at the University of California, San Diego with Prof. R. L. Cruz as the advisor; the thesis investigated innovative high-speed sorting switch designs. He is currently working as a consultant to a San Diego based Internet telephony company. He has received a Best Paper Award at the International Conference on Computer Design in 1998.

Supratik Chakraborty (Member, IEEE) received the B. Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Kharagpur in 1993, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA in 1995 and 1998, respectively. He is currently a member of the Advanced CAD Research group at Fujitsu Laboratories of America, Inc., Sunnyvale, CA. His research interests include the design, analysis, optimization and verification of digital systems, with emphasis on asynchronous systems and fast timing analysis algorithms. Dr. Chakraborty has received the President of India Gold Medal for best academic performance at Indian Institute of Technology, Kharagpur in 1993. He has also received a Best Paper Award at the International Conference on Computer Design in 1998.

Rene L. Cruz (Senior Member, IEEE) received the B.S. and Ph.D. degrees from the University of Illinois, Urbana, and an S.M. degree from the Massachusetts Institute of Technology, all in Electrical Engineering. He is currently a professor of electrical and computer engineering at the University of California, San Diego.