

(19) World Intellectual Property Organization
International Bureau



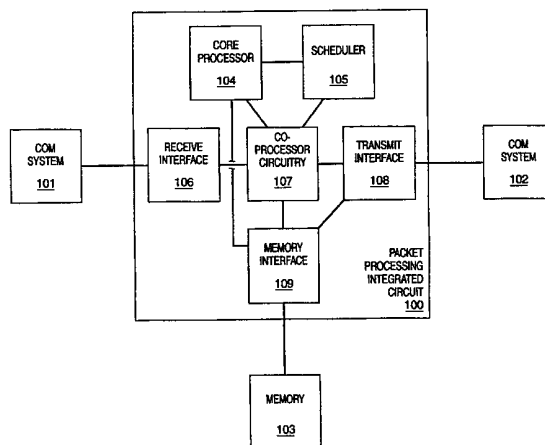
(43) International Publication Date
22 February 2001 (22.02.2001)

PCT

(10) International Publication Number
WO 01/13590 A1

- (51) International Patent Classification⁷: **H04L 12/56, H04Q 11/04**
- (74) Agents: **SETTER, Michael, J.**; Duft, Graziano & Forest, P.C., P.O. Box 270930, Louisville, CO 80027 et al. (US).
- (21) International Application Number: **PCT/US00/22431**
- (81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.
- (22) International Filing Date: 16 August 2000 (16.08.2000)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
60/149,376 17 August 1999 (17.08.1999) US
- (84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).
- (71) Applicant: **CONEXANT SYSTEMS, INC.** [US/US];
4311 Jamboree Road, Newport Beach, CA 92660 (US).
- (72) Inventors: **LUSSIER, Daniel, J.**; 10 Harness Lane, Holliston, MA 01746 (US). **TOMPKINS, Joseph, B.**; 35 Georgetown Drive, Apt. 12, Framingham, MA 01702 (US). **SNYDER, Wilson, P., II**; 127 White Pond Road, Hudson, MA 01749 (US).
- Published:**
— With international search report.
- For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: INTEGRATED CIRCUIT WITH A CORE PROCESSOR AND A CO-PROCESSOR TO PROVIDE TRAFFIC STREAM PROCESSING



(57) Abstract: An integrated circuit (100) comprises co-processor circuitry (107) and a core processor (104). The co-processor circuitry (107) comprises context buffers (315) and data buffers (316). The co-processor circuitry (107) receives and stores one of the communication packets in one of the data buffers (314). The co-processor circuitry (107) correlates the one communication packet with one of a plurality of channel descriptors. The co-processor circuitry (107) associates the one data buffer with one of the context buffers (315) holding the one channel descriptor to maintain the correlation between the one communication packet and the one channel descriptor. The co-processor circuitry (107) determines a prioritized processing order for the communication packet relative to the other communication packets. The core processor (104) executes a packet processing software application that directs the core processor (104) to process the one communication packet in the one data buffer based on the one channel descriptor in the one context buffer and based on the prioritized processing order. The co-processor circuitry (107) is configured to operate in parallel with the core processor (104).

INTEGRATED CIRCUIT WITH A CORE PROCESSOR AND A CO-PROCESSOR TO PROVIDE TRAFFIC STREAM PROCESSING

5

Related Cases

This application is related to United States provisional patent application 60/149,376, entitled "HIGH SPEED COMMUNICATIONS PROCESSING DEVICE FOR INTERNET PROTOCOL, ASYNCHRONOUS TRANSFER MODE, FRAME RELAY, AND SONET COMMUNICATIONS", filed on August 17, 1999, and which is
10 hereby incorporated by reference.

Background of the Invention

1. Field of the Invention

The invention is related to the field of communications, and in particular, to
15 integrated circuits that process communication packets.

2. Statement of the Problem

Communications systems transfer information in packet streams. The packets in the streams each contain a header and a payload. The header contains
20 control information, such as addressing or channel information, that indicate how the packet should be handled. The payload contains the information that is being transferred. Some packets are broken into segments for processing. The term "packet" is intended to include packet segments. Some examples of packets include, Asynchronous Transfer Mode (ATM) cells, Internet Protocol (IP) packets,
25 frame relay packets, Ethernet packets, or some other packet-like information block.

An integrated circuit known as a stream processor has been developed recently to address the special needs of packet communication networking. Traffic stream processors are designed to apply robust functionality to extremely high-speed packet streams. This dual design requirement is often in conflict because the high-speeds limit the level of functionality that can be applied to the packet stream.

Robust functionality is critical with today's diverse but converging communication systems. Stream processors must handle multiple protocols and interwork between streams of different protocols. Stream processors must also ensure that quality-of-service constraints are met with respect to bandwidth and priority. Each stream should receive the bandwidth allocation and priority that is defined in corresponding service level agreements. This functionality must be applied differently to different streams – possibly thousands of different streams.

To provide such functionality, a RISC-based core processor was developed with its own network-oriented instruction set. The instruction set is designed to accomplish common networking tasks in the fewest cycles. The core processor executes software applications built from the instruction set to apply the robust functionality to high-speed packet streams.

To determine how the packets should be handled, the core processor constantly correlates packets with their respective context information. The core processor then retrieves the correlated context information from off-chip memory. Typically, expensive off-chip memory systems are required to speed up this process. When handling multiple packets, the core processor must maintain the coherency of the context information as the context information changes in a pipe-lined processing environment. Given the high-speeds of the packet streams, this function places a heavy burden on the core processor and expends critical processing capacity.

Instead of providing additional quality-of-service processing, the core processor spends time correlating packets with context information, retrieving the correlated context information, and maintaining the coherency of the context information.

Unfortunately, the core processor must constantly determine which packet
5 stream to handle next. Given the high-speeds of the packet streams, this function places a heavy burden on the core processor and expends critical processing capacity. Instead of providing additional quality-of-service processing, the core processor spends time deciding which packets to process.

The buffers that store the communication packets externally to the stream
10 processor integrated circuit are segregated into multiple classes. To avoid fragmentation where packets undesirably cross external buffer boundaries, each external buffer is sized to hold a single packet. In high-speed systems, this requires thousands of external buffers that are separated into several different classes.

To store a packet in an external memory, the core processor must first
15 allocate an external buffer in the memory. In addition to allocation, the core processor must manage buffer conditions, such as buffer exhaustion. This buffer allocation and management consumes bandwidth between the stream processor and the external memory. These tasks are dramatically increased when they are applied differently across multiple buffer classes. Given the high-speeds of the
20 packet streams, these tasks place a heavy burden on the core processor and expend critical processing capacity. They also consume significant bandwidth that may require additional pins or silicon.

A primary task of the core processor is managing a packet transmission schedule. The schedule must attempt to maintain various bandwidth guarantees
25 across multiple streams of traffic. For example, real time traffic is a higher priority

than non-real time traffic, and e-mail traffic is a higher priority than system back-up traffic. Unfortunately, scheduling of this complex nature requires significant core processing capacity. This use of processing capacity for prioritized scheduling diminishes the level of functionality provided by the stream processor.

5 Scheduling is often implemented through a complex prioritization scheme and requires the execution of scheduling algorithms, such as a guaranteed cell rate algorithm. To assist the core processor, scheduling circuitry has been developed. Unfortunately, such circuitry is inadequate because it is not effectively programmable from a cache memory that stores scheduling parameters for each given packet. This
10 scheduling circuitry is too static and non-responsive with respect to complex scheduling tasks.

Summary of the Solution

 The invention helps solve the above problems with an integrated circuit that
15 performs as a traffic stream processor. The integrated circuit uses co-processor circuitry to correlate packets with respective context information and transfer the correlated context information to an on-chip context buffer. Cheaper off-chip memory systems can be used for context information because the latency of context retrieval is hidden from the core processor. The co-processor circuitry maintains the
20 coherency of the context information by preventing multiple copies from existing.

 The co-processor circuitry also establishes a prioritized work queue for the core processor. The core processor can simply process packets from the work queue and avoid expending capacity to determine a prioritized processing order. The capacity savings provided by the co-processor can be used by the core
25 processor to handle higher-speed streams or increase the available functionality.

The integrated circuit has a buffer management engine that relieves the core processor of significant external buffer management. The buffer management engine also reduces the bandwidth between the integrated circuit and external memory that is required for external buffer management. Advantageously, the buffer management engine allows the use of relatively small packet-sized external buffers if desired.

The integrated circuit has scheduling circuitry. The scheduling circuitry executes scheduling algorithms to schedule packet transmissions. Advantageously, cached scheduling parameters that are retrieved from a context buffer for the packet are used in the algorithm. Thus, the scheduling circuitry is highly programmable, and also conserves important core processor capacity for other services.

The integrated circuit comprises co-processor circuitry and a core processor. The co-processor circuitry comprises context buffers and data buffers. The co-processor circuitry receives and stores one of the communication packets in one of the data buffers. The co-processor circuitry correlates the one communication packet with one of a plurality of channel descriptors. The co-processor circuitry associates the one data buffer with one of the context buffers holding the one channel descriptor to maintain the correlation between the one communication packet and the one channel descriptor. The co-processor circuitry determines a prioritized processing order for the communication packet relative to the other communication packets. The core processor executes a packet processing software application that directs the processor to process the one communication packet in the one data buffer based on the one channel descriptor in the one context buffer and based on the prioritized processing order. The co-processor circuitry is configured to operate in parallel with the core processor

In some examples of the invention, the co-processor circuitry prevents multiple valid copies of the one channel descriptor from existing in the context buffers by tracking a number of the data buffers associated with the one context buffer. In some examples of the invention, the co-processor circuitry determines the prioritized processing order based on an availability of resources required by the
5 core processor to process the communication packet.

In some examples of the invention, the integrated circuit further comprises a pointer cache and control logic. The pointer cache stores pointers that correspond to external buffers. The external buffers store the communication packets externally to
10 the integrated circuit. The external buffers and the pointers to the external buffers are distributed among a plurality of classes. The control logic allocates the external buffers as the corresponding pointers are read from the pointer cache. The control logic de-allocates the external buffers as the corresponding pointers are written back to the pointer cache. The control logic tracks a number of the pointers to the de-
15 allocated external buffers for at least one of the classes.

In some examples of the invention, the integrated circuit further comprises scheduling circuitry. The scheduling circuitry comprises multiple scheduling boards wherein at least some of the scheduling boards have multiple priority levels. The scheduling circuitry retrieves scheduling parameters cached in the one context buffer
20 and executes an algorithm based on the scheduling parameters to schedule subsequent transmission of the communication packet. The scheduling circuitry processes one of the scheduling boards to schedule and subsequently initiate transmission of the communication packet.

The same reference number represents the same element on all drawings.

FIG. 1 is a block diagram that illustrates a packet processing integrated circuit in an example of the invention.

FIG. 2 is a block diagram that illustrates packet processing stages and pipe-
5 lining in an example of the invention.

FIG. 3 is a block diagram that illustrates co-processor circuitry in an example of the invention.

FIG. 4 is a flow diagram that illustrates buffer correlation and in-use counts in an example of the invention.

10 FIG. 5 is a block diagram that illustrates buffer management circuitry in an example of the invention.

FIG. 6 is a table that illustrates buffer classes in an example of the invention.

FIG. 7 is a block diagram that illustrates scheduler circuitry in an example of the invention.

15 FIG. 8 is block diagram that illustrates a scheduling board in an example of the invention.

FIG. 9 is block diagram that illustrates a scheduling board in an example of the invention.

20 Detailed Description of the Invention

Packet Processing Integrated Circuit -- FIG. 1

FIG. 1 depicts a specific example of an integrated circuit in accord with the present invention. Those skilled in the art will appreciate numerous variations from this example that do not depart from the scope of the invention. Those skilled in the
25 art will also appreciate that various features could be combined to form multiple

variations of the invention. Those skilled in the art will appreciate that some conventional aspects of FIG. 1 have been simplified or omitted for clarity. Various aspects of packet processing integrated circuits are discussed in United States patent 5,748,630, entitled "ASYNCHRONOUS TRANSFER MODE CELL
5 PROCESSING WITH LOAD MULTIPLE INSTRUCTION AND MEMORY WRITE-BACK", filed May 9, 1996, and which is hereby incorporated by reference into this application.

FIG. 1 is a block diagram that illustrates packet processing integrated circuit 100 in an example of the invention. Integrated circuit 100 comprises core processor
10 104, scheduler 105, receive interface 106, co-processor circuitry 107, transmit interface 108, and memory interface 109. These components may be interconnected through a memory crossbar or some other type of internal interface. Receive interface 106 is coupled to communication system 101. Transmit interface 108 is coupled to communication system 102. Memory interface is coupled to
15 memory 103.

Communication system 101 could be any device that supplies communication packets with one example being the switching fabric in an Asynchronous Transfer Mode (ATM) switch. Communication system 101 could be any device that receives communication packets with one example being the physical line interface in the
20 ATM switch. Memory 103 could be any memory device with one example being Random Access Memory (RAM) integrated circuits. Receive interface 106 could be any circuitry configured to receive packets with some examples including UTOPIA interfaces or Peripheral Component Interconnect (PCI) interfaces. Transmit interface 108 could be any circuitry configured to transfer packets with some
25 examples including UTOPIA interfaces or PCI interfaces.

Core processor 104 is a micro-processor that executes networking application software. Core-processor 104 supports an instruction set that has been tuned for networking operations – especially context switching. In some examples of the invention, core processor 104 has the following characteristics: 132 MHz, pipelined
5 single-cycle operation, RISC-based design, 32-bit instruction and register set, 4K instruction cache, 8 KB zero-latency scratchpad memory, interrupt/trap/halt support, and C compiler readiness.

Scheduler 105 comprises circuitry configured to schedule and initiate packet processing that typically results in packet transmissions from integrated circuit 100,
10 although scheduler 105 may also schedule and initiate other activities. Scheduler 105 schedules upcoming events, and as time passes, selects scheduled events for processing and re-schedules unprocessed events. Scheduler 105 transfers processing requests for selected events to co-processor circuitry 107. Scheduler 105 can handle multiple independent schedules to provide prioritized scheduling
15 across multiple traffic streams. To provide scheduling, scheduler 105 may execute a guaranteed cell rate algorithm to implement a leaky bucket or a token bucket scheduling system. The guaranteed cell rate algorithm is implemented through a cache that holds algorithm parameters. Scheduler 105 is described in detail with respect to FIGS 7-9.

20 Co-processor circuitry 107 receives communication packets from receive interface 106 and memory interface 109 and stores the packets in internal data buffers. Co-processor circuitry 107 correlates each packet to context information describing how the packet should be handled. Co-processor circuitry 107 stores the correlated context information in internal context buffers and associates individual
25 data buffers with individual context buffers to maintain the correlation between

individual packets and context information. Importantly, co-processor circuitry 107 ensures that only one copy of the correlated context information is present the context buffers to maintain coherency. Multiple data buffers are associated with a single context buffer to maintain the correlation between the multiple packets and the
5 single copy the context information.

Co-processor circuitry 107 also determines a prioritized processing order for core processor 104. The prioritized processing order controls the sequence in which core processor 104 handles the communication packets. The prioritized processing order is typically based on the availability of all of the resources and information that
10 are required by core processor 104 to process a given communication packet. Resource state bits are set when resources become available, so co-processor circuitry 107 may determine when all of these resources are available by processing the resource state bits. If desired, the prioritized processing order may be based on information in packet handling requests. Co-processor circuitry 107 selects
15 scheduling algorithms based on an internal scheduling state bits and uses the selected scheduling algorithms to determine the prioritized processing order. The algorithms could be round robin, service-to-completion, weighted fair queuing, simple fairness, first-come first-serve, allocation through priority promotion, software override, or some other arbitration scheme. Thus, the prioritization technique used
20 by co-processor circuitry 107 is externally controllable. Co-processor circuitry 107 is described in more detail with respect to FIGS. 2-4.

Memory interface 109 comprises circuitry configured to exchange packets with external buffers in memory 103. Memory interface 109 maintains a pointer cache that holds pointers to the external buffers. Memory interface 109 allocates the
25 external buffers when entities, such as core processor 104 or co-processor circuitry

107, read pointers from the pointer cache. Memory interface 109 de-allocates the external buffers when the entities write the pointers to the pointer cache.

Advantageously, external buffer allocation and de-allocation is available through an on-chip cache read/write. Memory interface 109 also manages various external
5 buffer classes, and handles conditions such as external buffer exhaustion. Memory interface 109 is described in detail with respect to FIGS 5-6.

In operation, receive interface 106 receives new packets from communication system 101, and scheduler 105 initiates transmissions of previously received packets that are typically stored in memory 103. To initiate packet handling, receive
10 interface 106 and scheduler 105 transfer requests to co-processor circuitry 107. Under software control, core processor 104 may also request packet handling from co-processor circuitry 107. Co-processor circuitry 107 fields the requests, correlates the packets with their respective context information, and creates a prioritized work queue for core processor 104. Core processor 104 processes the packets and
15 context information in order from the prioritized work queue. Advantageously, co-processor circuitry 107 operates in parallel with core processor 104 to offload the context correlation and prioritization tasks to conserve important core processing capacity.

In response to packet handling, core processor 104 typically initiates packet
20 transfers to either memory 103 or communication system 102. If the packet is transferred to memory 103, then core processor instructs scheduler 105 to schedule and initiate future packet transmission or processing. Advantageously, scheduler 105 operates in parallel with core processor 104 to offload scheduling tasks and conserve important core processing capacity.

Various data paths are used in response to core processor 104 packet transfer instructions. Co-processor circuitry 107 transfers packets directly to communication system 102 through transmit interface 108. Co-processor circuitry 107 transfers packets to memory 103 through memory interface 109 with an on-chip pointer cache. Memory interface 109 transfers packets from memory 103 to communication system 102 through transmit interface 108. Co-processor circuitry 107 transfers context information from a context buffer through memory interface 109 to memory 103 if there are no packets in the data buffers that are correlated with the context information in the context buffer. Advantageously, memory interface 109 operates in parallel with core processor 104 to offload external memory management tasks and conserve important core processing capacity.

Co-processor Circuitry -- FIGS. 2-4

FIGS. 2-4 depict a specific example of co-processor circuitry in accord with the present invention. Those skilled in the art will appreciate numerous variations from this example that do not depart from the scope of the invention. Those skilled in the art will also appreciate that various features could be combined to form multiple variations of the invention. Those skilled in the art will appreciate that some conventional aspects of FIGS. 2-4 have been simplified or omitted for clarity.

FIG. 2 demonstrates how co-processor circuitry 107 provides pipe-lined operation. FIG. 2 is vertically separated by dashed lines that indicate five packet processing stages: 1) context resolution, 2) context fetching, 3) priority queuing, 4) software application, and 5) context flushing. Co-processor circuitry 107 handles stages 1-3 to provide hardware acceleration. Core processor 104 handles stage 4 to provide software control with optimized efficiency due to stages 1-3. Co-processor

circuitry 107 also handles stage 5. Co-processor circuitry 107 has eight pipelines through stages 1-3 and 5 to concurrently process multiple packet streams.

In stage 1, requests to handle packets are resolved to a context for each packet in the internal data buffers. The requests are generated by receive interface 5 106, scheduler 105, and core processor 104 in response to incoming packets, scheduled transmissions, and application software instructions. The context information includes a channel descriptor that has information regarding how packets in one of 64,000 different channels are to be handled. For example, a channel descriptor may indicate service address information, traffic management parameters, 10 channel status, stream queue information, and thread status. Thus, 64,000 channels with different characteristics are available to support a wide array of service differentiation. Channel descriptors are identified by channel identifiers. Channel identifiers may be indicated by the request. A map may be used to translate selected bits from the packet header to a channel identifier. A hardware engine may 15 also perform a sophisticated search for the channel identifier based on various information. Different algorithms that calculate the channel identifier from the various information may be selected by setting correlation state bits in co-processor circuitry 107. Thus, the technique used for context resolution is externally controllable.

In stage 2, context information is fetched, if necessary, by using the channel 20 identifiers to transfer the channel descriptors to internal context buffers. Prior to the transfer, the context buffers are first checked for a matching channel identifier and validity bit. If a match is found, then the context buffer with the existing channel descriptor is associated with the corresponding internal data buffer holding the packet.

In stage 3, requests with available context are prioritized and arbitrated for core processor 104 handling. The priority may be indicated by the request – and it may be the source of the request. The priority queues 1-8 are 16 entries deep. Priority queues 1-8 are also ranked in a priority order by queue number. The priority
5 for each request is determined, and when the context and data buffers for the request are valid, an entry for the request is placed in one of the priority queues that corresponds to the determined priority. The entries in the priority queues point to a pending request state RAM that contains state information for each data buffer. The state information includes a data buffer pointer, a context pointer, context validity bit,
10 requester indicator, port status, a channel descriptor loaded indicator.

The work queue indicates the selected priority queue entry that core processor 104 should handle next. To get to the work queue, the requests in priority queues are arbitrated using one of various algorithms such as round robin, service-to-completion, weighted fair queuing, simple fairness, first-come first-serve,
15 allocation through priority promotion, and software override. The algorithms may be selected through scheduling state bits in co-processor circuitry 107. Thus, the technique used for prioritization is externally controllable. Co-processor circuitry 107 loads core processor 104 registers with the channel descriptor information for the next entry in the work queue.

20 In stage 4, core processor 104 executes the software application to process the next entry in the work queue which points to a portion of the pending state request RAM that identifies the data buffer and context buffer. The context buffer indicates one or more service addresses that direct the core processor to the proper functions within the software application. One such function of the software
25 application is traffic shaping to conform to service level agreements. Other functions

include header manipulation and translation, queuing algorithms, statistical accounting, buffer management, interworking, header encapsulation or stripping, cyclic redundancy checking, segmentation and reassembly, frame relay formatting, multicasting, and routing. Any context information changes made by the core processor are linked back to the context buffer in real time.

In stage 5, context is flushed. Typically, core processor 104 instructs co-processor circuitry 107 to transfer packets to off-chip memory 103 or transmit interface 108. If no other data buffers are currently associated with the pertinent context information, then co-processor circuitry 107 transfers the context information to off-chip memory 103.

FIG. 3 is a block diagram that illustrates co-processor circuitry 107 in an example of the invention. Co-processor circuitry 107 comprises a hardware engine that is firmware-programmable in that it operates in response to state bits and register content. In contrast, core processor 104 is a micro-processor that executes application software. Co-processor circuitry 107 operates in parallel with core processor 104 to conserve core processor capacity by off-loading numerous tasks from the core processor.

Co-processor circuitry 107 comprises context resolution 310, control 311, arbiter 312, priority queues 313, data buffers 314, context buffers 315, context DMA 316, and data DMA 317. Data buffers 314 hold packets and context buffers 315 hold context information, such as a channel descriptor. Data buffers 314 are relatively small and of a fixed size, such as 64 bytes, so if the packets are ATM cells, each data buffer holds only a single ATM cell and ATM cells do not cross data buffer boundaries.

Individual data buffers 314 are associated with individual context buffers 315 as indicated by the downward arrows. Priority queues 313 hold entries that represent individual data buffers 314 as indicated by the upward arrows. Thus, a packet in one of the data buffers is associated with its context information in an associated one of the context buffers 315 and with an entry in priority queues 313. 5 Arbitrator 312 presents a next entry from priority queues 313 to core processor 104 which handles the associated packet in the order determined by arbitrator 312.

Context DMA 316 exchanges context information between memory 103 and context buffers 315 through memory interface 109. Context DMA automatically 10 updates queue pointers in the context information. Data DMA 317 exchanges packets between data buffers 314 and memory 103 through memory interface 109. Data DMA 317 also transfers packets from memory 103 to transmit interface 108 through memory interface 109. Data DMA 317 signals context DMA 316 when transferring packets off-chip, and context DMA 316 determines if the associated 15 context should be transferred to off-chip memory 103. Both DMAs 316-317 may be configured to perform CRC calculations.

For a new packet from communication system 101, control 311 receives the new packet and a request to handle the new packet from receive interface 106.

Control 311 receives and places the packet in one of the data buffers 314 and 20 transfers the packet header to context resolution 310. Based on gap state bits, a gap in the packet may be created between the header and the payload in the data buffer, so core processor 104 can subsequently write encapsulation information to the gap without having to create the gap. Context resolution 310 processes the packet header to correlate the packet with a channel descriptor – although in some 25 cases, receive interface 106 may have already performed this context resolution.

The channel descriptor comprises information regarding packet transfer over a channel.

Control 311 determines if the channel descriptor that has been correlated with the packet is already in one of the context buffers 315 and is valid. If so, control 311
5 does not request the channel descriptor from off-chip memory 103. Instead, control 311 associates the particular data buffer 314 holding the new packet with the particular context buffer 315 that already holds the correlated channel descriptor. This prevents multiple copies of the channel descriptor from existing in context buffers 314. Control 311 then increments an in-use count for the channel descriptor
10 to track the number of data buffers 314 that are associated with the same channel descriptor.

If the correlated channel descriptor is not in context buffers 315, then control 311 requests the channel descriptor from context DMA 316. Context DMA 316 transfers the requested channel descriptor from off-chip memory 103 to one of the
15 context buffers 315 using the channel descriptor identifier, which may be an address, that was determined during context resolution. Control 311 associates the context buffer 315 holding the transferred channel descriptor with the data buffer 314 holding the new packet to maintain the correlation between the new packet and the channel descriptor. Control 311 also sets the in-use count for the transferred channel
20 descriptor to one and sets the validity bit to indicate context information validity.

Control 311 also determines a priority for the new packet. The priority may be determined by the source of the new packet, header information, or channel descriptor. Control 311 places an entry in one of priority queues 313 based on the priority. The entry indicates the data buffer 314 that has the new packet. Arbiter 312
25 implements an arbitration scheme to select the next entry for core processor 104.

Core processor 104 reads the next entry and processes the associated packet and channel descriptor in the particular data buffer 314 and context buffer 315 indicated in the next entry.

Each priority queue has a service-to-completion bit and a sleep bit. When the
5 service-to-completion bit is set, the priority queue has a higher priority than any
priority queues without the service-to-completion bit set. When the sleep bit is set,
the priority queue is not processed until the sleep bit is cleared. The ranking of the
priority queue number breaks priority ties. Each priority queue has a weight from 0-
15 to ensure a certain percentage of core processor handling. After an entry from a
10 priority queue is handled, its weight is decremented by one if the service-to-
completion bit is not set. The weights are re-initialized to a default value after 128
requests have been handled or if all weights are zero. Each priority queue has a
high and low watermark. When outstanding requests that are entered in a priority
queue exceed its high watermark, the service-to-completion bit is set. When the
15 outstanding requests fall to the low watermark, the service-to-completion bit is
cleared. The high watermark is typically set at the number of data buffers allocated
to the priority queue.

Core processor 104 may instruct control 311 to transfer the packet to off-chip
memory 103 through data DMA 317. Control 311 decrements the context buffer in-
20 use count, and if the in-use count is zero (no data buffers 314 are associated with
the context buffer 315 holding the channel descriptor), then control 311 instructs
context DMA 316 to transfer the channel descriptor to off-chip memory 103. Control
311 also clears the validity bit. This same general procedure is followed when
scheduler 105 requests packet transmission, except that in response to the request

from scheduler 105, control 311 instructs data DMA 317 to transfer the packet from memory 103 to one of data buffers 314.

FIG. 4 is a flow diagram that illustrates the operation of co-processor circuitry 107 when correlating buffers in an example of the invention. Co-processor circuitry 107 has eight pipelines to concurrently process multiple packet streams in accord with FIG. 3. First, a packet is stored in a data buffer, and the packet is correlated to a channel descriptor as identified by a channel identifier. The channel descriptor comprises the context information regarding how packets in one of 64,000 different channels are to be handled.

Next, context buffers 314 are checked for a valid version of the correlated channel descriptor. This entails matching the correlated channel identifier with a channel identifier in a context buffer that is valid. If the correlated channel descriptor is not in a context buffer that is valid, then the channel descriptor is retrieved from memory 103 and stored in a context buffer using the channel identifier. The data buffer holding the packet is associated with the context buffer holding the transferred channel descriptor. An in-use count for the context buffer holding the channel descriptor is set to one. A validity bit for the context buffer is set to indicate that the channel descriptor in the context buffer is valid. If the correlated channel descriptor is already in a context buffer that is valid, then the data buffer holding the packet is associated with the context buffer already holding the channel descriptor. The in-use count for the context buffer holding the channel descriptor is incremented.

Typically, core processor 104 instructs co-processor circuitry 107 to transfer packets to off-chip memory 103 or transmit interface 108. Data DMA 317 transfers the packet and signals context DMA 316 when finished. Context DMA 316 decrements the in-use count for the context buffer holding the channel descriptor,

and if the decremented in-use count equals zero, then context DMA 316 transfers the channel descriptor to memory 103 and clears the validity bit for the context buffer.

5 Memory Interface 109 -- FIGS. 5-6

FIGS. 5-6 depict a specific example of memory interface circuitry in accord with the present invention. Those skilled in the art will appreciate numerous variations from this example that do not depart from the scope of the invention.

Those skilled in the art will also appreciate that various features could be combined
10 to form multiple variations of the invention. Those skilled in the art will appreciate that some conventional aspects of FIGS. 5-6 have been simplified or omitted for clarity.

FIG. 5 is a block diagram that illustrates memory interface 109 in an example of the invention. Memory interface 109 comprises a hardware circuitry engine that is
15 firmware-programmable in that it operates in response to state bits and register content. In contrast, core processor 104 is a micro-processor that executes application software. Memory interface 109 operates in parallel with core processor 104 to conserve core processor capacity by off-loading numerous tasks from the core processor.

20 From FIG. 1, FIG. 5 shows memory 103, core processor 104, co-processor circuitry 107, transmit interface 108, and memory interface 109. Memory 103 comprises Static RAM (SRAM) 525 and Synchronous Dynamic RAM (SDRAM) 526, although other memory systems could be used in other examples of the invention. SDRAM 526 comprises pointer stack 527 and external buffers 528. Memory
25 interface 109 comprises buffer management engine 520, SRAM interface 521, and

SDRAM interface 522. Buffer management engine 520 comprises pointer cache 523 and control logic 524.

Conventional components could be used for SRAM interface 521, SDRAM interface 522, SRAM 525, and SDRAM 526. SRAM interface 521 exchanges
5 context information between SRAM 525 and co-processor circuitry 107. External buffers 528 use a linked list mechanism to store communication packets externally to integrated circuit 100. Pointer stack 527 is a cache of pointers to free external buffers 528 that is initially built by core processor 104. Pointer cache 523 stores pointers that were transferred from pointer stack 527 and correspond to external
10 buffers 528. Sets of pointers may be periodically exchanged between pointer stack 527 and pointer cache 523. Typically, the exchange from stack 527 to cache 523 operates on a first-in/first-out basis.

In operation, core processor 104 writes pointers to free external buffers 528 to pointer stack 527 in SDRAM 526. Through SDRAM interface 522, control logic 524
15 transfers a subset of these pointers to pointer cache 523. When an entity, such as core processor 104, co-processor circuitry 107, or an external system, needs to store a packet in memory 103, the entity reads a pointer from pointer cache 523 and uses the pointer to transfer the packet to external buffers 528 through SDRAM interface 522. Control logic 524 allocates the external buffer as the corresponding pointer is
20 read from pointer cache 523. SDRAM stores the packet in the external buffer indicated by the pointer. Allocation means to reserve the buffer, so other entities do not improperly write to it while it is allocated.

When the entity no longer needs the external buffer – for example, the packet is transferred from memory 103 through SDRAM interface 522 to co-processor
25 circuitry 107 or transmit interface 108 – then the entity writes the pointer to pointer

cache 523. Control logic 524 de-allocates the external buffer as the corresponding pointer is written to pointer cache 523. De-allocation means to release the buffer, so other entities may reserve it. The allocation and de-allocation process is repeated for other external buffers 528.

5 Control logic 524 tracks the number of the pointers in pointer cache 523 that point to de-allocated external buffers 528. If the number reaches a minimum threshold, then control logic 524 transfers additional pointers from pointer stack 527 to pointer cache 523. Control logic 524 may also transfer an exhaustion signal to core processor 104 in this situation. If the number reaches a maximum threshold,
10 then control logic 524 transfers an excess portion of the pointers from pointer cache 523 to pointer stack 527.

FIG. 5 is a table that illustrates buffer classes in an example of the invention. In the example of FIG. 3, there are 16 classes with 500 external buffers each for a total of 8,000 external buffers. Each class has a type: static, exclusive, or shared.
15 Static classes use their own external buffers without sharing. Exclusive classes use their own external buffers first, and then borrow external buffers from the fail-over classes. De-allocated external buffers from a given exclusive class are credited to the associated fail-over class until the number of borrowed external buffers is zero. Shared classes use their own external buffers and comprise the fail-over external
20 buffers for the exclusive classes.

Buffer classes are used to differentiate services among traffic streams by assigning different streams to different classes of external buffers. Traffic streams offering a higher quality-of-service are typically provided with greater access to external buffers. Bursty traffic may need an elastic exclusive/shared class
25 arrangement. A class may be associated with only one type of traffic, such as

Constant Bit Rate (CBR), Available Bit Rate (ABR), Variable Bit Rate (VBR), or Unspecified Bit Rate (UBR). CBR traffic without bursts typically uses static classes. ABR traffic uses exclusive and shared external buffer classes to respectively handle minimum cell rates and bursts. VBR and UBR traffic typically use shared buffer
5 classes.

The external buffers are separated into two separately managed pools A and B. Pools can be used for service differentiation. Pools are also helpful when the external buffers are located in separate memory devices, so each device may have its own independently managed pool.

10 For each class, control logic 524 tracks the number of pointers in pointer cache 523 that point to the de-allocated external buffers and the number of pointers in pointer cache 523 that point to the allocated external buffers. If the number of pointers to the de-allocated external buffers in one of the classes reaches a minimum threshold for that class, control logic 524 transfers an exhaustion signal for
15 that class to core processor 104. If the class is exclusive, control logic 524 may also borrow pointers from the corresponding fail-over class for use by the exclusive class, although conditions and thresholds may be used to limit the amount of borrowing. Control logic 524 tracks the number of pointers distributed to each class and may re-distribute pointers from one class to another based on certain conditions, such as
20 traffic loads. In addition, more ABR traffic may require more exclusive buffers at the expense of shared buffers where more UBR traffic has the opposite effect.

Scheduler Circuitry -- FIGS. 7-9

FIGS. 7-9 depict a specific example of scheduler circuitry in accord with the
25 present invention. Those skilled in the art will appreciate numerous variations from

this example that do not depart from the scope of the invention. Those skilled in the art will also appreciate that various features could be combined to form multiple variations of the invention. Those skilled in the art will appreciate that some conventional aspects of FIGS. 7-9 have been simplified or omitted for clarity.

5 FIG. 7 is a block diagram that illustrates scheduler 105 in an example of the invention. Scheduler 105 comprises a hardware circuitry engine that is firmware-programmable in that it operates in response to state bits and register content. In contrast, core processor 104 is a micro-processor that executes application software. Scheduler 105 operates in parallel with core processor 104 to conserve core
10 processor capacity by off-loading numerous tasks from the core processor.

 Scheduler 105 comprises control logic 730, scheduling boards 731-732, and context RAM 735. Board 731 is vertically separated into time periods 741-745 where a "1" indicates a reservation at that time period and a "0" indicates no reservation at that time period. Board 731 is horizontally separated into priority levels 751-754 that
15 are ranked from high at priority level #1 to low at priority level #4. Board 732 is similar to board 731, but has two priority levels and ten time periods. Control logic 730 process boards 731-732 independently of one another.

 Context RAM 735 has entries 736 that each hold one of thousands of possible channel descriptor identifiers. The channel descriptors that correspond to these
20 identifiers describe how packet transmission should be handled for a channel. For example, a channel descriptor indicates where packets for the channel are stored and how frequently they should be transmitted. Boards 731-732 are each associated with a different portion of context RAM 735. As indicated by the arrows, each time period at each priority level on each board is associated with its own one
25 of the context RAM entries 736, and thus, with a possibly unique channel descriptor.

To serve a reservation, control logic 730 sends a request that identifies the corresponding channel descriptor to co-processor circuitry 107.

In some examples of the invention, there are 64,000 channel descriptors, and thus, 64,000 corresponding board time periods. The 64,000 time periods can be distributed among boards in various combinations of 2,000, 4,000, 8,000, 16,000, 32,000, or 64,000 time periods per board. Boards can be separated into 1, 2, or, 4 priority levels that run concurrently in time. Each priority level uses up time periods, so an 8,000 time period board with four priority levels has 2,000 time periods per priority level that run concurrently. The minimum number of reservations in a given priority level is 512.

FIGS. 8-9 are block diagrams that illustrate scheduling board 731 in an example of the invention. Board 731 includes five time periods 741-745 and four priority levels 751-754. Fence 860 indicates the time period that control logic 730 is currently processing in each priority level. Control logic 730 processes board 731 to serve reservations and initiate packet transmissions at regular time intervals. At each time interval, referred to as a "GET", control logic 730 gets one reservation and advances fence 860. At a GET, fence 860 may advance in some priority levels but not others. As a result, fence 860 may be at different time periods in different priority levels.

At a GET, control logic 730 serves the highest priority reservation at fence 860. Control logic 860 then advances fence 860 to the next time period in the priority level that is served. Control logic 860 also advances fence 860 to the next time period in other priority levels that did not have a reservation at fence 860.

Control logic 860 does not advance fence 860 at priority levels that had an un-served reservation at fence 860.

On FIG. 8, fence 860 is at time period 3 for all priority levels, and priority level 1 is served, priority level 2 is un-served, and priority levels 3 and 4 are not reserved. As a result and as shown on FIG. 9, fence 860 is advanced to time period 4 at priority levels 1, 3, and 4, but not at priority level 2. Priority level 2 will have the highest priority reservation at the next GET, because fence 860 remains at time period 3 in priority level 2, and priority level 1 has no reservation at time period 4.

The various scheduling boards and priority levels provide a robust mechanism for differentiating services between traffic streams based on service level agreements. For example, the highest priority level of board 732 could be traffic with a guaranteed bandwidth rate, and the lower priority level could be traffic without any bandwidth guarantee. Board 731 has four priority levels and may have Constant Bit Rate (CBR) traffic at the highest priority, real time Variable Bit Rate (VBR) traffic at the second-highest priority, non-real-time VBR traffic at the third-highest priority level, and Available Bit Rate (ABR) traffic at the fourth and lowest priority level.

Traffic can also be allocated among boards to provide expensive high-quality service from one board, and cheap moderate-quality service from another board.

Prioritization circuitry within co-processor circuitry 107 can assign channels on the high-quality board to the highest priority queue to core processor 104 and assign channels on the moderate-quality board to the highest priority queue to core processor 104.

A board stall occurs when a higher priority level starves a lower priority level, and a reservation at the lower priority level will never get served. In a board stall, control logic 730 does not advance fence 860 until the stall is cured. If a force option is selected, the indicated priority level is serviced at the next GET. If a scan option is selected, the above-described board processing is modified. All reservations in the

highest priority level are serviced before any lower priority levels are serviced. In lower priority levels during this time, fence 860 is advanced at a GET if the time period is not reserved. The scan process repeats for the next highest priority level down through the lowest priority level.

5 Control logic 730 schedules a reservation in response to a "PUT" generated by core processor 104 or scheduler 105. Control logic 730 schedules the reservation by determining a start position. The start position is a number of time periods from the current fence location where the search starts for an available time period for the reservation. The start position may be specified in the PUT, or it may
10 be calculated by control logic 730.

 To calculate the start position, control logic 730 first retrieves scheduling parameters from the proper context buffer in co-processor circuitry 107 using a pointer in the PUT. Control logic 730 also increments the in-use count for the context buffer. Control logic 730 then executes dual Guaranteed Cell Rate (GCR)
15 scheduling algorithms based on the scheduling parameters to determine the start position.

 The scheduling parameters include a scheduling board indicator, first choice priority level, and second choice priority level that is a higher priority than the first choice. If the first priority level does not work for some reason, then the second
20 priority level is attempted in a priority promotion. In a priority promotion, the control logic 730 tries to find a reservation based on the Minimum Cell Rate (MCR), and if nothing is available, then control logic 730 tries to find a reservation based on the Peak Cell Rate (PCR).

The scheduling parameters also include usage values that determine how the dual algorithms are used. Based on the usage values, algorithm 1 and algorithm 2 are respectively be used for:

- PCR / nothing
- 5 MCR / PCR, or
- Sustained Cell Rate (SCR) / PCR.

For algorithm 1 and algorithm 2, the scheduling parameters respectively include:

- Theoretical Arrival Times (TAT1 / TAT2),
- Inter-Cell Intervals (ICI1 / ICI2), and
- 10 limits (L1 / L2).

The TATs are the expected arrival times, and the ICIs are the transmit frequencies. TATs are re-initialized if a reservation has been inactive for a long time (about one second).

The two scheduling algorithms are specified by the following psuedocode
 15 where PT_t is the present time, $TAT1_{t-1}$, $TAT2_{t-1}$ are old times read from the context buffer, and $TAT1_t$, $TAT2_t$ are new times written to the context buffer.

```

    if (PUT w/ no update instruction) {
        ICI1 = 0
        ICI2 = 0
        PIPELAT = 0
    } // else PUT with update; use the ICIs as provided

    TAT2t = max (TAT2t-1 + ICI2, PTt + ICI2)
    25 if (ALG==MCR/PCR)
        TAT1start = max (PTt - L1, min (TAT1t-1 + ICI1, PTt + L1))
    else TAT1start = max (TAT1t-1 + ICI1, PTt + ICI1)

    if (ALG==PCR/none) {
        30 start position = max (0, TAT1t - L1 - PTt)
    } else if (ALG==SCR/PCR) {
        start position = max (0, TAT2t - L2 - PTt, TAT1t - L1 - PTt)
    } else if (ALG==MCR/PCR) {
        start position = max (0, TAT2t - L2 - PTt)
    35 }
  
```

$TAT2_t = \max (TAT2_{start}, \text{landing position} + PT_t)$
 if (ALG==MCR/PCR) $TAT1_t = TAT1_{start}$
 else $TAT1_t = \max (TAT1_{start}, \text{landing position} + PT_t)$

5 if (PUT with write instruction) {
 write $TAT1_t$, $TAT2_t$, back to context buffer
 }
 decrement in-use count.

10

In addition to determining the starting position, control logic 730 determines the board configuration, fence location, and the oldest reservation at the fence in each priority level. Control logic 730 locates the start position from the fence at the selected priority level and on the selected scheduling board. Control logic 730 then
 15 searches for an available reservation time period. If the time period at the start time is already reserved, then the next available time period is reserved.

The number of time periods between the fence and the reserved time period is referred to as the landing position. The landing position must be smaller than the board to avoid wrapping around the board and reserving a time period that is
 20 improperly close to the fence. It may be the case that the priority level is full and reservations only open up as the fence is advanced. In this situation, any PUT will be reserved just behind the advancing fence.

Once the reservation is made, the applicable channel descriptor identifier is placed in the corresponding entry in context RAM 735. The scheduling parameters
 25 may also be updated and written back to the context buffer. The in-use count for the context buffer is decremented.

Those skilled in the art will appreciate variations of the above-described embodiments that fall within the scope of the invention. As a result, the invention is not limited to the specific examples and illustrations discussed above, but only by the
 30 following claims and their equivalents.

What is claimed is:

1. An integrated circuit (100) that processes communication packets, the integrated circuit (100) comprising:

co-processor circuitry (107) comprising context buffers (315) and data buffers (314) and configured to receive and store one of the communication packets in one of the data buffers (314), correlate the one communication packet with one of a plurality of channel descriptors, associate the one data buffer with one of the context buffers (315) holding the one channel descriptor to maintain the correlation between the one communication packet and the one channel descriptor, and determine a prioritized processing order for the communication packet relative to the other communication packets; and

a core processor (104) configured to execute a packet processing software application that directs the core processor (104) to process the one communication packet in the one data buffer based on the one channel descriptor in the one context buffer and based on the prioritized processing order; and wherein

the co-processor circuitry (107) is configured to operate in parallel with the core processor (104).

2. The integrated circuit (100) of claim 1 wherein the co-processor circuitry (107) is configured to and prevent multiple valid copies of the one channel descriptor from existing in the context buffers (315) by tracking a number of the data buffers (314) associated with the one context buffer.

3. The integrated circuit (100) of claim 1 wherein the co-processor circuitry (107) is configured to determine the prioritized processing order based on an availability of

resources required by the core processor (104) to process the communication packet.

4. The integrated circuit (100) of claim 1 further comprising:

- 5 a pointer cache (523) configured to store pointers that correspond to external buffers (528) configured to store the communication packets externally to the integrated circuit (100) and wherein the external buffers (528) and the pointers to the external buffers (528) are distributed among a plurality of classes; and
- control logic (524) configured to allocate the external buffers (528) as the
- 10 corresponding pointers are read from the pointer cache (523), de-allocate the external buffers (528) as the corresponding pointers are written back to the pointer cache (523), and track a number of the pointers to the de-allocated external buffers (528) for at least one of the classes

- 15 5. The integrated circuit (100) of claim 1 further comprising scheduling circuitry (105) comprising multiple scheduling boards (731, 732) wherein at least some of the scheduling boards (731, 732) have multiple priority levels (751-754), and wherein the scheduling circuitry (105) is configured to retrieve scheduling parameters cached in
- 20 the one context buffer and execute an algorithm based on the scheduling parameters to schedule subsequent transmission of the communication packet and to process one of the scheduling boards (731, 732) to schedule and subsequently initiate transmission of the communication packet.

6. A method of operating an integrated circuit (100) that processes communication packets, the method comprising:

in co-processor circuitry (107) comprising context buffers (315) and data buffers (314):

5 receiving and storing one of the communication packets in one of the data buffers (314);

correlating the one communication packet with one of a plurality of channel descriptors;

10 associating the one data buffer with one of the context buffers (315) holding the one channel descriptor to maintain the correlation between the one communication packet and the one channel descriptor; and

determining a prioritized processing order for the communication packet relative to the other communication packets; and

15 in a core processor (104):

executing a packet processing software application that directs the core processor (104) to process the one communication packet in the one data buffer based on the one channel descriptor in the one context buffer and based on the prioritized processing order; and

20 wherein

the co-processor circuitry (107) is configured to operate in parallel with the core processor (104).

7. The method of claim 6 further comprising, in the co-processor circuitry (107), preventing multiple valid copies of the one channel descriptor from existing in the context buffers (315) by tracking a number of the data buffers (314) associated with the one context buffer.

5

8. The method of claim 6 wherein determining the prioritized processing order comprises determining the prioritized processing order based on an availability of resources required by the core processor (104) to process the communication packet.

10

9. The method of claim 6 further comprising:

in a pointer cache (523):

storing pointers that correspond to external buffers (528)

configured to store the communication packets externally to the

15

integrated circuit (100) and wherein the external buffers (528) and the

pointers to the external buffers (528) are distributed among a plurality

of classes; and

in control logic (524):

allocating the external buffers (528) as the corresponding

20

pointers are read from the pointer cache (523);

de-allocating the external buffers (528) as the corresponding

pointers are written back to the pointer cache (523); and

tracking a number of the pointers to the de-allocated external

buffers (528) for at least one of the classes.

25

10. The method of claim 6 further comprising:

in scheduling circuitry (105) comprising multiple scheduling boards (731, 732)
wherein at least some of the scheduling boards (731, 732) have multiple priority

5 levels (751-754):

retrieving scheduling parameters cached in the one context buffer;

executing an algorithm based on the scheduling parameters to
schedule subsequent transmission of the communication packet; and

processing one of the scheduling boards (731, 732) to schedule and
10 subsequently initiate transmission of the communication packet.

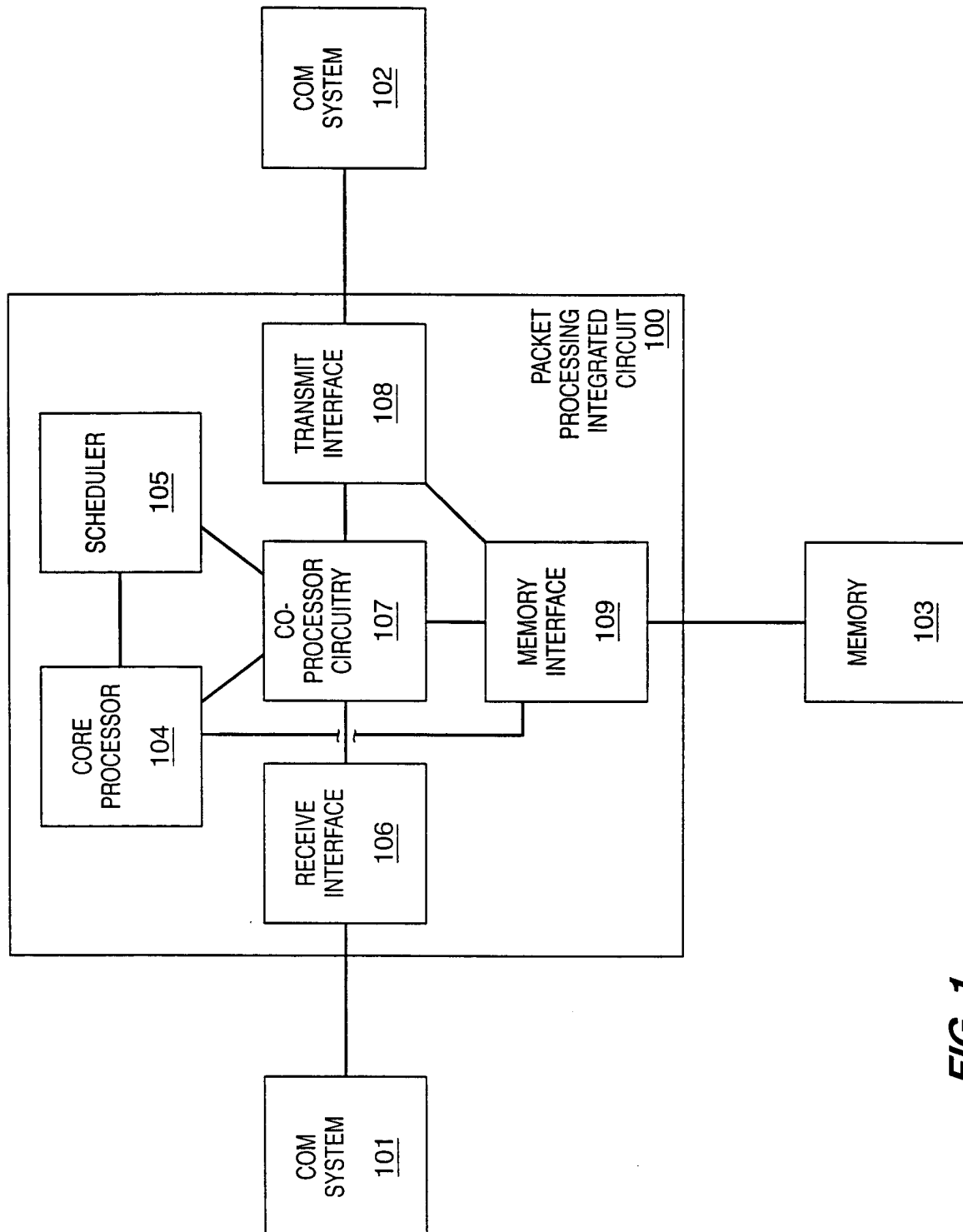


FIG. 1

2/9

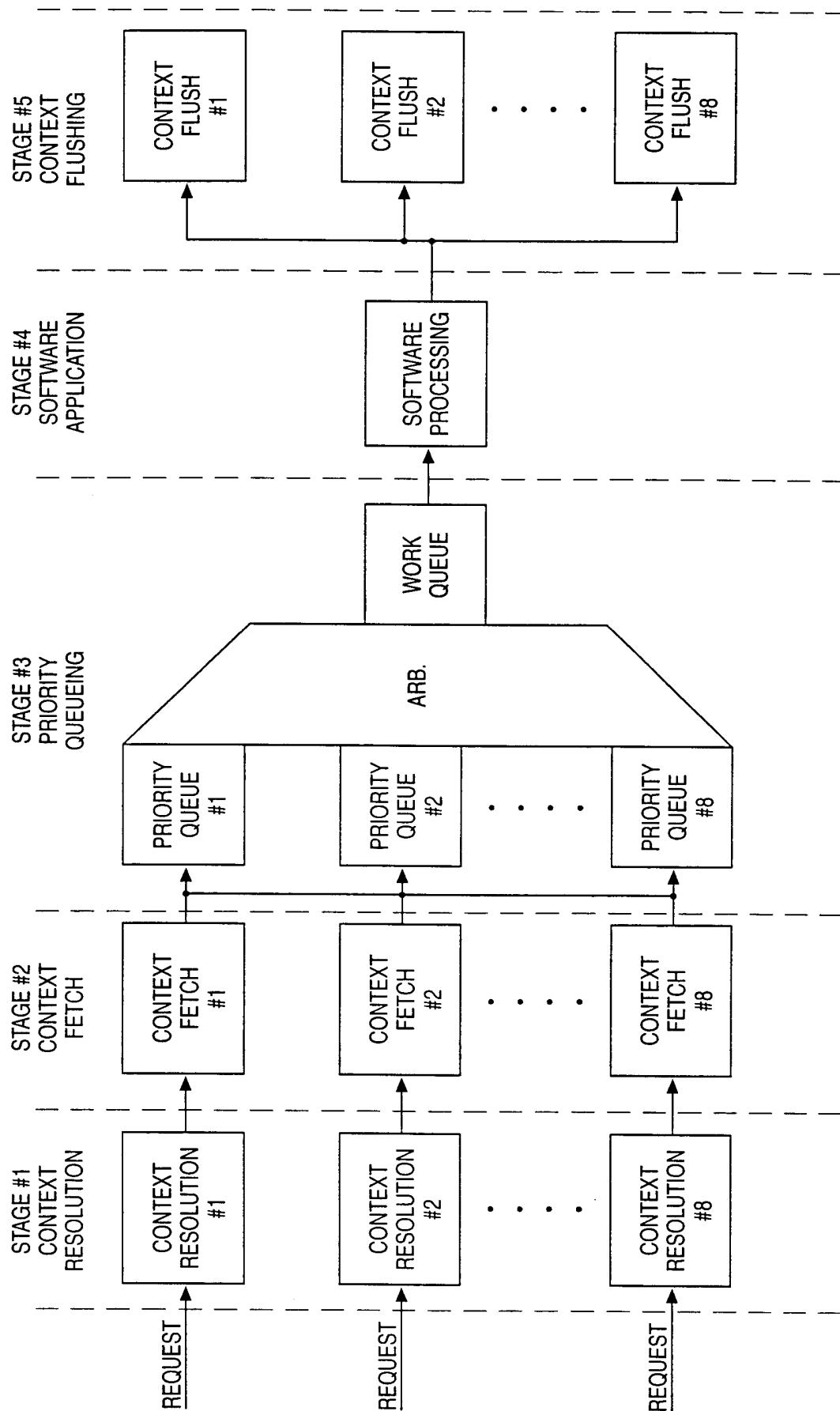


FIG. 2

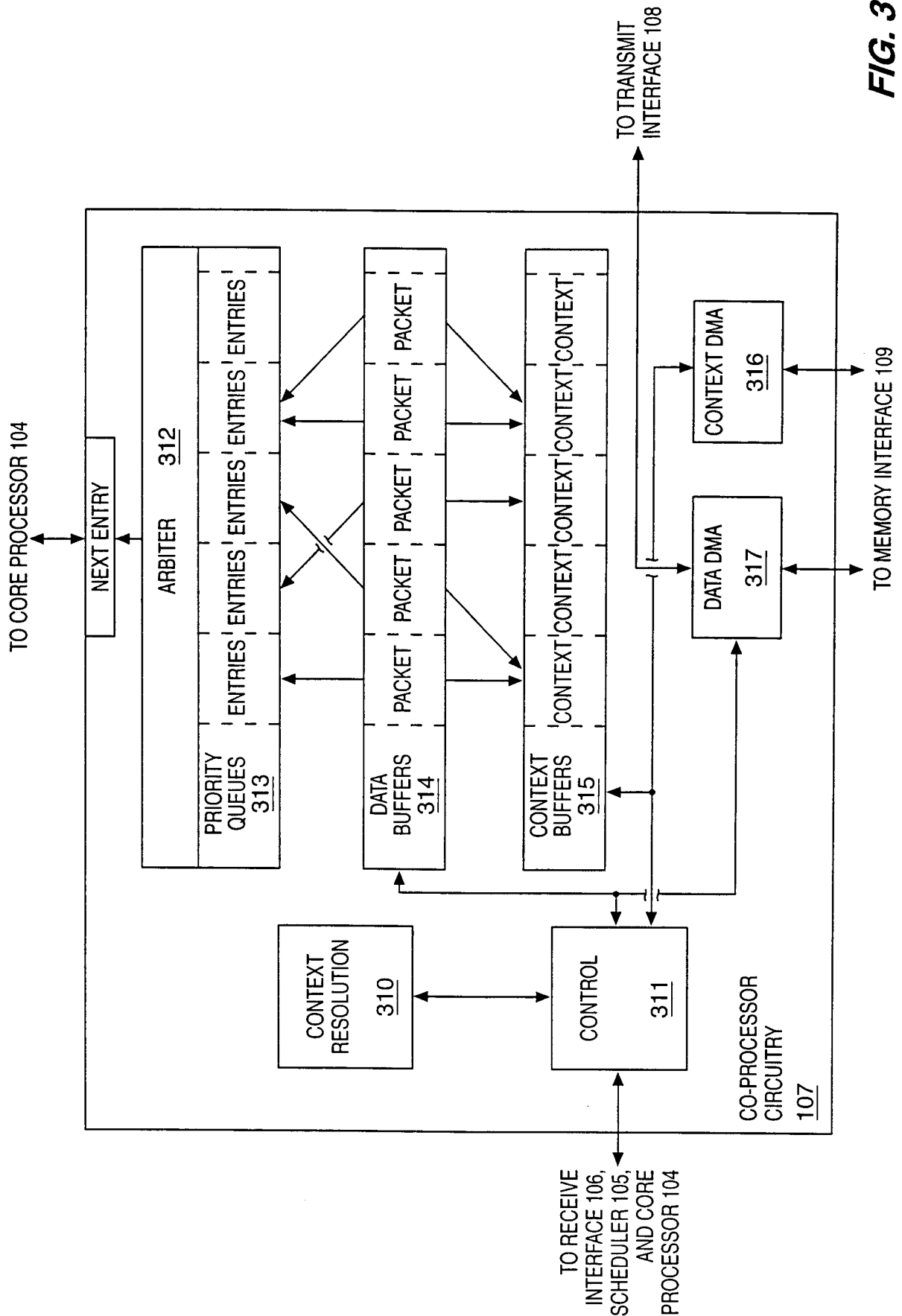
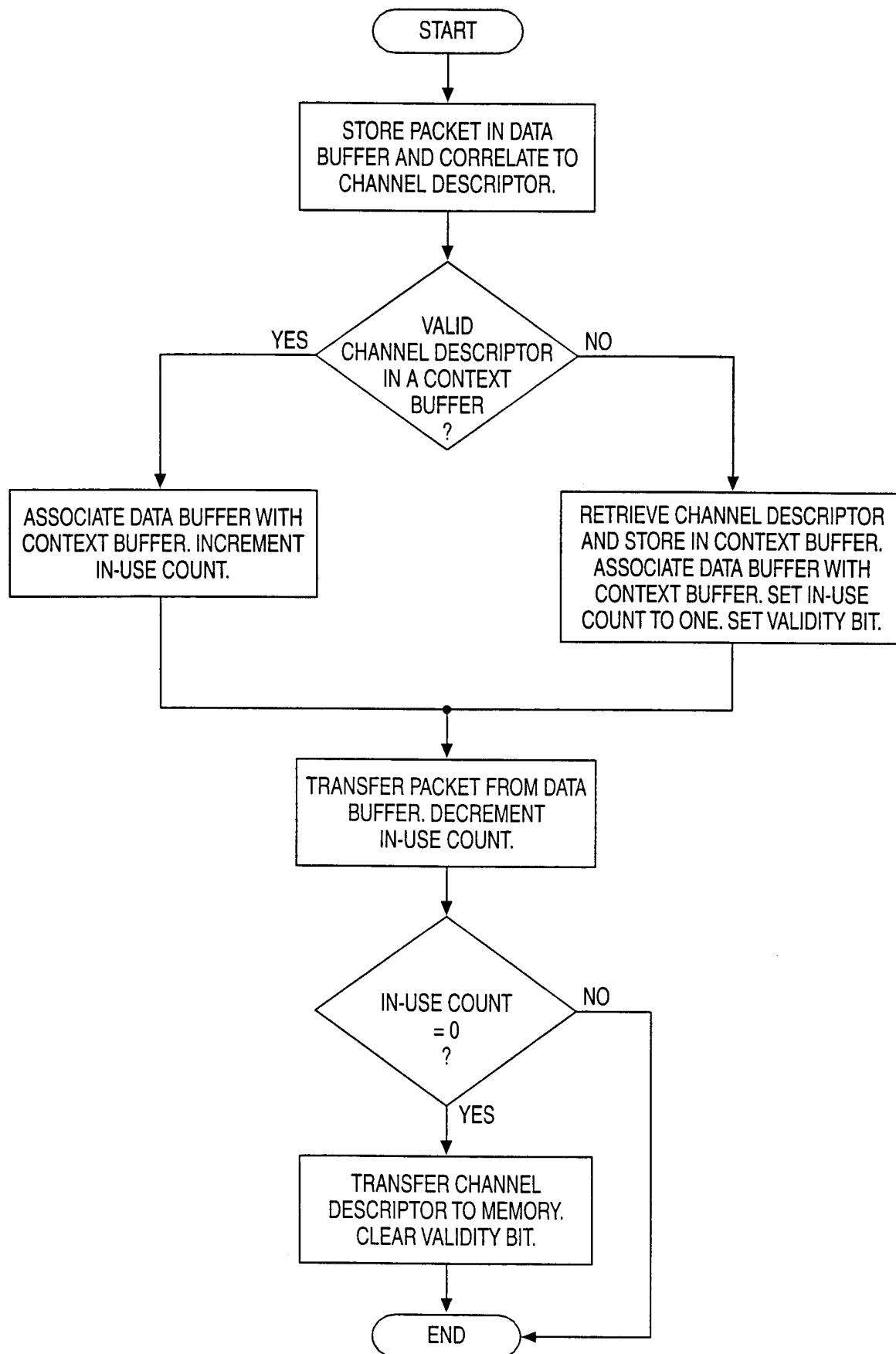


FIG. 3

4/9

FIG. 4

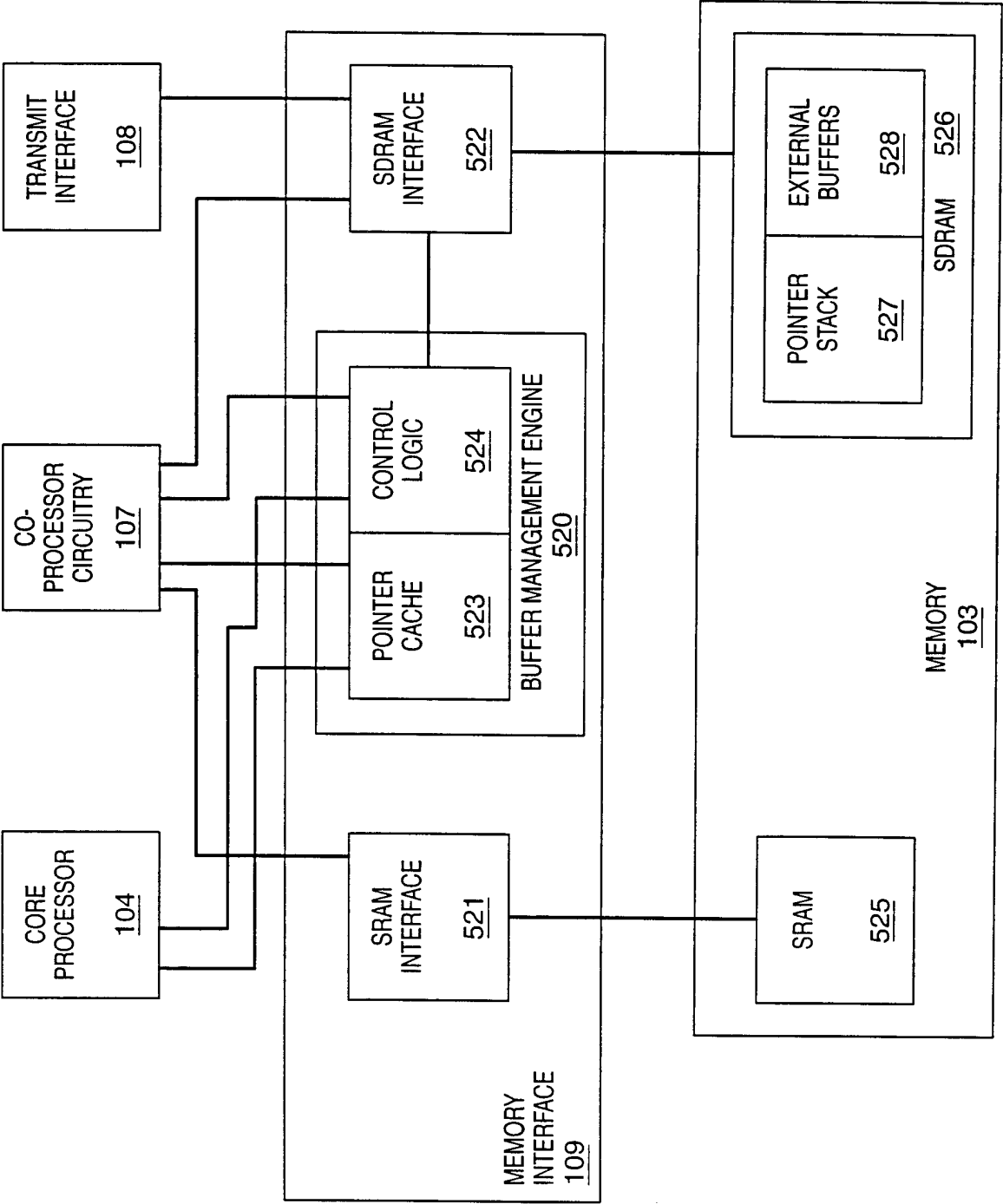


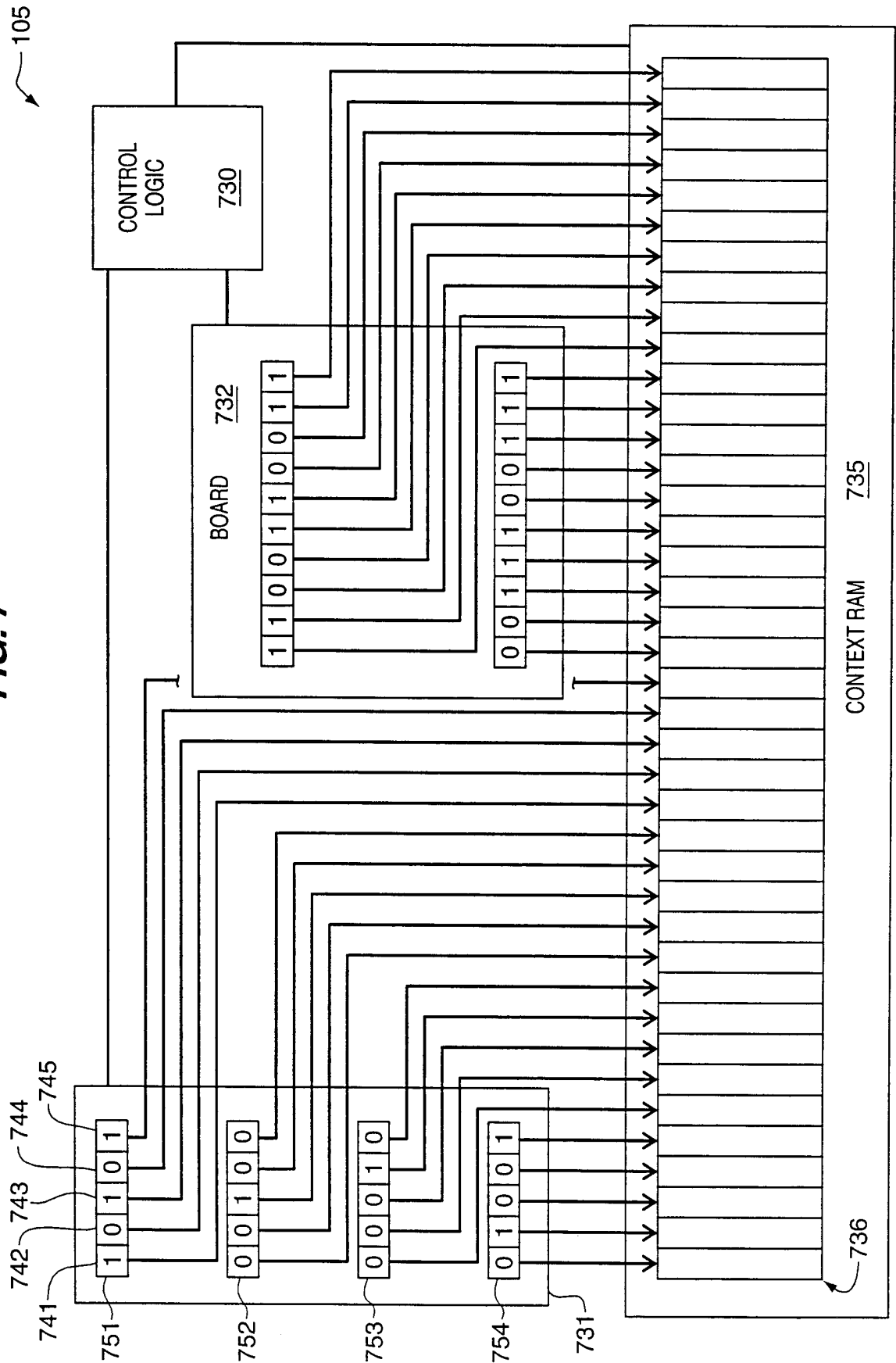
FIG. 5

6/9

CLASS	EXTERNAL BUFFERS	POOL	TYPE	FAIL OVER
1	1 - 500	A	STATIC	--
2	501 - 1000	A	EXCL.	3
3	1001 - 1500	A	SHARED	--
4	1501 - 2000	A	EXCL.	6
5	2001 - 2500	A	EXCL.	6
6	2501 - 3000	A	SHARED	--
7	3001 - 3500	A	STATIC	--
8	3501 - 4000	A	STATIC	--
9	4001 - 4500	B	STATIC	--
10	4501 - 5000	B	EXCL.	11
11	5001 - 5500	B	SHARED	--
12	5501 - 6000	B	EXCL.	14
13	6001 - 6500	B	EXCL.	14
14	6501 - 7000	B	SHARED	--
15	7001 - 7500	B	STATIC	--
16	7501 - 8000	B	STATIC	--

FIG. 6

FIG. 7



BOARD 731					
	741	742	743	744	745
TIME PERIOD	1	2	3	4	5
PRIORITY #1	1	0	1	0	1
PRIORITY #2	0	0	1	0	0
PRIORITY #3	0	0	0	1	0
PRIORITY #4	0	1	0	0	1

751752753754860

FIG. 8

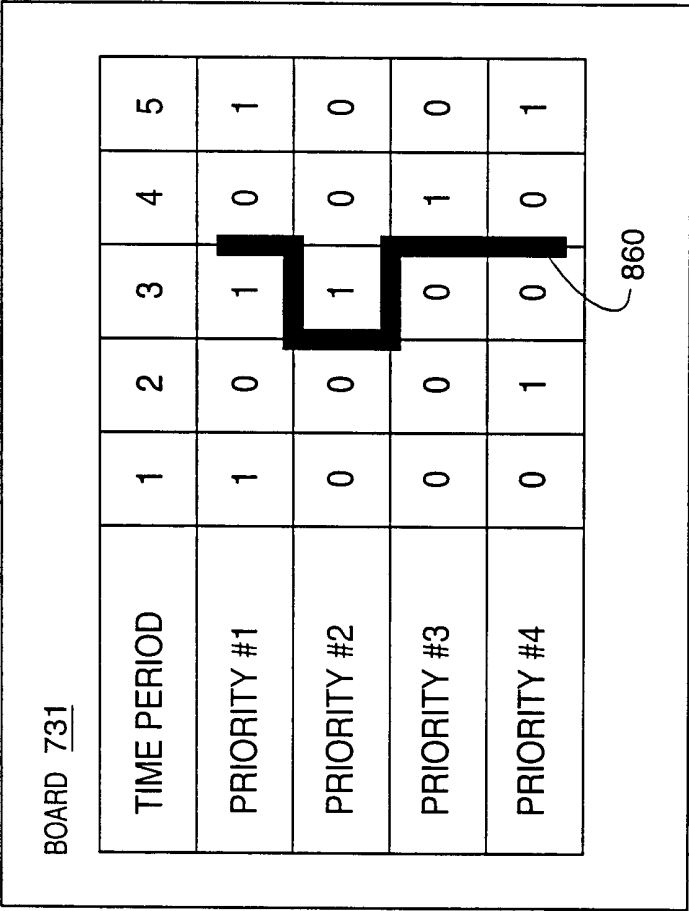


FIG. 9

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 00/22431

A. CLASSIFICATION OF SUBJECT MATTER

IPC 7 H04L12/56 H04Q11/04

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L H04Q

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 920 561 A (NATTKEMPER DIETER ET AL) 6 July 1999 (1999-07-06) figures 2,4-7,9 column 6, line 8 - line 65 column 11, line 64 -column 12, line 57 column 13, line 23 - line 24 column 13, line 63 -column 14, line 59 column 15, line 24 - line 44 column 21, line 22 -column 23, line 56 ---	1,4,6,9
A	EP 0 710 046 A (IBM) 1 May 1996 (1996-05-01) abstract figures 1,3,4 column 2, line 46 - line 47 column 4, line 21 - line 36 --- -/--	1-10

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents:

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- *G* document member of the same patent family

Date of the actual completion of the international search

17 November 2000

Date of mailing of the international search report

27/11/2000

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Lamadie, S

INTERNATIONAL SEARCH REPORT

Inter. nal Application No

PCT/US 00/22431

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	<p>US 5 748 630 A (BERGANTINO PAUL V ET AL) 5 May 1998 (1998-05-05) figures 1A,2 column 19, line 32 - line 39 column 24, line 11 - line 20 -----</p>	1-10

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/US 00/22431

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
US 5920561	A	06-07-1999	NONE	
EP 0710046	A	01-05-1996	US 5533020 A JP 8214042 A	02-07-1996 20-08-1996
US 5748630	A	05-05-1998	AU 3130497 A WO 9742736 A	26-11-1997 13-11-1997