PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46 | A1

(11) International Publication Number: **WO 98/36355**

(43) International Publication Date: 20 August 1998 (20.08.98)

(21) International Application Number: PCT/US98/02016

(22) International Filing Date: 2 February 1998 (02.02.98)

(30) Priority Data:
08/797,967 12 February 1997 (12.02.97) US

(71) Applicant: THE DOW CHEMICAL COMPANY [US/US]; 2030 Dow Center, Midland, MI 48674 (US).

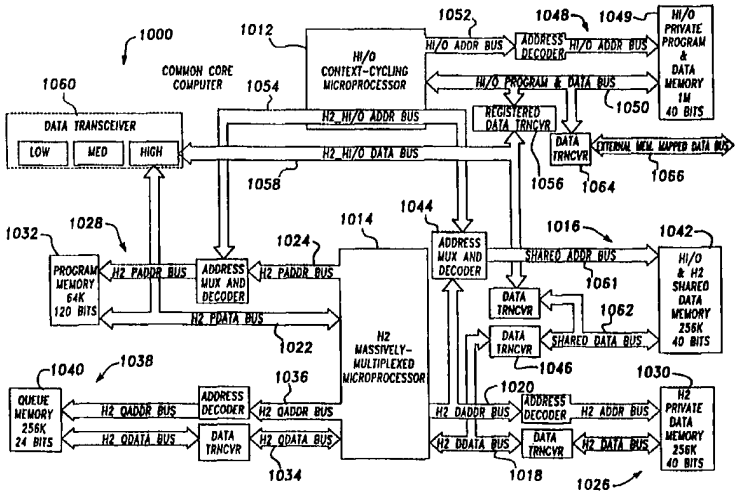
(72) Inventors: DUPREE, Wayne, P.; 5301 Stony Creek, Midland, MI 48640 (US). LUCAS, Jeff; 518 Parkwood Drive, Windsor, CO 80550 (US). VERNIERS, Gerrit; Heirweg 88, B–9180 Moerbieke–Waas (BE). ROOT, Larry; 3005 Camberley Lane, Midland, MI 48640 (US). CHURCHILL, Steve; 2588 Waskevich, Midland, MI 48642 (US).

(74) Agent: SCHULTZ, Dale, H.; Patent Dept., P.O. Box 1967, Midland, MI 48641–1967 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: A DEDICATED CONTEXT–CYCLING COMPUTER



(57) Abstract

A dedicated context–cycling microprocessor which features a plurality of input/output circuits for receiving and transmitting information and an individual set of dedicated on–board resources for each plurality of processing contexts. A distinct processing context is provided for each of a plurality of the input/output circuits, and a timed context is also provided for concurrently scheduling multiple processing contexts and enforcing time constraints associated with this schedule. The timed context has a pseudo–queue list which represents an ordered set of data parameters and program memory addresses for scheduling each of the processing contexts. The dedicated on–board resources include a plurality of registers for each of the processing contexts, such as at least one general purpose register and a program counter. A multiplexer circuit is also provided for moving data between the input/output circuits, the dedicated registers of the processing contexts and the computational unit. The input/output circuits include at least one serial and shared memory management unit, a plurality of embedded SCSI interfaces and a plurality of memory mapped registers. The dedicated context cycling microprocessor is also adapted to work in conjunction with, and effectively control another microprocessor, as well as control the access to an external memory system which is shared between these two microprocessors.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

A DEDICATED CONTEXT-CYCLING COMPUTER

The present invention relates to computer architectures and particularly to a dedicated context-cycling microprocessor which is capable of functioning as either a stand-alone processor or as an intelligent interface between one or more external devices and the shared data memory of another computer, such as a Harvard architecture computer.

To achieve high speed, some multiprocessor computers employ specialized dedicated processors for handling specific tasks. For example, it is common to supplement the main central processing unit with a separate math coprocessor, the math coprocessor being specifically designed to perform arithmetic computations at high speed. Although the math coprocessor may perform different computational functions (add, subtract, multiply, divide) these functions are all of the same type or class. A math coprocessor would not be suitable to handle asynchronous data communication in addition to its other arithmetic functions. The math coprocessor is an example of a special purpose processor for a multiprocessor architecture that is dedicated to performing only one task.

As multiprocessor computer systems become more sophisticated, single purpose coprocessors begin to have drawbacks. It would be desirable if multiprocessor systems could be designed to support different classes of tasks by the coprocessor. For example, The Dow Chemical Company has developed a common core computer that employs a Harvard architecture computer as its main central processing unit. The Applicants' assignee has found the Harvard architecture to be highly desirable in process control applications where high speed performance is of utmost importance. The context-cycling microprocessor of the present invention works as an asynchronous data communication handler for this Harvard architecture common core computer.

A computer which includes the following characteristics is generally referred to as having a "Harvard" architecture. Namely, the computer will be designed with separate instruction and data stores and independent buses will be provided to enable the central processing unit (CPU) of the computer to communicate separately with each of these stores. Contrast this architecture with the "von Neumann" or "Princeton" based computer architecture, which generally employs the same physical store for both instructions and data, and a single bus structure for communication with the CPU.

Various approaches have been taken to designing a microcomputer or microprocessor with a Harvard architecture, as represented by the following patents: Yasui et al., U.S. Patent No. 5,034,887, issued on July 23, 1991, entitled "Microprocessor With Harvard Architecture"; Portanova et al., U.S. Patent No. 4,992,934, issued on February 12, 1991, entitled "Reduced Instruction Set Computing Apparatus And Methods"; Mehrgardt et al., U.S. Patent No. 4,964,046, issued on October 16, 1990, entitled "Harvard Architecture Microprocessor With Arithmetic Operations And Control Tasks for Data Transfer Handled Simultaneously"; and Simpson, U.S. Patent No. 4,494,187, issued on January 15, 1985, entitled "Microcomputer With High Speed Program Memory." Additionally, it should be noted that the Intel i860 64-bit microcomputer has been described as having an on-board Harvard architecture due to the provision of separate instruction and data cache paths. In this regard, a description of the Intel i860 chip design may be found in i860 Microprocessor Architecture, by Neal Margulis, Osborne McGraw-Hill 1990.

The use of separate instruction and data communication paths in a Harvard architecture machine effectively increases the overall speed of the computer by enabling an instruction to be accessed at the same time that data for this or another instruction is accessed. In the context of programmed operations, the instruction is usually referred to as the "opcode" (the operation code), and the data is referred to the "operand." While the benefit in speed of using the Harvard architecture is significant, a significant advance in this design has also been described in commonly assigned Sederlund et al., International Application No. PCT/US95/13423, filed on October 5, 1995, published as International Publication No. WO 96/11443 on April 18, 1996, and entitled "An Extended Harvard Architecture Memory System." This extended memory system employs an address store for containing an ordered sequence of program memory addresses. The address store (referred to as "queue memory") determines the sequence of operations to be implemented through its list of program memory addresses. In this regard each of these program memory addresses identifies the location of the first instruction of a particular subroutine which is contained in the program memory. The address store may also contain the addresses of one or more subroutine arguments which are, in turn, contained in either a value store or in a data memory. Thus, the address store may be utilized as a location server for both the program memory and the data memory of a computer which is based upon the Harvard architecture.

As the preceding suggests, The Dow Chemical Company is committed to the use of highly advanced computer systems that are uniquely suited to process control applications. In this regard, whereas the massively multiplexed CPU design, based on a Harvard architecture, is highly efficient at performing process control instructions at very high speed, another computer architecture is needed to best handle numerous asynchronous events that occur in process control applications. In this regard, although The Dow Chemical Company envisions its massively multiplexed CPU design and the dedicated context-cycling computer of the present invention to be well suited to chemical and manufacturing process applications, the foregoing innovations are equally suited to all other types of process control applications, ranging from power plant to spacecraft.

The present invention provides a dedicated context-cycling microprocessor or computer for handling the input and output tasks on behalf of its associated Harvard architecture central processing unit. As it turns out, providing this functionality is not as easy as it might appear. In a complex process control system there are many widely varying input/output functions and asynchronous events that must be handled. Inasmuch as the main process control computer has been designed to operate at extremely high speeds, the input/output functions must also be handled at comparable high speeds, otherwise the input/output becomes a bottleneck in the system. One approach, consistent with conventional practice, would be to provide a separate coprocessor for each input/output function. However, this would dramatically increase the cost of the system and would unduly complicate the interface between the main processor and the input/output processors.

Another approach would be to employ a single coprocessor that may be programmatically cycled through different functional states. However, such an approach has not heretofore been feasible because of the computational overhead required to save one context state before operating the next.

The present invention solves the problem by providing a dedicated context-cycling microprocessor that has dedicated registers and program counters for each different functional context. As used herein the term "context" (within a processor) refers to a specialized physical circuit having a dedicated program counter to identify the current location in a program instance for fetching an op code for execution in a computational unit—such as the computational unit of a general purpose computer processor. The specialized physical circuit is useful for exchanging data between external entities and the configuration and computational assets of the processor (including the computational unit)

according to a predefined protocol, or for performing predefined data manipulation functions. The context may also include an embedded state machine that may operate independently of the computational unit to aid in the exchange of data.

In the preferred configuration, a plurality of contexts share common assets, such as by time division multiple access, each context cycling through active and inactive states according to a context priority scheme. Common assets are assets actively used by only one context at a time (such as, without limitation, external data memory, computational unit, comparator, program counter calculator and local memory). Each context assumes one of a plurality of functional states, including at least one active state and at least one inactive state. In the active state the context may communicate with the computational unit and may also use other common assets. In the inactive state, the embedded state machine of a context may continue to function although the program counter of its context is not being used to identify an op code for the manipulation of configuration and computational assets. Each context has a data store, such as a memory or register, for retaining the process and configuration state of pertinent resources, attributes, or aspects associated with either the interface or the function while another context is active. In the preferred embodiment, each context is configured as an embedded microcontroller associated with a processor for performing mathematical computations.

The context-cycling microprocessor rapidly cycles from context to context without the usual overhead of saving state; the dedicated registers and program counters automatically save state without wasting processor cycles as in the conventional practice.

Accordingly, it is a principal objective of the present invention to provide a microprocessor design which has special capabilities for handling input/output and asynchronous events.

It is a more specific objective of the present invention to provide a microprocessor design which is capable of dynamically cycling between a number of both device and time handling contexts.

It is another objective of the present invention to provide a context-cycling microprocessor design which may be utilized in either a stand-alone processor or as an intelligent communication interface for another processor which operates as the source and destination of data.

It is a further objective of the present invention to provide a context-cycling microprocessor design which is capable of acting as a time-line enforcer.

It is an additional objective of the present invention to provide a context-cycling microprocessor design which is also capable of acting as a plurality of distinct, but coordinated controllers for processing communication requests with external devices.

It is also an objective of the present invention to provide a context-cycling microprocessor design which is capable of not only rapidly handling external communication requests, but also capable of managing a shared memory system in conjunction with another microprocessor.

To achieve the foregoing objectives, the present invention provides a dedicated context-cycling microprocessor which has at least one computational unit for performing mathematical or logical functions (or a combination thereof), a plurality of circuits for receiving and transmitting information, and a bus structure for transferring instructions and data with external memory devices, characterized by: a plurality of processing contexts each having an individual set of dedicated registers in which a distinct processing context is provided for each of the plurality of the circuits, the processing contexts including a timed context for concurrently scheduling multiple processing contexts into a schedule and for enforcing time constraints associated with the schedule; and a multiplexer circuit for moving data between the circuits, the dedicated registers, and the computational unit - the multiplexer circuit having a move bus coupled to the bus structure.

The overall features of the context-cycling microprocessor can, therefore, be summarized as: a plurality of input/output circuits for receiving and transmitting information; a plurality of processing contexts each having an individual set of dedicated registers in which a distinct processing context is provided for each of the plurality of the input/output circuits, the processing contexts including a timed context for concurrently scheduling multiple processing contexts into a schedule and for enforcing time constraints associated with the schedule; at least one computational unit for performing mathematical performing mathematical or logical functions or a combination thereof; a bus structure for transferring instructions and data with external memory devices; and a multiplexer circuit for moving data between the input/output circuits, the dedicated registers, and the computational unit, the multiplexer circuit having a move bus coupled to the bus structure. In further detail, the above may include a local RAM, a program memory, and a pseudo-queue list (for at least

the timed context) representing an ordered set of pointers (in the local RAM) to data and program memory words used for scheduling the processing contexts. Alternatively, the input/output circuits can also include at least one serial and shared memory management unit (incorporating a parallel-to-serial converter and a transmitter circuit which is capable of transmitting a plurality of serial word frames without any spaces between them) having its own processing context.

In its use within a computer, the invention provides for a multi-protocol data processing computer for communicating with multiple processes comprising: an input/output (I/O) processor (for communicating with multiple processes using different protocols) having a context cycling system for defining a plurality of independent operating-state contexts (each context devoted to one of the protocols); the context cycling system having (A) a plurality of program counter registers and a plurality of general purpose registers where each register is dedicated to the contexts in such a manner that each context has its own program counter register and its own general purpose registers and (B) a priority encoder for selecting one of the contexts as the operative context, such that (a) the program counter register and general purpose registers of the operative context are identified to the I/O processor as (1) the operative program counter register and (2) the operative general purpose registers, and (b) the remaining program counter registers and general purpose registers are treated by the I/O processor as inoperative; the I/O processor (A) communicating with one of the multiple processes by accessing the operative program counter register and the operative general purpose registers, such that communication is effected in accordance with the operative context, and (B) saving the state of the contexts not selected as operative by leaving the contents of the inoperative program counters and general purpose registers unchanged while communication is effected in accordance with the operative context.

The invention is also useful as a high-performance common core computer which features: a first microprocessor; a first private memory system connected to the first microprocessor; a second microprocessor; a second private memory system connected to the second microprocessor (including a plurality of input/output circuits for receiving and transmitting information with external communication devices and timing means for controlling the operation of the first microprocessor and access to the shared memory system); and a shared memory system connected to both the first and second microprocessors. The first microprocessor may have a Harvard architecture and the second

microprocessor may have an opcode width corresponding to a data word width of the first microprocessor. Additionally, the second microprocessor may include: an individual set of dedicated registers for each of a plurality of processing contexts in which a distinct processing context is provided for each of a plurality of the input/output circuits; a timed context (type of processing context) for concurrently scheduling multiple processing contexts and enforcing time constraints associated with the schedule; at least one computational unit for performing mathematical and/or logical functions (the computation unit being coupled to at least one of the input/output circuits); multiplexer circuit means for moving data between the input/output circuits, the dedicated registers, and the computational unit; a bus structure for transferring instructions and data with the shared memory system and the second private memory system; and a multiplexer circuit (having a move bus coupled to the bus structure) for moving data between the input/output circuits, the dedicated registers, and the computational unit. Additionally, some of the processing contexts can have their own plurality of interrupt signal inputs.

The invention is also useful in the form of a multi-protocol multiprocessor data processing system for communicating with and controlling multiple processes which has: an input/output (I/O) processor, for communicating with multiple processes using different protocols, with a first private memory for access in communicating with the multiple processes; a second processor, for executing process control instructions to control the multiple processes, with a second private memory for access in executing process control instructions; a shared memory separate from the first and second private memories for access by both the I/O processor and the second processor for establishing communication between the I/O processor and the second processor; and a context cycling system in the I/O processor for defining a plurality of independent operating-state contexts, each context being devoted to one of the protocols. The context cycling system has (a) a plurality of program counter registers and a plurality of general purpose registers each dedicated to the contexts such that each context has its own program counter register and its own general purpose registers and (b) a priority encoder for selecting one of the contexts as the operative context such that (1) the program counter register and general purpose registers of the operative context are identified to the I/O processor as the operative program counter register and the operative general purpose registers, and (2) the remaining program counter registers and general purpose registers are treated by the I/O processor as inoperative. The I/O processor (a) communicates with one of the multiple processes by accessing the operative program counter register and the operative general purpose registers, such that

communication is effected in accordance with the operative context, and (b) saves the state of the contexts not selected as operative by leaving the contents of the inoperative program counters and general purpose registers unchanged while communication is effected in accordance with the operative context.

Each of the sets of dedicated registers in the present invention includes at least one general purpose register and a program counter, and the timed context has access to at least the program counter of the dedicated registers in all of the other of the processing contexts. The processing contexts in the present invention each include an idle state in which the context is waiting to be scheduled, an inactive state in which a scheduled context is waiting for an event to occur, a bidding state in which a scheduled context is competing for service, and an active state in which the scheduled context has been granted the highest priority and is executing a fetched instruction; furthermore, a scheduler latch is provided for each of the processing contexts except for the timed context, and a context purpose register is provided for the timed context (with each of the scheduler latches being mapped into the context purpose register for the timed context in order to schedule one or more of the processing contexts for service). The processing contexts have preassigned priority levels, and at least some high priority processing contexts are capable of lowering their own priority. The multi-context microprocessor further includes a priority encoding circuit which ensures that one instruction of an active processing context must be executed before the active processing context is permitted to bid again for service. In the described embodiment, the multi-context processor accesses a local RAM and a program memory, and the timed context of the processor has a pseudo-queue list stored in local RAM which represents an ordered set of pointers to data and program memory words used for scheduling the processing contexts. The input/output circuits can include at least one serial and shared memory management unit comprising a parallel-to-serial transmitter circuit capable of transmitting a plurality of serial word frames without deadtime, and the serial and shared memory management unit has its own processing context. Some of the input/output circuits also can include a plurality of SCSI interfaces, with each of the SCSI interfaces having its own programming context.

The present invention can also be described as a method of processing input/output signals of different types by a process control computer by: defining a plurality of functional contexts corresponding to the different types of input/output signals; configuring an input/output processor to have a dedicated program counter and at least one dedicated

general purpose register for each of the functional contexts; and operating the input/output processor to cycle from one context to another such that the processing state of a first context is saved in the dedicated program counter and dedicated general purpose register of that first context while switching to process a second context by accessing the program counter of that second context.

The input/output processor operates in synchronism with a system clock that defines a fundamental clock interval; accordingly, the step of operating the input/output processor is performed such that one context is active during a given fundamental clock interval. The fundamental clock interval is subdivided into a first component and a second component (that follows the first component); the processing state of the first context is saved in the dedicated general purpose register of the first context during the first component of a first fundamental clock interval, and the program counter of the second context is accessed in preparation to process the second context during the second component of the first fundamental clock interval.

A second processor can be configured to have a memory space that is shared with the input/output processor and uses the input/output processor to communicate information about the input/output signals to the second processor by storing the information as data in the shared memory space. The second processor can also have a memory space that is shared with the input/output processor and uses the input/output processor to supply timing information to the second processor; the plurality of contexts can include a timed context that oversees the order in which context switching is performed; and the functional contexts can have different priorities assigned so that the step of operating the input/output processor is performed by selecting the second context through a priority bidding process.

Additional features and advantages of the present invention will become more fully apparent from a reading of the detailed description of the preferred embodiment and the accompanying drawings in which:

Figure 1 is a block diagram of a Harvard architecture-based common core computer which employs a context-cycling microprocessor according to the present invention.

Figure 2 is a simplified block diagram of the context-cycling microprocessor shown in Figure 1.

Figure 3 (Figures 3A and 3B, collectively) is a detailed block diagram of the context-cycling microprocessor shown in Figure 1.

Figure 4A is a detailed block diagram of the timer circuit associated with the timed context. Figure 4B is a detailed block diagram of a portion of the context purpose register for the timed-context, which is contained in the context management circuit (1132) generally shown in Figure 3.

Figures 5A and 5B provide diagrammatic views of typical pseudo-queue entries according to the present invention.

Figure 6 is a detailed block diagram of an interrupt priority encoder circuit as generally shown in Figure 3.

Figure 7 is a detailed block diagram of the interrupt selection circuit for the timed context.

Figures 8A-8F are a set of detailed block diagrams which illustrate examples of the dedicated registers for the processing contexts.

Figure 9 is a timing diagram of the clock signals used by the context-cycling microprocessor.

Figure 10A is a detailed block diagram of the multiplexed program/data bus structure shown in Figure 3. Figure 10B is a detailed block diagram of the bank switch circuit of the preferred embodiment.

Figure 11 is a detailed block diagram of the move multiplexer circuit contained in mux module circuit 1146 of Figure 3 and of the external RAM and program data memory RAM.

Figures 12A-12D represent a general schematic diagram of the move multiplexer and other multiplexing and data manipulation circuits contained in mux module circuit 1146 of Figure 3.

Figure 13 is a block diagram of the opcode decoder shown in Figure 3 and memory protect functionality.

Figures 14A and 14B are charts illustrating the instruction types utilized by the context-cycling microprocessor.

Figures 15A-15E provide a set of detailed block diagrams for the serial and shared memory management units shown in Figure 3.

Figure 16 is a detailed block diagram for the arbitration circuit shown in Figure 3.

Figure 17 is a timing diagram of the clock signals used in connection with the shared memory interface and coprocessing Harvard architecture ASIC.

Figure 18 is a diagram of the software reference model of the common core computer.

Figure 19 is a data flow diagram illustrating how the HI/O processor saves state.

Referring to Figure 1, a block diagram of a common core computer 1000 is shown. In particular, the common core computer 1000 features a dedicated context-cycling microprocessor 1012 according to the present invention. The common core computer 1000 further includes a Harvard architecture microprocessor 1014 and a shared memory system 1016. For sake of simplicity, the Harvard architecture microprocessor 1014 is generally referred to herein as the "H2" microprocessor. Similarly, the dedicated context-cycling microprocessor 1012 is also generally referred to herein as the "HI/O." However, it should be noted that the HI/O microprocessor 1012 is sometimes referred to as the "HarvI/O" microprocessor in the drawings.

While the HI/O microprocessor 1012 is designed for use as a stand-alone processor, Figure 1 specifically serves to illustrate how the HI/O microprocessor 1012 may advantageously be used to work with another microprocessor. In this particular embodiment, the HI/O microprocessor 1012 is used to provide an intelligent interface between the shared memory system 1016 of the H2 microprocessor 1014 and external devices. For example, the H2 microprocessor 1014 may be used in a process control computer which receives input signals from a number of field sensors and determines the output signals needed to drive the physical equipment that governs the process, such as pumps and valves. In such a case, the H2 microprocessor 1014 is the eventual destination

of all field input data and source of all field output data. Accordingly, the design of the H2 microprocessor 1014 should focus on rapidly executing the instructions needed to process the various input signals received. Nevertheless, the need exists to transmit both input and output data from the H2 microprocessor 1014 to other external devices, such as a computer network which includes one or more operator stations for supervising the process being controlled. Thus, the HI/O microprocessor 1012 has been designed with exceptional capabilities for rapidly managing asynchronous events, such as communication requests for data from the H2 microprocessor 1014 and other external computer communication and computation entities. The design of the common core computer 1000 capitalizes on the signal transfer capabilities of the HI/O microprocessor 1012 as will be more fully appreciated below.

In the embodiment of the common core computer 1000 illustrated in Figure 1, the H2 microprocessor 1014 is shown to be of a Harvard architecture design which utilizes an expanded memory system. As generally provided in a Harvard architecture-based design, the H2 microprocessor 1014 includes separate data and program bus structures. Specifically, the H2 microprocessor 1014 includes data bus 1018 (DDATA), a data address bus 1020 (DADDR), a program data bus 1022 (PDATA) and a program address bus 1024 (PADDR). The data bus 1018 and the data address bus 1020 are used to provide read/write communication with an external data memory system 1026 while the program data bus 1022 and the program address bus 1024 are used to provide read/write communication with an external program memory system 1028. The data memory system 1026 includes a private data memory circuit 1030, while the program memory system provides a physically separate program memory circuit 1032.

Preferably, the H2 microprocessor 1014 also includes a queue data bus 1034 (QDATA) and a queue address bus 1036 (QADDR) for communicating with an external queue memory system 1038. The queue memory system 1038 includes a queue memory circuit 1040 which is used to store an ordered list of program memory addresses (as well as subroutine arguments). Each of these program memory addresses generally represents the initial address of a distinct subroutine which has been stored in the program memory circuit 1032. In other words, the H2 microprocessor 1014 is preferably programmed to proceed sequentially down through the "queue list" found in the queue memory circuit 1040 in order to execute the instructions which are contained in the subroutines stored in the program memory circuit 1032. This approach has several advantages, such as significant

decrease in the amount of memory required to store an application program for the computer. Additionally, execution speed may be enhanced because normal subroutine overhead is avoided, such as pushing one or more addresses onto a stack. Additional subroutine arguments may be readily changed by a programmer without having to recompile the program. Rather, the argument value may be changed directly in the memory location referenced in queue memory circuit 1040.

The H2 microprocessor 1014 is preferably constructed of a central processing unit design such as that described in the Dupree et al., International Application No. PCT/US95/00341, filed on January 10, 1995, published as International Publication No. WO 95/19006 on July 13, 1995, and entitled "A Massive Multiplexed Super Scalar Harvard Architecture Computer." However, the common core computer 1000 may also employ other types of microprocessor designs in the appropriate application, such as those based upon the Harvard architecture. In this regard, the use of separate data/program bus structures in a Harvard architecture facilitates the use of a data memory that may be shared with at least one other computer entity without unduly complicating the operational constraints on the microprocessor or adversely affecting its processing throughput.

A fundamental departure in Harvard architecture design is described in "A Massive Multiplexed Super Scalar Harvard Architecture Computer" (commonly assigned Dupree et al., International Application No. PCT/US95/00341, filed on January 10, 1995, published as International Publication No. WO 95/19006 on July 13, 1995) as noted above. This massively multiplexed CPU employs a very wide instruction word format which eliminates the need for microcode decoding or even an instruction register and parallel processing to achieve compound superscalar operations. More specifically, this CPU employs a plurality of independent computational circuits, a separate internal result bus for transmitting the resultant output from each of these computational circuits, and a plurality of general purpose registers coupled to each of the computational circuits. Each of the general purpose registers has multiplexed input ports which are connected to each of the result buses. Each of the general purpose registers also has an output port which is connected to a multiplexed input port of at least one of the computational circuits. Each of the computational circuits is dedicated to at least one unique mathematical function and at least one of the computational circuits includes at least one logical function. At least one of the computational circuits includes a plurality of concurrently operable mathematical and logical processing circuits and an output multiplexer for selecting one of the resultant outputs for

transmission on its result bus. The very long instruction word format uses a series of assigned bit locations to represent the selection and operation codes for each of the CPU components. These selection codes are directly transmitted to each of the CPU components by a program control circuit. A separate data control circuit and data bus are further provided to achieve a Harvard architecture design for the CPU.

As illustrated in Figure 1, the shared memory system 1016 includes a shared data memory circuit 1042 which has the same 40-bit word width as the private data memory circuit 1030 of the H2 microprocessor 1014. In some of the Figures, word width is indicated by brackets [ ]. The H2 microprocessor 1014 may either write to or read from the shared data memory circuit 1042 or the private data memory circuit 1030 depending upon the appropriate control signals, such as a memory enable signal and a write/read opcode bit. In the present embodiment, the H2 microprocessor 1014 transmits these memory control signals (for example, the shared active opcode bit) to the HI/O microprocessor 1012, which is responsible for deciding which device will actually be given access to the shared data memory circuit 1042. However, with such a configuration, it is preferred that the H2 microprocessor be given priority to the shared data memory circuit 1042 over the HI/O microprocessor 1012.

The data memory circuit 1030 is characterized as private; it is considered private by the H2 microprocessor 1014. However, other entities may access this memory as well. In other words, as long as access to the data memory circuit 1030 is transparent to the H2 microprocessor 1014, other external computer entities may also have access to this memory.

As with the other memory systems discussed thus far, the shared memory system 1016 includes an address decoder/buffer circuit 1044. Similarly, the shared memory system 1016 also includes a transceiver circuit 1046. The address decoder/buffer circuit 1044 is used to extract an 18-bit address from the 24-but wide data address bus 1020 and store this address in a latch. The transceiver circuit 1046 is used to control the direction, destination and source of data transfer to or from the shared data memory circuit 1042 in response to the write/read control signal controlled by the HI/O microprocessor 1012 (namely the H2SharedR/W signal shown in Figure 2). In one embodiment according to the present invention, the address decoder/buffer circuit 1044 is comprised of three FCT157 muxes and an FCT157 3-to-8 decoder circuit and FCT244 buffers, all from Integrated Device Technology, Santa Clara, California. Similarly, the

transceiver circuit 1046 may be comprised of five FCT245 transceiver circuits from Integrated Device Technologies. However, as with the other components specifically referenced herein, it should be appreciated that other suitable circuit devices could be employed in the appropriate application.

The HI/O microprocessor 1012 is also shown in Figure 1 to include its own private memory system 1048 employing private program and data memory circuits 1049. While the HI/O microprocessor 1012 could be constructed to have a Harvard architecture like that of the H2 microprocessor 1014, the presently preferred HI/O microprocessor utilizes a multiplexed program/data bus structure. In this regard, the multiplexed program/data bus structure is comprised of the 40-bit wide program and data bus 1050 and the 20-bit wide program and data address bus 1052. Such a multiplexed bus structure was employed in order to reduce the number of input/output pins needed by the HI/O microprocessor 1012 in light of the significant number of input/output circuits designed into the HI/O microprocessor, as will be discussed more fully below.

The HI/O microprocessor 1012 also includes a separate 24-bit wide H2_HI/O address bus 1054 for use in accessing either the shared memory system 1016 or the program memory system 1028 of the H2 microprocessor 1014. The separate address bus is used in the HI/O microprocessor 1012 to enable the HI/O microprocessor to complete a shared memory operation (delayed due to instruction skew and possible access by the H2 microprocessor 1014), and still allow the HI/O microprocessor to execute other instructions. This works as follows. Once the H2 microprocessor 1014 has completed its access to the shared data memory circuit 1042, an H2 opcode bit is cleared, informing the HI/O microprocessor that H2 will not be doing a shared memory operation during the next instruction period, and HI/O is clear to initiate a shared access of its own. The HI/O microprocessor 1012 preferably employs a 40-bit opcode which corresponds to the data word width of the H2 microprocessor 1014, as well as a one-third segment of the 120-bit instruction word format employed in the H2 microprocessor 1014. It should be understood that the design of a context-cycling microprocessor according to the present invention is not limited to any particular data/address bit width combination. While the HI/O microprocessor 1012 could also be provided with a Queue memory system like the Queue memory system 1038 for the H2 microprocessor 1014, a Queue memory system is more advantageous for the H2 microprocessor, due to the regularity in which the H2 microprocessor calls routines on a repeated basis.

In order for the HI/O microprocessor 1012 to access either the shared memory system 1016 or the program memory system 1028 of the H2 microprocessor 1014, a registered transceiver 1056 is also connected to the program and data bus 1050. In one embodiment according to the present invention, the registered transceiver is comprised of five 29FCT52 devices from Integrated Device Technology. This particular transceiver has a pair of internal latches which are interconnected front-to-back for storing and controlling the bidirectional flow of data. The HI/O microprocessor 1012 is capable of loading the program memory circuit 1032 of the H2 microprocessor 1014 with a programmed set of instructions through the circuit created by program and data bus 1050, the registered transceiver 1056, the H2_HI/O data bus 1058, the data transceiver 1060 and the program data bus 1022, with the addresses being transmitted on the H2_HI/O address bus 1054. Similarly, the HI/O microprocessor 1012 is capable of reading input and output signal data stored in the shared data memory circuit 1042 through the circuit created by program and data bus 1050, the registered transceiver 1056, the H2_HI/O data bus 1058, the transceiver circuit 1046 and the shared data bus 1062 with the addresses being transmitted on the H2_HI/O address bus 1054 (through address decoder/buffer circuit 1044 to shared address bus 1061).

Figure 1 also shows that another data transceiver 1064 may be connected to the HI/O program and data bus 1050. The transceiver 1064 is used to facilitate external access to the HI/O's program and data bus 1050, such as through the HI/O external memory mapped data bus 1066. For example, such external access may be used for memory mapped devices, such as a debug panel (for reading a key depressed on the panel or writing to an LCD display).

Before proceeding with a further description of the context-cycling processor, a deeper understanding of the bus structure and memory architecture may be helpful. In Figure 10A a detailed block diagram of the multiplexed program/data bus structure is shown; and in Figure 10B, a detailed block diagram is shown of the bank switching functionality whereby memory may be rapidly bank switched by changing the value of bits 23 through 20 of internal memory mapped register at local RAM address FA.

Referring to Figure 10A, the GP1 register and the program counter are multiplexed to supply the data and PC addresses, respectively, to external RAM. Note how the program counters of different contexts 220 and the GP1 registers 222 are ultimately multiplexed through multiplexer 224. The data read from RAM memory 226 is distributed to the opcode latch 228 and the data read latch 230. In this embodiment when the processor is

accessing memory in the internal ROM range of program memory (above FFFE0) a value from onboard ROM 232 is substituted as the opcode. Bank switching is implemented in the presently preferred embodiment. Referring to Figure 10B the circuitry to accomplish this is illustrated. The bank switching information is supplied as the output of multiplexer 224. Bank switching is used to allow quick switching to a different block of memory within the HI/O processor. This allows parallel programs to be available concurrently inside the HI/O processor. To toggle between these two programs all that must change is the 4 bits (bits 20-23) and these bits are added to the upper 4 bits of the program counter of every instruction that runs. By writing to these 4 bits, the system automatically bank switches, allowing the other parallel program to operate. This enhances the speed of the system in a program load situation.

A program and data memory control circuit 1122 is provided for controlling the HI/O private memory system 1048 (shown in phantom in Figure 2). Similarly, a shared data and H2 program memory control circuit 1124 is provided for controlling both the shared memory system 1016 and the program memory system 1028 of the H2 microprocessor 1014 (shown together in phantom in Figure 2). In this regard, each of the memory system control signals generated by the HI/O microprocessor 1012 are identified in Figure 2. The HI/O microprocessor 1012 also includes several other circuits, which are illustrated in Figure 2, but are best seen in Figure 3.

Referring to Figure 2, a simplified block diagram of the HI/O microprocessor 1012 is shown. In this regard, the HI/O microprocessor 1012 includes a substantial number and variety of input and output circuits which are embedded into this single chip central processing unit. While the principles of the present invention are not limited to a single chip implementation, the ability to include all of the I/O functionality of the HI/O microprocessor 1012 into a single integrated circuit chip is considered to be advantageous. In one embodiment according to the present invention, the HI/O microprocessor 1012 may be constructed through the use of a large scale Application Specific Integrated Circuit (ASIC). An ASIC is a type of integrated circuit which includes a significant number of logic gates that may be connected together to perform specific circuit functions. For example, the HI/O microprocessor 1012 may be embodied in a 391 pin package, 100K ASIC device by LSI Logic Corp., Milpitas, California. This particular ASIC device contains well over 100,000 gates that may be combined together to form a variety of different circuits. However, it should be appreciated that other suitable integrated circuit

technologies may be employed to construct HI/O microprocessor 1012 in the appropriate application, including a fully custom integration.

With respect to its input/output circuit capabilities, the HI/O microprocessor 1012 includes a set of four "small computer system interface" SCSI circuits 1100-1106. A SCSI circuit is a high speed parallel interface that generally conforms to the X3T9.2 standard promulgated by the Committee of the American National Standards Institute (ANSI). Each of these SCSI circuits may be used to connect the HI/O microprocessor 1012 to a number of external peripheral devices, such as magnetic disk drives, optical disk drives and printers. The HI/O microprocessor 1012 also includes a set of ten serial input/output circuits 1108 for 8-bit serial communication with such devices as modems, a mouse, keyboards, printers and even other computers. These serial circuits may also be used to gather input signals from and distribute output signals to a network of remote field computer units, such as the type of multitiered network described in the Glaser et al., International Application No. PCT/US93/02253, filed on March 15, 1993, published as International Publication No. WO 93/20488 on October 14, 1993, and entitled "Process Control Interface System Having Triply Redundant Remote Field Units."

The HI/O microprocessor 1012 further includes a set of six serial and shared memory management unit (SASMMU) circuits 1110. The SASMMU circuits 1110 are parallel-to-serial and serial-to-parallel converters for transmitting and receiving information at a serial baud rate of 28.782 mega baud. At this extremely high baud rate, a 40-bit word will be received every 1.4 microseconds. Separate SASMMU circuits 1110 are used for transmitting and receiving in order to maximize the information transfer capabilities of the HI/O microprocessor 1012. Details of these circuits are shown in Figures 15A-15E.

Referring to Figures 15A-15E, the SASMMU circuits are seen in more detail. Figure 15A shows a SASMMU input register 112 and a SASMMU output register 114. The interrupt multiplexers for a SASMMU receiver are shown in Figure 15B at 116. The interrupt multiplexer for a SASMMU transmitter is shown at 118 in Figure 15C. Note that program data bits 37 and 38 of a COMBO instruction type are used (for both transmitter and receiver) to select what the interrupt source will be until changed again. (A COMBO instruction is one of several different types of opcode instructions described below.)

The context cycling processor assigns priorities to different contexts to determine which context should be run next. The different priorities ensure that no data are

lost. The SASMMU contexts are ordinarily assigned a very high priority because of the very high serial baud rate. However, the SASMMU contexts can programmatically reduce their effective priority to allow other lower priority contexts to utilize processor cycles.

Figure 15D shows how SASMMU contexts can effectively lower their priority by choosing an interrupt source referred to as "no high priority." Normally, SASMMUs are considered very high priority (due to the very high data transfer rates). By running SASMMU contexts, other contexts of lower priority can be effectively blocked from performing any operations. To handle this, the SASMMU circuits are capable of selecting a lower priority operation when low data rate operations are being performed. The logic circuitry of Figure 15D shows how this no high priority signal is generated. Figure 15E shows how one of the CP bits (context purpose bits) of the SASMMU context is generated. In this case CP bit 19 is illustrated. This is the end-of-message bit. Two successive stop bits generates this EOM signal. The end-of-message signal can be used as an interrupt, or as a CP bit. The CP bits of a context are generally used for configuration of hardware assets associated with that particular context.

When the common core computer 1000 is used in a process control computer, it is preferred that the process control computer actually comprises a set of three actively redundant common core computers, which make independent process control decisions on mutually exchanged input information from the field. In such process control computers, the SASMMU circuits 1110 may advantageously be used to provide communication between each of the redundant common core computers. To support redundancy, a substantial number of input signals may need to be mutually exchanged between the redundant common core computers in a very limited amount of time during a precise time interval in an overall process control cycle. For example, in one preferred method of process control, a 1 second cycle is provided, during which all of the following must occur: receiving and mutually exchanging over 1,000 input signals, arbitrating between these input signals, making process control decisions based upon the arbitrated input signal values, mutually exchanging the output signal values which represent these process control decisions and transmitting all of the necessary output signals to the field. While the H2 microprocessor 1014 is preferably used to make the needed process control decisions, the HI/O microprocessor 1012 is also capable of being programmed to execute the instructions for these types of decisions. For example, while the H2 microprocessor could be programmed to arbitrate between input signals received from sensors in the field, it is

preferred that the HI/O microprocessor 1012 be utilized for this purpose, as its hardware-based arbitration is capable of faster execution. The arbitration circuitry to accomplish this is shown in Figure 16.

Referring to Figure 16, the arbitration circuitry arbitrates data from triply redundant computers, each supplying a separate data word. In Figure 16 these are depicted as word 1, word 2 and word 3, as shown at 120. Ideally, all three word values should be the same, if all three computers and data sources are performing properly. If all three word values are not the same, then the arbitration selector circuitry 122 determines which computer or data source or a combination thereof is out of tolerance. A process control system will generally operate on both analog and digital signals. The analog signals are first converted to digital signals through appropriate analog to digital conversion circuitry. Nevertheless, the converted values still represent analog values. In Figure 16 there is a separate analog path 124 and digital path 126. Analog values (digitally represented) are fed to processor block 128 whereas digital values are fed to processor block 130. Processor block 128 compares each of the word 1, word 2 and word 3 values with one another. Processor block 130 performs a similar function using AND gates and OR gates to determine the lowest signal of each bit position on line 132 and the digitally triple arbitrated value of each bit position on line 134. A bit by bit compare is performed by the arbitration circuitry and the selector circuitry 122 arbitrates based on 2 out of 3 wins. Selector circuitry 122 can also be set to arbitrate to take the lowest digital or analog value. In some cases the lowest digital value is considered the best default value to select (as it typically will represent a valve in either the open or closed state that is a preferred state when there has been a system malfunction). The analog/digital mode of the incoming data is set on line 136. Because analog values can be expressed as either 40 bit words or 16 bit words, a bit compensation circuit 138 is provided to fill out 16 bit words into their 40 bit equivalent.

The data in Table I lists the result for each different possible input combination to selector circuit 122. The result is used by the system to indicate when the arbitration circuitry has detected a disagreement.

Table I

3/2-/1/10 In-Service	Word1 & Word2 Agree ?	Word2 & Word3 Agree ?	Word3 & Word1 Agree ?	Word1 > Word2 ?	Word2 > Word3 ?	Word3 > Word1 ?	Own/-Lowest-Highest	Answer
0	x	x	x	x	x	x	x	word 1
1	x	x	x	x	x	x	x	word 1
2	0	x	x	0	x	x	own	word 1
2	0	x	x	0	x	x	lowest	word 1
2	0	x	x	0	x	x	high	word 2
2	0	x	x	1	x	x	own	word 1
2	0	x	x	1	x	x	lowest	word 2
2	0	x	x	1	x	x	high	word 1
2	1	x	x	x	x	x	x	word 1
3	0	0	0	x	x	x	own	word 1
3	0	0	0	0	0	1	lowest	word 1
3	0	0	0	0	0	1	highest	word 3
3	0	0	0	0	1	0	lowest	word 3
3	0	0	0	0	1	0	highest	word 2
3	0	0	0	0	1	1	lowest	word 1
3	0	0	0	0	1	1	highest	word 2

3/2-/1/10 In-Service	Word1 & Word2 Agree ?	Word2 & Word3 Agree ?	Word3 & Word1 Agree ?	Word1 > Word2 ?	Word2 > Word3 ?	Word3 > Word1 ?	Own/-Lowest-Highest	Answer
3	0	0	0	1	0	0	lowest	word 2
3	0	0	0	1	0	0	highest	word 1
3	0	0	0	1	0	1	lowest	word 2
3	0	0	0	1	0	1	highest	word 3
3	0	0	0	1	1	0	lowest	word 3
3	0	0	0	1	1	0	highest	word 1
3	0	0	1	x	x	x	x	word 1
3	0	1	x	x	x	x	x	word 2
3	1	x	x	x	x	x	x	word 1

With reference again to Figures 1 and 2, the HI/O microprocessor 1012 further includes a clock circuit 1114, which is responsive to a 144 MHZ oscillator and an external synchronizing signal, such as the "universal time clock" (UTC) pulse signal 1116. In this regard, the UTC pulse signal operates as an interrupt signal for the timed context of the HI/O microprocessor 1012. The UTC pulse signal 1116 is beamed from a satellite in the U.S. government's "global positioning system" (GPS) every second, and the clock circuit 1114 preferably is synchronized within 1 microsecond to the UTC pulse signal. The clock circuit 1114 is used to generate a set of internal clock signals (Mst, Eit, Dst and Dat), as well as to produce a number of timing-dependent signals which are used by external devices (such as the memory systems). In the common core computer 1000, the H2 microprocessor 1014 receives its clock signals from the HI/O microprocessor 1012, such as the fundamental 72 MHZ clock signal 1118. The HI/O microprocessor 1012 also includes a separate 24-bit timer circuit 1120 which is used to set timeout values for the processing contexts to be discussed below. The HI/O microprocessor 1012 is also capable of

temporarily suspending the operation of the H2 microprocessor 1014 by applying an Immobilize signal, without necessarily affecting the operation of the H2 microprocessor after the Immobilize signal has been restored to its normally inactive state. In this regard, the Immobilize signal may be used to prevent a clock-enabling CLEAR signal on one of the H2 microprocessor clock signals. Referring to Figure 9, a timing diagram of some of the clock signals used by the HI/O microprocessor 1012 is shown.

Figure 9 shows the manner in which processor cycles are utilized in the presently preferred embodiment. As noted above, the clock circuit 1114 generates the internal clock signals Mst, Eit, Dst and Dat. These are shown in Figure 9. The presently preferred embodiment subdivides the fundamental 112 ns (nanosecond) clock period into two components, designated as cycle 1 and cycle 2, each being of equal duration. The fundamental clock period is broken into two periods to provide timing for (a) the active context and (b) the new context. As will be more fully explained, the system makes a distinction between the "active" context and the "new" context. The “new” context becomes the “active” context during the next cycle. The “active” context is the context which is currently running (executing) and the “new” context is the context that will run (execute) next. As illustrated at 68 the value in register GP1 of the “active” context is used as an address to access data memory (for data memory operations). This value is allocated to cycle 1. As depicted at 70, the value in the program counter of the “new” context may serve as an address to access program memory (to fetch the op code to be executed during the next instruction period). This value is allocated to cycle 2. Although not fully illustrated in Figure 9, the next cycle 1 would immediately follow the illustrated cycle 2. Thus, as depicted at 72, the GP1 register would again be used to address data memory. The instruction performed on that data memory would be the one fetched at 70. When used as an input/output coprocessor for the H2 microprocessor, it is desirable to skew the respective clocks of the HI/O and H2 processors by several nanoseconds. This skew between the clocks makes it easier to access shared memory. Figure 17 shows how this is accomplished in the preferred embodiment. Figure 17 is similar to Figure 9. It illustrates cycle 1 and cycle 2 of the HI/O processor at 74 and the corresponding timing of the H2 processor at 76. Note the H2 processor does not perform context switching nor does it have a shared program/data memory, and therefore it does not require having its clock period broken into two cycles.

Turning now to Figure 3 (Figures 3A and 3B, collectively), a detailed block diagram of the HI/O microprocessor 1012 is shown. Figure 3 illustrates the circuits which have already been introduced including arbitration circuit 1126. As previously noted, the arbitration circuit is used to arbitrate between corresponding signals from up to three sources (digital and digitally-represented analog data). For example, these sources could represent the input signals received from sensors in the field and the corresponding input signals received from the sibling microprocessors of the other two common core computers in a process control node.

A bootstrap ROM circuit 1128 is provided to store a basic operating system routine needed to boot up the HI/O microprocessor 1012 on start-up. An opcode decode and control circuit 1130 is provided for decoding the 40-bit opcode word used by the HI/O microprocessor 1012. The HI/O microprocessor 1012 electrically enables the circuits specified in the opcode instruction. A block diagram of a portion of the opcode decoder is shown in Figure 13. Note that the memory protect range value is compared with the value on the GP1 register bus by comparator 236. The output of this comparator controls the input to the decode modifier 238 to control whether memory protection is enabled or disabled for a particular location. The presently preferred embodiment handles several different instruction types designated LIM (Long Immediate Math), LIC (Long Immediate Compare) and COMBO. These are decoded from the op code and selected as inputs to the decoder modifier 238. An example of the instruction types utilized by the HI/O microprocessor 1012 is shown in Figures 14A and 14B. Figure 14A shows the COMBO instruction at 38 and the Long Immediate Math instruction at 40. Figure 14B shows the Long Immediate Compare instruction at 42. In Figures 14A and 14B the numbers 39...0 at the top of each chart represent the bit field widths. These charts show how the presently preferred embodiment decodes bits in the respective bit fields. By way of example (referring to the Long Immediate Math instruction set at 40) under the main math unit operation category (bits 33...31), a 0 value would specify the AND operation, a 1 value would specify the OR operation, a 2 value would specify the exclusive OR operation, and so forth.

When referring to Figures 14A and 14B, 40 bit operands are configured as follows. Bit 39 represents the sign, where 1 denotes a negative number; bits 38-15 represent 24 bits of the integer value; and bits 14-0 represent 15 bits of the fraction. All data source values of less than 40 bits are automatically padded with zeros. Further, 20 bit source values (used in 40 bit operations) are treated as positive integers and are aligned

with the least significant bit (LSB) of all the 20 bit values at the bit 15 position of the 40 bit operand. Deviations from this practice are specifically noted in the Table below. When referring to the Table below, note that the PIMMED/GP1 field acts as an address for local RAM and memory mapped registers. There is no interrupt source modification in the LIM or LIC instruction types. Thus the interrupt source (chosen during the last COMBO instruction executed) is maintained while executing either Long Immediate instruction type. Interrupt source selection is forced to the "00" (binary) selection (representing "Always") when the context is descheduled or the processor goes through a hardware reset. The Table below explains the abbreviations used in Figures 14A and 14B.

TABLE

COMBO instruction type:

b39	INST TYPE	Instruction Type. "0" indicates the instruction type is COMBO
b38..37	INT CND	Interrupt Condition. Selects, via context's interrupt mux, the 1 (of 4) interrupt condition requisite for context to execute next instruction. Number of conditions (up to 4) along with type of conditions, vary with context type. Selections are: CD0 (condition0 of interrupt mux = "Always" for all 24 contexts) CD1 (condition1 of interrupt mux; condition varies based on context) CD2 (condition2 of interrupt mux; condition varies based on context) CD3 (condition3 of interrupt mux; condition varies based on context)
b36..35	JMP TGT	Jump Target. Value with which context's PC will be updated at end of instruction period in which this opcode is executed. Selections are: PC+ 1 (PC of current instruction + 1)

		PC+ 1+ IM (same as previous + PIMMED value, located at b7..0 of the opcode, PIMMED to be interpreted as 1 bit sign, 7 bits magnitude)
		GP3 (bits 34..15 of value in context's GP3 register RAM location)
		0 (Zero)
b34..33	COMP S2	Comparitor Source2 operand. Selections are:
		GP2 (value in context's GP2 register RAM location)
		IMG1 (value of PIMMED/GP1 bus - the value on this 20-bit bus is equal to the right-justified value of the PIMMED field, as 8 bits integer, unless PIMMED = FFh, where the bus assumes the value of GP1)
		GP3 (value in context's GP3 register RAM location)
		0 (Zero)
b32..30	COMP OP	Comparitor Operation, in form (COMP S1.COMP OP.COMP S2, like GP2.GTE.LRR). Selections are:
		EQ (equal)
		NEQ (not equal)
		GT (greater than)
		GTE (greater than or equal)
		LT (less than)
		LTE (less than or equal)
		S1@S2T (bit of comparitor source1 operand, at bit position equivalent to comparitor source2 operand integer value, is TRUE)

		S1@S2F (bit of comparitor source1 operand, at bit position equivalent to comparitor source2 operand integer value, is FALSE)
b29..28	COMP S1	Comparitor Source1 operand. Selections are:
		GP2 (value in context's GP2 register RAM location)
		LRR (value on multiplexed Local RAM/(mmed) Registers bus)
		MVB (value on Move Bus)
		MMO (value on Main Math Output bus)
b27..25	MMU S2	Main Math Unit Source2 operand. Selections are:
		GP1 (value in context's GP1 register RAM location)
		GP2 (value in context's GP2 register RAM location)
		GP3 (value in context's GP3 register RAM location)
		LRR (value on multiplexed Local RAM/(mmed) Registers bus)
		+1 (integer value of +1)
		1's (all 40 bits of operand forced to 1's)
		PIMMED (8-bit Program Immediate field of the present opcode, used as integer)
		0 (Zero)
b24..21	MMU OP	Main Math Unit Operation, in form (MMU S1.MMU OP.MMU S2, like GP1.EXOR.1's). Selections are:
		AND (logical AND)
		OR (logical OR)
		EXOR (logical EXOR)

+ (addition)

- (subtraction)

* (integer 20-bit by 20-bit multiply; result MSB -> LSB will be in bit positions 14..0, 39..15.)

/ (software assisted, successive approximation divide - 40-bit numerator, 20-bit denominator, 20 bit quotient; numerator operand selected as COMP S2, denominator operand as MMU S2, and interim quotient as MMU S1. The PIMMED value , interpreted as an 8-bit, positive integer, will be used to point to each bit position of the temporary quotient to be calculated)

S1 IF CO ELSE S2 (MMU Source1 operand will be the MMU output IF the output of the comparitor is TRUE, else the value of the MMU Source2 operand will be the output)

S1 BSET@S2 (value of MMU Source1 operand with bit set at position pointed to by integer value of MMU Source2 operand value)

S1 BCLR@S2 (value of MMU Source1 operand with bit cleared at position pointed to by integer value of MMU Source2 operand value)

S1 BSUB@S2 (value of MMU Source1 operand with bit substituted with at position pointed to by integer value of MMU Source2 operand value)

ROTATE/MERGE 8 (value of rotated Source1 operand merged with Source2 operand on byte boundaries. Bits 7..5 of the PIMMED field furnish the code for 1 of 8 discrete rotation amounts. Bits 4..0 of the PIMMED field are the byte select bits, a "1" translated as "from rotated Source1 operand", a "0" as "from Source2 operand")

		CHKSUM (new checksum calculation with the Source1 operand as the input data value and the Source2 operand as the previous checksum value)
		ENCR/DECR (encrypt/decrypt data with the Source1 operand as the data to be encrypted/decrypted, and the Source2 operand as the encryption/decryption dynamic key to use, with the static key provided by a Local RAM/(mmed) register)
		DYNAMIC KEY (hardware permute the Dynamic Key value, provided as the Source1 operand)
		PARITY (using Source1 operand only, exor bits 8..0, and substitute this 1 bit value for bit 7 of the Source1 operand, passing all other 39 bits of the Source1 operand unaltered)
b20..18	MMU S1	Main Math Unit Source1 operand. Selections are:
		GP1 (value in context's GP1 register RAM location)
		GP2 (value in context's GP2 register RAM location)
		GP3 (value in context's GP3 register RAM location)
		LRR (value on multiplexed Local RAM/(mmed) Registers bus)
		CP (value in context's "Context Purpose", CP, register)
		MVBUS (value on the Move Bus)
		PC+ 1 (PC of current instruction + 1)
		RAWIN (raw, un-autodecrypted data from context's input register)
b17	GP3	GP3 update value. Selections are:
		NOÆ (no change to context's GP3 value)
		MMO (update context's GP3 value to that on Main Math Output bus)

b16..15	GP2	GP2 update value. Selections are: NOÆ (no change to context's GP2 value) MMO (update context's GP2 value with bits 34..15 of the value on the Main Math Output bus) - 1 (decrement context's current GP2 value by 1) MVB (update context's GP2 value with bits 34..15 of the value on the Move Bus)
b14..13	GP1	GP1 update value. Selections are: NOÆ (no change to context's GP1 value) MMO (update context's GP1 value with bits 34..15 of the value on the Main Math Output bus) + 1 (increment context's current GP1 value by 1) LRR (update context's GP1 value with bits 34..15 of the value on the multiplexed Local RAM/(mmed) Registers bus)
b12..8	MOVE OPT.	Move Bus Option. This portion of the opcode handles 4 functions simultaneously, in the form W.X.Y.Z, where W is the source operand to be placed on the Move Bus, X is the destination of the Move Bus value, Y is the Local RAM/(mmed) Register update source, and Z is the CP register update source. Abbreviations are: DM - Data Memory, DI - Context's Default Input Register, DO - Context's Default Output Register, LR - multiplexed Local RAM/(mmed) Registers bus, G3 - Context's GP3 value, MB - Move Bus, NC - No Change, and MM - Main Math Output bus. The 32 selections are as follows: 0. DM.MB.NC.NC 1. DM.MB.MM.NC

2. DM.MB.MB.NC
3. DM.DO.NC.NC
4. DM.DO.MM.NC
5. DM.DO.MB.NC
6. DM.MB.NC.MM
7. DM.DO.NC.MM
8. DI.DM.NC.MM
9. DI.MB.NC.NC
10. DI.MB.MM.NC
11. DI.MB.MB.NC
12. DI.DM.NC.NC
13. DI.DM.MM.NC
14. DI.DM.MB.NC
15. LR.MB.NC.NC
16. LR.MB.MM.NC
17. LR.DM.NC.NC
18. LR.DM.MM.NC
19. LR.DO.NC.NC
20. LR.DO.MM.NC
21. MM.DO.NC.NC
22. MM.DO.MM.NC
23. DI.MB.NC.MM

24. G3.DM.NC.MM

25. G3.DM.NC.NC

26. G3.DM.MM.NC

27. G3.MB.MB.NC

28. G3.DO.NC.NC

29. G3.DO.MM.NC

30. G3.DO.MB.NC

31. G3.DO.NC.MM

b7..0	PIMMED	Program Immediate field. 8 bits used in a variety of ways, as explained above.

Long Immediate Math (LIM) instruction type:

b39..38	INST TYPE	Instruction Type. "10" (binary) indicates the instruction type is Long Immediate Math
b37	JMP TGT	Jump Target. Value with which context's PC will be updated at end of instruction period in which this opcode is executed. Selections are: PC+ 1 (PC of current instruction + 1) MMO (bits 34..15 of the value on the Main Math Output bus)
b36..34	MMU S2	Main Math Unit Source2 operand. Selections are: GP1 (value in context's GP1 register RAM location) GP2 (value in context's GP2 register RAM location) GP3 (value in context's GP3 register RAM location) LINT0 (20 bits of Long Immediate field placed into bit positions 34..15, and padded with zeros)

		LINT1 (20 bits of Long Immediate field placed into bit positions 34..15, and padded with ones)
		LFRC0 (20 bits of Long Immediate field placed into bit positions 39..35 and 14..0, and padded with zeros)
		LFRC1 (20 bits of Long Immediate field placed into bit positions 39..35 and 14..0, and padded with ones)
		0 (Zero)
b33..31	MMU OP	Main Math Unit Operation, in form (MMU S1.MMU OP.MMU S2, like GP1.OR.LFRC0). Selections are:
		AND (logical AND)
		OR (logical OR)
		EXOR (logical EXOR)
		+ (addition)
		- (subtraction)
		* (integer 20-bit by 20-bit multiply; result MSB -> LSB will be in bit positions 14..0, 39..15.)
		ROTATE/MERGE 40 (value of rotated Source1 operand merged with Source2 operand on 4-bit, nibble boundaries, with LS nibble defined as bit positions 2..0 including bit 39 as the LS bit. Bits 15..10 of the Long Immediate field, LIMMED, furnish the continuous, right-rotation amount (rotated right 0 to 63 times). Bits 9..0 of the LIMMED field are the nibble select bits, a "1" translated as "from rotated Source1 operand", a "0" as "from Source2 operand")
		S2 (the value of the Source2 operand is output)
b30..28	MMU S1	Main Math Unit Source1 operand. Selections are:

			GP1 (value in context's GP1 register RAM location)
			GP2 (value in context's GP2 register RAM location)
			GP3 (value in context's GP3 register RAM location)
			LINT0 (20 bits of Long Immediate field placed into bit positions 34..15, and padded with zeros)
			CP (value in context's "Context Purpose", CP, register)
			MVBUS (value on the Move Bus)
			PC+1 (PC of current instruction + 1)
			RAWIN (raw, un-autodecrypted data from context's input register)
	b27..26	GP3	GP3 update value. Selections are:
			NOÆ (no change to context's GP3 value)
			MMO (update context's GP3 value to that on Main Math Output bus)
			PC+ 1 (PC of current instruction + 1)
			MVBUS (value on the Move Bus)
	b25	GP2	GP2 update value. Selections are:
			NOÆ (no change to context's GP2 value)
			MMO (update context's GP2 value with bits 34..15 of the value on the Main Math Output bus)
	b24	GP1	GP1 update value. Selections are:
			NOÆ (no change to context's GP1 value)
			MMO (update context's GP1 value with bits 34..15 of the value on the Main Math Output bus)

b23..20 MOVE OPT. Move Bus Option. This portion of the opcode handles 3 functions simultaneously, in the form P.Q.R, where P is the source operand to be placed on the Move Bus, Q is the destination of the Move Bus value, and R is the CP register update source. Abbreviations are: DM - Data Memory, DI - Context's Default Input Register, DO - Context's Default Output Register, G3 - Context's GP3 value, MB - Move Bus, NC - No Change, and MM - Main Math Output bus. The 16 selections are as follows:

0. DM.MB.NC
1. DM.DO.NC
2. DI.MB.NC
3. DI.DM.NC
4. DI.DO.NC
5. MM.DO.NC
6. G3.DM.NC
7. G3.DO.NC
8. DM.MB.MM
9. DM.DO.MM
10. DI.MB.MM
11. DI.DM.MM
12. DI.DO.MM
13. MM.DO.MM
14. G3.DM.MM
15. G3.DO.MM

b19..0	LIMMED	Long Immediate field. 20 bits used in a variety of ways, as explained above.

Long Immediate Compare (LIC) instruction type:

b39..38	INST TYPE	Instruction Type. "11" (binary) indicates the instruction type is Long Immediate Compare
b37..36	JMP TGT	Jump Target. Value with which context's PC will be updated at end of instruction period in which this opcode is executed. Selections are: PC+ 1 (PC of current instruction + 1) PC+ 1+ IM (same as previous + PIMMED value, located at b27..20 of the opcode, PIMMED to be interpreted as 1 bit sign, 7 bits magnitude) GP3 (bits 34..15 of value in context's GP3 register RAM location) LI (value of 20-bit Long Immediate field)
b35..34	COMP S2	Comparitor Source2 operand. Selections are: GP2 (value in context's GP2 register RAM location) IMG1 (value of PIMMED/GP1 bus - the value on this 20-bit bus is equal to the value of the PIMMED field, as 8 bits integer, unless PIMMED = FFh, where the bus assumes the value of GP1) GP3 (value in context's GP3 register RAM location) 0 (Zero)
b33..31	COMP OP	Comparitor Operation, in form (COMP S1.COMP OP.COMP S2, like GP2.GTE.LRR). Selections are: EQ (equal)

		NEQ (not equal)
		GT (greater than)
		GTE (greater than or equal)
		LT (less than)
		LTE (less than or equal)
		S1@S2T (bit of comparitor source1 operand, at bit position equivalent to comparitor source2 operand integer value, is TRUE)
		S1@S2F (bit of comparitor source1 operand, at bit position equivalent to comparitor source2 operand integer value, is FALSE)
b30..29	COMP S1	Comparitor Source1 operand. Selections are:
		GP2 (value in context's GP2 register RAM location)
		LRR (value on multiplexed Local RAM/(mmed) Registers bus)
		CP (value in context's "Context Purpose", CP, register)
		LINT0 (20 bits of Long Immediate field placed into bit positions 34..15, and padded with zeros)
b28	LRR	Local RAM/(mmed) Registers update value. Selections are:
		NOÆ (no change to addressed Local RAM/(mmed) Register's value)
		LINT0 (20 bits of Long Immediate field placed into bit positions 34..15, and padded with zeros)
b27..20	PIMMED	Program Immediate field. 8 bits used in a variety of ways, as explained above.

b19..0	LIMMED	Long Immediate field. 20 bits used in a variety of ways, as explained above.

In reference to Figure 3B, the HI/O microprocessor 1012 also includes a context management circuit 1132 for controlling each of the 24 contexts which are arranged within the HI/O microprocessor. In this regard, a separate processing context is assigned for each of the input/output circuits discussed above. For example, six processing contexts are dedicated for the six SASMMU circuits 1110 (Figure 3A), while ten processing contexts are dedicated for the ten serial circuits 1108 and so forth. At the heart of the context management circuit is an interrupt priority encoder.

Referring to Figure 6, a detailed block diagram of an interrupt priority encoder circuit 1210 is shown. The interrupt priority encoder circuit 1210 forms part of the context management circuits 1132, which are shown as a block in Figure 3. Each context has at least one interrupt source and circuitry to select of multiple sources (if it has multiple sources). If no context is active, the processor executes a NOP so that no stored values are altered. The priority encoder receives the interrupts from all different contexts. The priority encoder interprets these, determines which job has the highest priority and then generates the new context code. As previously noted and as will be more fully explained, the system makes a distinction between the "active" context and the "new" context. The “new” context becomes the “active” context during the next instruction period. Demultiplexer 44 demultiplexes the “new” context code into one of 24 decoded lines that can be used for state machine control elsewhere in the system. Similarly, demultiplexer 46 demultiplexes the “active” context’s code into one of 24 decoded lines for a similar purpose.

A detailed block diagram of the interrupt selection circuit for the timed context is shown in Figure 7. The timed context serves a special purpose within the system of coordinating operation of the HI/O processor and the H2 processor. In Figure 7, note that the source of the timed context interrupts can come from a UTC (universal time constant) pulse or a pulse from a global positioning satellite (GPS), for example. Note the UTC pulse input to latch 48. This is used to generate both CP-bit 30 at 50 , and the UTC leading edge interrupt.

The timed context must also generate an interrupt when other contexts deschedule themselves. This descheduling is how the timed context knows when the other contexts are finished with a given task. This eliminates the need to wait for a timeout based

on a time-clocked event, although time-clocked events can be used and are preferred when using the pseudo-queue. The "activity change" interrupt is generated for this purpose at 52. It is generated by monitoring the falling edge on any of the scheduler bits for any of the contexts. The falling edges on all 23 of the bits corresponding to the other contexts are ORed together at OR gate 54.

Returning again to Figure 3A, the input/output circuits of the HI/O microprocessor 1012 also include a set of memory mapped register control circuits, generally designated by reference numeral 1112. As with all of the various input/output circuits disclosed herein, the number of such memory mapped control circuits may be varied as required. However, in the present embodiment, a total of 32 external memory mapped locations are provided, even though all of these locations may not necessarily be decoded. These control circuits perform address decoding in order to minimize the amount of external logic required for devices that are attached to the HI/O microprocessor 1012 through the external memory mapped data bus 1066 (Figure 1). These external devices are memory mapped within the data memory space of the HI/O microprocessor itself. Such external devices, for example, may include a debug panel.

Figures 8A-8F comprise detailed block diagrams which illustrate each of the dedicated registers for the processing contexts. Figure 8A illustrates how the GP1 register RAM is configured. In the preferred embodiment there are 24 locations of GP1 registers that are addressed by the context code ("active" context or "new" context). The update data is input through a multiplexer 56 that is switched according to the type of opcode instruction (e.g., Long Immediate Compare, COMBO). Figures 8B and 8C are similar to Figure 8A but illustrate the general purpose registers GP2 and GP3, respectively. Note that the GP3 register is addressed only by the "active" context's code, since the GP3 value is not prefetched as are the GP1 and GP2 values. Thus there is no need for an address multiplexer. Although the preferred embodiment does not require register GP3 to be addressed by both the "active" context and the "new" context, it would be feasible to implement register GP3 similar to register GP2, if desired. Generally, registers GP1 and GP2 are used as address generators and counters, respectively. These registers are 20 bits wide because that is the width of the address bus in the presently preferred embodiment. Register GP3 is a 40 bit wide random access memory to support the full data width (40 bits) of the current embodiment.

Figure 8D depicts the logic of the program counter. Note that the program counter, implemented by RAM 58, can be loaded from various sources. Multiplexer 60 loads the program counter register RAM 58. The timed context may map the program counters of other contexts into the local RAM space via the addressing circuitry shown generally at 62, thus controlling context program counter initialization. To help in accomplishing this, the timed context uses a register for its PC (rather than a location in the PC RAM) so that it can update the PC RAM location of another context while still updating its own PC (register). This eliminates the need for a double update of the PC RAM in one instruction period. In Figure 8D the short PIMMED (Program Immediate) value of the COMBO instruction (8 bits) is evaluated as a signed displacement (sign/magnitude) when used to modify a PC value. This allows relative jumping 127 locations backward and forward.

Figure 8E illustrates an output register. The illustrated register is 40 bits wide. The register is written-to when gated by the signals supplied to AND gate 64. The signal designated as "Mst" is the main store clock. This signal appears in numerous places throughout the circuitry. It is one of the principal timing signals that controls operation of the HI/O processor. This timing signal and the other principal timing signals will be more fully described in connection with Figure 9.

Figure 8F illustrates a generic CP register configuration. There is a CP register for each context. This register is 40 bits wide, though size varies based on context need. It is updated by using the main math out (MMO) value. The CP register uses the "this context active" signal to ensure that the register is updated at the proper time relative to the context that is "active". The multiplexer 66 supplies a signal on the CP-bus when a context's CP value is used in the main math unit. The active context code is used to select which CP register value is actually used in the main math unit. For instance, if another context is running a routine where it is required to AND out bit 26 of its CP register, the "active" context code (of this running context) will select its own CP value to be fed to the main math unit for ANDing. Thus, (a) its bit 26 gets ANDed out, and then (b) its CP register will be rewritten at the end of the current instruction.

Substantial on-board RAM capacity is also provided for the HI/O microprocessor 1012 through a set of RAM circuits 1138 (Figure 3A) which function as a collection of registers. These RAM circuits 1138 include a local RAM circuit 1140, which may be employed to store up to 128 40-bit words in this embodiment. The RAM circuits 1138 also include a set of registers which are dedicated to the 24 processing

contexts of the HI/O microprocessor 1012. Each of these sets of dedicated registers includes a general purpose register GP1, a general purpose register GP2, a general purpose register GP3 and a program counter PC, each addressed by the active context code in a 24 location RAM. A checksum circuit 1142 (Figure 3B) also provides error checking on external data transmissions for 10 of the contexts (namely the SCSI and SASMMU contexts). A dynamic key RAM 1180 is also used to provide a dynamic key for an encryption algorithm to enhance the security of external data transmissions of some contexts.

Figure 11 shows a more detailed schematic on how the external RAM and program data memory RAM is constructed. The RAM is configured into different banks and there are two pre-decoded chip enable signals (CE) to select which bank is being used. The chip enable signals are supplied at 108 in Figure 11. The presently preferred embodiment implements multiple 64K banks of RAM. The presently preferred embodiment employs a 1 megabyte memory space for the HI/O processor. The chip enables can be selectively decoded to switch on banks of different sizes (64K, 128K, 512K and 1 meg). When operating in the 1 meg mode, (a) the 0-0FFFF chip enable signal will be active and (b) the other chip enable signal will never be active. Figure 11 also illustrates a portion of the move bus discussed below.

The HI/O microprocessor 1012 further includes a main math unit 1134 (Figure 3A) for performing a number of mathematical and logical operations. As shown in Figure 3, the output bus 1135 from the main math unit 1134 is connected to several on-board circuits, including the input/output circuits, such as the SASMMU circuits 1110 and the SCSI circuits 1100-1106. A separate comparator circuit 1136 is provided for performing a number of alternative compare operations. It should be noted that the comparator circuit 1136 is capable of operating concurrently-with and independently-from the main math unit 1134. As a consequence, a single compound instruction may be used to cause the main math unit 1134 to perform an addition operation on one set of input source values, while the comparator 1136 is being commanded to perform one of a number of possible compare operations on its own separate input source values, one of which may be the output of the math unit. Indeed, as shown in Figure 3, the output from the comparator provides one of the selectable input sources for the main math unit 1134 during the same clock cycle. The output from the comparator 1136 could also be used to choose the source signals for the main math unit 1134 during the same clock cycle in which the main math unit conducts a desired computational or logical operation or a combination thereof.

Centrally disposed in Figure 3 is a mux module 1146 (move multiplexer) which represents the combination of a number of circuits used to rapidly move data between the input/output circuits and the main math unit 1134, the comparator circuit 1136, the dedicated registers of the processing contexts, and the other circuits of the HI/O microprocessor 1012 as shown. The mux module 1146 is shown more fully at 90 in the schematic diagram of Figures 12A-12D. The mux module 1146 includes a move bus 109 (Fig 11) which is tristate buffered onto the program and data bus 1050 (Figure 1) of the HI/O microprocessor 1012.

Figures 12A-12D are high level block diagrams showing, among other things, the circuitry found in the mux module. The mux module is also depicted at 1146 in Figures 3A-3B. This circuitry provides a collection of multiplexers for different purposes. For example, the CP multiplexer 78 selects the appropriate CP register based on the active context code. The input multiplexer 80 selects raw data from the context's input registers, also based on the "active" context's code. Also included is the encryption/decryption control module 82 that may be used to perform hardware encryption and decryption of data. The checksum control circuitry 84 generates checksums for data integrity control. Figures 12B-12D show high level block diagrams of the following additional circuitry: local RAM control 86, GP1 control 88, mux module 90, output mux 92, MMU source muxes 94, comparitor source muxes 96, logic analyzer mux 98, GP3 control 100, GP2 control 102, program counter control 104, and shared context control 106.

Referring back to Figure 11, a portion of the move bus 109 is depicted in further detail. The circuitry shows how the output of the move bus can be used as a source of data when writing to external data memory. The output of the move bus is always written to data memory. The processor selects what the source of the data is going to be. The source of data can be general purpose register GP3, main math unit MMU, local RAM/registers or default input. This selection is controlled by the mux module 90 based on the Move Source Selection input.

Before proceeding to discuss the specific circuits which are associated with the processing contexts (illustrated in Figures 4A-8F), an overall discussion of the operation and interaction between the timed and device processing contexts is in order. Each of the external devices (coupled to the HI/O microprocessor 1012) are handled by separate subroutines running on the HI/O microprocessor. Each of these subroutines will run in its own environment (referred to herein as a processing context) and, therefore, does not need to "know of" the existence of other software running on the HI/O microprocessor 1012

(except for the existence of the timed context which acts as a "scheduler routine"). Because several resources are shared between different contexts (such as the computational functions of the main math unit 1134), only one context will be "active" at one time (with "active" meaning that the software instruction of that context is currently being executed). Of course, it could be reasonable for more than one context to be "active" at one time, provided that more than one set of shared resources is also provided.

All processing contexts are preferably controlled by a scheduling mechanism. This mechanism schedules the different processing contexts so that they are coordinated in time (some tasks should run only after other tasks have been completed, while some tasks should be coordinated with tasks in other computer systems), so that over-allocation of resources is avoided. It is recognized that much of this scheduling can be orchestrated at compile time since many of the tasks are done repetitively each second and at the same (relative) time within each second. Scheduling of those tasks would then be done through the "timed context" (the context enforcing a "timeline"). However there will also be some spontaneous tasks (nonrecurring) which have to be scheduled between the recurring ones.

Processing contexts are assigned different priorities and a priority encoder is used to activate the highest "bidding" context. Bidding requires (a) being scheduled, (b) having an interrupt present and (c) having a respective interrupt selected on the interrupt MUX. For a processing context to be scheduled, the timed context must set that processing context's scheduler bit, in order to enable the muxed interrupts of the processing context to be forwarded to the priority encoder. Each scheduler bit of a processing context is co-mapped in that processing context's CP register as well as in the timed context's CP register. A processing context will be bidding when (a) it has been scheduled and (b) it needs some software action. For instance, the processing context dealing with human input devices will normally only be bidding when either the mouse or keyboard has been touched. After this event has been processed (meaning that this processing context has had the highest priority at least once and has executed at least one instruction), this processing context will not ask for service until the next human interaction. Until then, this processing context is said to be "inactive."

Most interrupts from external sources will be handled in a single instruction, so it should be appreciated that rapid context cycling will be enabled. To make this context cycling highly efficient, each processing context is provided with its own set of registers. In the present embodiment these registers preferably include a program counter, three general

purpose registers, a context purpose register (for miscellaneous context hardware configuration) as well as input and output registers. However, it should be understood that a different set of hardware assets may be employed without departing from the principles of the present invention. For example, certain processing contexts may not need input or output registers and a greater or lesser number of general purpose registers could be provided in the appropriate application. The present embodiment uses the same number of general purpose registers for all contexts. When a processing context becomes "active," its program counter and other registers will be available to access or control the shared resources, such as the private program and data memory circuits 1049 shown in Figure 1 and the main math units and comparator (and all other singly provided common use resources).

A processing context can only be "active" when the context has been scheduled, it had the highest priority when priority encoding took place, its op code has been fetched, and it is now executing that op code. An interrupt can be an external event, but some may also be generated through software. Accordingly, it is preferred that each processing context have a set of different interrupt sources, with only one interrupt source selected at a given time. In the present embodiment, one to eight different interrupt sources are provided for each processing context. However, it should be appreciated that a greater or lesser number of interrupt sources may be provided in the appropriate application. As a result, the execution of each instruction effectively sets the condition to execute the next instruction (the interrupt source triggering the execution of the next instruction). The default for this condition would be "always," meaning that the next instruction should be executed regardless of external events, provided that (a) this context is scheduled and (b) it wins priority encoding.

As an example of this process, assume a reception on Serial 5. The timed context schedules Serial 5 several instruction cycles prior to the scheduled transmission of the data from an external source. Serial 5 runs an instruction to set interrupt source to "receive word" and becomes inactive, though still scheduled. When the autonomous state machine of Serial 5 clocks-in a full 8-bit word, (a) the received word interrupt is generated, (b) Serial 5 bids for service, and (c) when it wins, Serial 5 awakens and runs an instruction to empty its input register. Then Serial 5 becomes inactive until the next word comes in. This method processes one 8-bit word in one instruction.

Each processing context may be in one of a plurality of different states. In the present embodiment, each processing context may be in one of 4 possible states, which are summarized below. However it should be appreciated that other arrangements of processing context states may be employed without departing from the principles of the present invention. In the "idle" state, the context is waiting for the scheduler routine to set its scheduler latch. In the "inactive" state, the context has been scheduled, but it has no interrupt pending (that is, it is waiting for some event to occur). In the "bidding" state, the context has been scheduled, it also has an interrupt pending (the event has happened), and it is now competing for service based on the priority of its interrupt.

Priority encoding for the next instruction takes place two instruction periods before the bidding contest winner's op code is executed. When the interrupt selected by a context becomes true, and it eventually wins priority encoding, its opcode (the function of which is usually to service the interrupt) is fetched during the next instruction period. Since the instruction will not be executed until the instruction period after this, the interrupt will still be present when the subsequent priority encoding takes place. This could cause this context to be "active" twice for the same interrupt. To assure that a single "event" (such as receiving a word) does not make that device context "active" for two consecutive instruction periods, a processing context is excluded from priority encoding when it is "new" (that is, when its instruction is being fetched). This guarantees that one instruction of the processing context is executed before that context can bid again. This also means that each processing context will get no more than 50 percent of the instruction periods. This mechanism also avoids the need for a jump pipeline. When a jump is performed within the HI/O microprocessor 1012, the next instruction is the first at the "jumped to" location.

In the present embodiment, processing contexts have preassigned priority levels. For example, those processing contexts requiring the shortest interrupt response time could be assigned the highest priority. However, depending on what a processing context is doing, it might need a less immediate interrupt response than it normally does. So a mechanism is implemented that certain contexts (with preassigned high priorities) can lower their priority level in favor of contexts which are normally lower in priority. Accordingly, it should be appreciated that the HI/O microprocessor 1012 does provide a unique form of dynamic priority reassignment as warranted. A situation can also occur where no processing context is active. In this case, a "NOP" (meaning "no operation") is fetched out of internal bootstrap ROM 1128, so that all registers and memory remain unaffected.

All of the timed functions within HI/O microprocessor 1012 are controlled from the timed context. Based on time, other processing contexts will be scheduled, configurations will be changed, control signals to the H2 microprocessor 1014 generated and so forth. The timed context also governs the control of external devices, such as the debug panel or a crossbar signal communication switch. Indeed, when the HI/O microprocessor 1012 is "reset," all processing context are "descheduled," so that the timed context is the only context running. At RESET, the timed context's program counter "PC" is preset to a predetermined value, which is the first address of the internal bootstrap ROM 1128 program. This bootstrap program is a mini program that loads a slightly more complex program at address zero of the external program and data memory, and then jumps to address zero and begins execution. It should also be noted that the timed context can schedule multiple contexts at the same time up to the maximum number of processing contexts for the HI/O microprocessor 1012. In this regard, the HI/O microprocessor 1012 is provided with a total of 23 processing contexts, not including the timed context. However, it should be understood that the number of processing contexts will depend upon the particular implementation for a context cycling microprocessor according to the present invention.

It should also be appreciated that, in some applications, it would be preferable to create the general schedule for the timed context in advance. Within this general schedule, slots in time should be left to provide for "spontaneous" needs (such as, for example, sporadic communication, and disk transfers). Since the HI/O microprocessor 1012 does the time keeping for the common core computer, it is preferred that this schedule should be enforced by the HI/O microprocessor. This schedule would provide a pseudo-queue list, which should be both application and system dependent. For example, this schedule could contain the times to start transfers from the shared memory system to the private memory system to do both sibling communication and arbitration. As such, this schedule should be coordinated with tasks to be performed by the H2 microprocessor 1014 and coordinated with other computers that communicate with the system. Besides providing the "start time" for a specific context, the timed context pseudo-queue schedule should also be able to control the specific task that a context is going to accomplish. This is achieved by giving the timed context access to all program counters of other processing contexts.

If tasks are to be started at specific times, the context running that task should complete its previous task before it is scheduled again. Every scheduled context will need a certain amount of time to accomplish a specific task. Once this time has elapsed, (a) its task

should be finished and (b) the context should have de-scheduled itself. Various reasons could cause the task not to be completed in the assigned time period (for example, a broken data link or a SCSI problem). In such an event, the task should be interrupted and the processing context should be "reset," so that it can resume proper operation. The timed context will set the timeout periods for each context by a scheduled "timeout" interrupt when the task should be completed. When the interrupt occurs, the timed context will determine whether the task was completed by reading the message status or scheduler latch (latch should have been cleared by that context if finished in time). If the task is unfinished, the timed context will first "deschedule" that context, then preset the PC of that context to an error service routine if appropriate, and then again schedule the context to run the error service routine. After this, this processing context's task should be ready to be rescheduled again. Note that the timed context may schedule or deschedule other contexts, since it has access to the scheduler bits of the 23 other processing contexts (timed context is perpetually scheduled, thus has no need for its own scheduler bit). Each of the 23 other processing contexts, however, may not schedule itself, but only deschedule itself, since these each of these contexts must already be scheduled and "active" to update its scheduler bit.

The H2 and HI/O processors of the common core computer 1000 interact with one another through shared memory and by means of interrupts and other control signals. Figure 18 illustrates this relationship. As previously described, the H2 processor is primarily involved in performing process control applications developed by the process control engineer to accomplish different process control tasks. The HI/O processor is principally involved in handling asynchronous events, such as communication with external devices. Both processors are capable of operating independently of one another, although, in the common core computer configuration, these two processors have a symbiotic relationship, the H2 processor performing most of the mathematical and logical computation functions and the HI/O processor handling communication with the outside world. Although both processors have their own internal clocks, the HI/O processor controls the overall timing of the system. The HI/O processor supplies the H2 processor with its base 72 MHz clock.

Referring to Figure 18, the timed context of the HI/O processor sends timing signals via interrupts to the H2 processor. These interrupts are responded-to by the Dispatcher software routine of the H2 processor. In this way the H2 processor is able to be synchronized with the GPS system. This architecture allows all common core computers to be synchronized to a common time base (via GPS satellite, for example). Such

synchronization is extremely valuable in a process control application, as many processes running concurrently will need to be coordinated possibly on a geographically large, worldwide scale. Note that the HI/O and H2 processors are able to share data through the shared memory that is accessible to both processors. In the presently preferred embodiment, the shared memory comprises a mutual data memory system of both processors. Although the presently preferred embodiment employs Harvard architecture for the H2 processor, this is not a requirement. The invention can be implemented using the Princeton or Von Neumann architecture. As noted above, the Harvard architecture is presently preferred because the illustrated Harvard implementation is able to execute instructions more quickly.

As illustrated in Figure 18, the program memory (PMem) is dedicated principally to the handling of the different functional contexts. Essentially, these functional contexts break down commonly performed input/output, arbitration, and timing tasks into separate functional modules that may be rapidly cycled-through without undue computational overhead to save state. In a conventional processor arrangement, the machine state is saved by pushing the values of all registers onto a stack, before a jump operation is executed to begin processing a different context. When the "new" context procedure is completed, the conventional system restores the state of the processor by popping from the stack all of the previously stored register values. This takes a great deal of time, particularly when the processor has a large number of registers that must be saved. The present invention is able to cycle through a plurality of contexts at high speed, without this computational overhead. Because the states are always saved automatically, context switching can occur in an instant.

The H2 processor of the preferred embodiment has program memory (PMem) that is dedicated to coordinating the schedule of process control tasks based on the timing information received by the Dispatcher from the timed context routines of the HI/O processor. This timing info is only one of the factors which influence the tasks to be scheduled. There are also dynamic inputs which influence this scheduling. Like the HI/O processor, the H2 processor has its own data memory. A portion of this data memory may be shared with the HI/O processor, as illustrated.

The presently preferred embodiment employs a particular improvement in the Harvard architecture whereby a third queue (QMem) memory system exists in which subroutines are called and executed. Unique to this system, all process control programs

are, by design, required to run once every second and to fully execute all required steps within one second. To accomplish this, a very rigorously enforced software architecture is employed in which all possible data constant and variable memory locations are allocated as part of the system design. This ensures that all memory locations are accessible by direct addressing to gain speed. In addition, each operating period is subdivided into different time segments, to ensure that all time critical applications are run every second, leaving any remaining unused cycle time available for less critical applications. In the preferred embodiment, three different time segments are employed. These are labeled Hz1, Hz25 and Hz∞. The Hz1 time segment is used for all process control applications. This is where all time critical routines are run. The Hz25 segment is optional. It may be used for non-time-critical tasks. The Hz∞ segment is used for any back burner or housekeeping operations that only need infrequent attention.

Referring to Figure 19, the HI/O processor's function is further illustrated. In one respect, the HI/O processor may be considered a multi-protocol data router. It supplies data to the H2 processor through shared memory. Thus, the H2 processor is able to concentrate on performing all time critical tasks without concern for coordination with external devices that may require asynchronous communication capability. Thus, as shown in Figure 19, the H2 processor entity 10 communicates with the HI/O processor entity 12. External data is input and output through the HI/O processor as illustrated at 14.

The HI/O processor supports a plurality of functional contexts 16. These functional contexts may represent different protocol types (such as SCSI protocol or serial protocol) or they may represent different instances of a single protocol. (Multiple contexts, like the serial context, can execute the same routines.) Each of the functional contexts has associated with it its own dedicated program counter, depicted at 18. These individual program counters each support an individual processing program indicated at 20. Although programs 20 can be for any function, the principal function of these programs within the preferred embodiment is to handle communication and system timing, as previously discussed. The functional contexts 16 also each include dedicated general purpose (GP) registers 22. These registers are used to store the individual data 23 used by each of the functional contexts.

By providing an architecture to support functional contexts that have multiple program counters and dedicated multiple general purpose registers, the system is able to save state automatically when switching from context to context. Unlike conventional

systems which push and pop register values on and off of a stack, the present system allows all important state variables, such as the program counter value and all register values, to be stored automatically in their respective dedicated program counters and GP registers. As the HI/O processor cycles from context to context, the operation of a given context may be suspended, but the state is fully preserved until that context is next given CPU attention.

The context cycling architecture of the HI/O processor allows for very rapid cycling between contexts. In fact, in the presently preferred embodiment, every successive instruction executed by the HI/O processor is within a different functional context. All of this is performed without the need to save state in the conventional fashion using a stack data structure. Recognizing that different functional contexts may have different priorities, the presently preferred architecture also includes a priority encoder 24 as part of the HI/O processor. The priority encoder tells the HI/O processor which of the active contexts to operate next.

Referring now specifically to Figure 4A, a detailed block diagram of a timer circuit 1200 for the timed context is shown. This 24-bit timer is presettable and should be synchronized to whatever synchronizing source of time is used for the HI/O microprocessor 1012 (such as the UTC or GPS pulse). The timer circuit 1200 increments at the 112 ns instruction period shown in Figure 9 and its value is compared against a timeout value, which is also set by the timed context (or any other suitable context). The Mst and Eit clock signals are also illustrated in Figure 9. The timer circuit 1200 includes a comparator 1202 for comparing an incrementing timer value with a preassigned timeout value. Once the timer exceeds this timeout value and timeout was selected as an interrupt source by the timed context, it will begin bidding and will immediately win priority encoding (because the timed context has the highest priority of all 24 contexts). In this particular embodiment the maximum period attainable with this 24-bit timer is 1.8 seconds. However, other suitable timer periods could be provided in the appropriate application.

Once active, the timed context goes through a list of items to do for that moment. While this list may initially be built by the compiler, it may also be updated dynamically, such as by the H2 microprocessor 1014. Items on the list could include the configuring of the sibling ports, scheduling of the other processing contexts, descheduling of any processing contexts (if they have "timed out"), the generation of an interrupt to the H2 microprocessor 1014, and the running of specific subroutines. As mentioned above, processing contexts are scheduled by setting the "scheduler" latch bit of the corresponding

context. This latch is mapped into the timed context's CP register (that is the context purpose register). When the timeout period defined for that context is reached, the timed context will diagnose whether a particular context finished its task; and, if it did not finish, then the timed context will "deschedule" it by resetting the same scheduler latch. Descheduling a processing context or resetting the microprocessor will automatically change the interrupt source selected by that context to the "always" selection. It should also be noted that the timed context can always deschedule another context, whether or not the other context has finished its task.

Referring to Figure 4B, a detailed block diagram of a scheduler latch mechanism is shown. In this regard, the CP register 1204 includes a separate scheduler latch 1206 for each of the processing contexts of the HI/O microprocessor 1012. In this regard, the CP register 1204 includes a multiplexer 1208 which is connected to a different bit of the main math output bus 1135 (shown in Figure 3). For example, in terms of the present embodiment, the list set forth below identifies one of the many possible connection schemes that could be employed. The preferred embodiment has 23 processing contexts corresponding to the context scheduler bits enumerated below as bits 0 through 22, and shown in Fig 4B at 1204:

bit 0: (Sasmmu-)Receiver1 context scheduler bit

bit 1: (Sasmmu-)Receiver2 context scheduler bit

bit 2: (Sasmmu-)Receiver3 context scheduler bit

bit 3: (Sasmmu-)Receiver4 context scheduler bit

bit 4: (Sasmmu-)Transmitter1 context scheduler bit

bit 5: (Sasmmu-)Transmitter2 context scheduler bit

bit 6: Serial 1 context scheduler bit

bit 7: Serial 2 context scheduler bit

bit 8: Serial 3 context scheduler bit

bit 9: Serial 4 context scheduler bit

bit 10: Serial 5 context scheduler bit

bit 11: Serial 6 context scheduler bit

bit 12: Serial 7 context scheduler bit

bit 13: Serial 8 context scheduler bit

bit 14: Serial 9 context scheduler bit

bit 15: Serial 10 context scheduler bit

bit 16: Shared Context context scheduler bit

bit 17: SCSI1 context scheduler bit

bit 18: SCSI2 context scheduler bit

bit 19: SCSI3 context scheduler bit

bit 20: SCSI4 context scheduler bit

bit 21: Arbitration context scheduler bit

bit 22: Hz∞ context scheduler bit

Other bits within the CP register 1204 have been assigned the following functions:

bit 23: H2-interrupt

bit 24: Interrupt-vector ("0" is highspeed)

bit 25: "Unprotect" PC-registerRam

When this bit is set, the timed context sees the PCs of other context mapped into the LRAM space of Local RAM 1140 (this feature could be utilized during reset, scheduling or error recovery routines). This bit is automatically cleared at RESET

bit 26: HI/OWatchdog

bit 27: Immobilize

bit 28: Timeout

bit 29: activity-change

bit 30: Received UTC-puls

bit 31: H2-reset

bit 32: Enable Sibling1 Reset

bit 33: Enable Sibling2 Reset

It should be noted that the timed context has no scheduler-latch since it runs the scheduling program. It is always scheduled, though possibly not active at any one moment. After a RESET, the timed context will be the only context running.

As previously noted, the H2 processor is preferably a Harvard architecture processor having separate program and data memories. In its presently preferred form, the H2 processor also has an additional memory, called the queue memory, for storing the master program that calls other subroutines from program memory. This has been found to be a particularly desirable configuration for process control applications, as it segregates the prewritten, well-tested, standard subroutines from the more application-specific programs written by the control engineer. Although the presently preferred embodiment of the HI/O processor is not a true Harvard architecture (it does not require physically separate program and data memories) the benefits afforded by a queue memory are nevertheless present. Accordingly, the present invention implements a software pseudo-queue.

Figures 5A and 5B give some examples of how this pseudo-queue memory works. The pseudo-queue memory is implemented as part of the HI/O data memory. Pseudo-queue pointers for each of the contexts are stored in local RAM. Figure 5A depicts the memory space within local RAM circuit 1140. Figure 5B depicts a portion of the memory space within the private program and data memory circuits 1049. Every context has its own pseudo-queue. In Figure 5B, a portion of the pseudo-queue of the timed context, the SASMMU context and the shared memory context have been illustrated. There are similar pseudo-queues for each of the remaining contexts. The entries in the pseudo-queue comprise groups of 40-bit values stored in data memory. These values function as subroutine calls and as parameters necessary for those calls. The pseudo-queue for the timed context functions more like an event queue, triggering various events within the

remaining contexts. The remaining pseudo-queues contain the parameters necessary for a particular context to complete its required function. These parameters may include (a) where to read data, (b) where to write data, (c) how to set up context specific resources, and (d) where in PMem to continue execution. In one respect, the remaining pseudo-queues behave more like job queues directing which subroutines to call next.

Referring to Figure 5B, at the portion designated as "timed context," each scheduled event in the timed context pseudo-queue requires several 40-bit values. Note that the contents at addresses 01057, 0105B, and 0105F each correspond to a timeout value used to start one of the other contexts. In this example, the value stored at 01057 is a timeout value at which time the timed context will start the SASMMU receive context by values at 0105C-0105E. The value at 0105B will start the shared transfer context. Following each timeout event, (a) three additional 40-bit values are used to toggle CP register bits, and (b) PC values are used to deschedule and schedule events.

Referring in Figure 5B to that portion of the pseudo-queue that corresponds to the SASMMU context, note that each entry is a parameter used in the subroutine that the timed context told the SASMMU context to jump to. Each context's pseudo-queue has a queue pointer stored in local RAM (shown in Figure 5A). The timed context has the ability to update (a) its queue pointer and (b) those pointers of the other contexts. Thus, when a routine within a certain context starts, its pointer is used to locate the correct parameters to use in running that routine. By writing to the other context's program counters and queue pointers, the timed context is, therefore, able (a) to revector where the other contexts run and (b) to acquire their run parameters. In this way the timed context can change the subroutines and the parameters that are used by the other contexts in executing these subroutines.

The timed context is used to enforce a desired task schedule for the various tasks that are to be executed by the HI/O microprocessor 1012 during the normal timeline. This software function is referred-to as "Time Enforcer". More specifically, the goal for the timed context is to look for timeline timeout events and then respond to these timeouts quickly. The number of consecutive instructions executed by the timed context should be kept to a minimum because the timed context has the highest priority in the HI/O microprocessor 1012 and running software in this context (with "Always" selected as the interrupt condition) will consume 50 percent of the available instruction cycles.

In the current pseudo-queue embodiment, there are two conditions which necessitate the execution of software in the timed context. The first condition for which the timed context must become involved is for each event which requires the initialization of a PC value for a context. Coincident with the initialization of a PC value for a context, the timed context may also deschedule that or other contexts. In conjunction with the scheduling and descheduling of contexts, the timed context can manipulate the bits in its CP register 1204 for such tasks as immobilizing the H2 microprocessor 1014 or resetting a sibling microprocessor. The second condition for which the timed context must become involved is when one or more bits in the CP register 1204 must be manipulated in order to have the events properly execute. For example, the timed context must clear the Immobilize bit in order to allow H2 microprocessor 1014 to begin its program execution. In this second case, there are no contexts which must have their PC values initialized — the requirement is that there is a manipulation of the timed context CP register, which involves setting and clearing of bits in that register.

The Time Enforcer function should not place restrictions on the number of contexts which can be scheduled or descheduled at a given time. In this way, the total number of instructions executed in the timed context can be minimized, and the desired task scheduler software will be given the authority on how to most efficiently arrange the schedule. The number of contexts that may be scheduled or descheduled is the maximum number of HI/O contexts, not including the timed context, which is 23 in this particular embodiment. In this regard, it should be noted that the timed context does not have a scheduler bit.

A context may both be descheduled and then rescheduled as part of the same pseudo-queue event. An example of this situation occurs when the SASMMU-RCV reception routine is to be scheduled for one or more of the SASMMU circuits 1110. The timed context should deschedule the appropriate SASMMU-RCV context (ensuring that it was not still executing or waiting to execute), then set the SASMMU-RCV context's PC value to the initial location of the reception routine, and finally the timed context would schedule this same SASMMU-RCV context for the receiving of data.

In order to reduce the number of instructions run in the timed context, the resources that are initialized by this context should be kept to a minimum. Therefore the timed context only manipulates scheduler bits and other CP bits as well as PCs. The timed context is the only context that has access to the Program Counters (PCs) of the other

contexts and the scheduler bits for the other contexts. Therefore, the timed context has the responsibility (a) to initialize another context's PC and (b) to set (or clear) the scheduler bits in order to have other contexts begin (or terminate) program execution. The timed context must also set or clear the other bits in its CP register 1204 (for example, H2 Immobilize and UTC Pulse Received) as appropriate. A further discussion of the manipulation of the bits which are a part of the CP register 1204 will be provided below. All required parameters that are accessible by the context which will be carrying out the actual task are to be initialized by that context. If the SASMMU-RCV context is to be scheduled to receive a message, then, except for pointers to parameters, the SASMMU-RCV context will be the one who initializes all necessary resources it has access-to in order for the task to successfully complete.

The following information is required for the Time Enforcer function to successfully complete its task:

A. The Timeout Value for the Next Event

B. The CP Bits Which are to be Cleared for this Event

C. If Context are to be Scheduled for this Event, The Context IDs and the Initial PC Values for the Contexts Which are to be Scheduled

D. The CP Register Bits Which are to be Set

If the writing of at least one PC value of a context is required, then this information is stored in a minimum of 4 words in the timed context pseudo-queue. If only the timed context's CP register manipulation is required, then only 3 words are required. The specific parameters which are a part of the timed context pseudo-queue are discussed in detail below:

A. Timed Out Value Entry

1. Timeout Value (24 Bits) — the time (in HI/O 112 ns clock cycles) at which the next event is to occur. An event is defined as the scheduling of one or more contexts to carry out a given task or the bit manipulation of the timed context CP register 1204. This information requires one berth each instance the Time Enforcer routine is executed.

B. CP Register AND Mask Entry

1. CP Manipulation Only (1 Bit) — this bit is included in this word in order to support the case when only a timed context CP register manipulation is required. If this bit is true, then the CP register 1204 is only to be updated (no contexts require their PC values to be written). If this bit is false, then there are contexts which require their PC values to be written.

2. CP Register AND Mask (CP Bits to Clear) (34 Bits) — the mask for the timed context CP register 1204 is provided for the contexts that are to be descheduled and other bits which may concurrently need to be cleared (for example, the Activity Change flag). For the bits in the CP register 1204 which are to be cleared, the corresponding bits in this word should be false. Any bits that are true indicate that those bits are to be kept in the same state they are currently in. It is preferred to clear the scheduler bit for the context which is to have its PC register initialized to ensure that the context is not running when timed writes its PC value or to be descheduled until at least the next event. Therefore any context which will have its PC register written to by the Time Enforcer routine should have a zero in the bit position for this context's scheduler bit. This word is required each time the Time Enforcer routine is executed.

C. PC Value Entry

1. Final PC Value word (1 Bit) — this indicates there are more PC Value words which are required in the specification of the contexts that are to be scheduled at this time. If this bit is true, then this word is the final word in the specification of PC Values. If this bit is false, then there are more PC Value words in the pseudo-queue for this time. Each context that

requires initialization to a separate PC Value will require a separate word in the timed context pseudo-queue.

2. PC Value for Context(s) (20 Bits) — the initial PC value for program execution of the consecutive context(s) that are to be scheduled at this PC value.

3. Number of Consecutive Contexts to Schedule (5 Bits) — the number of consecutive contexts that are to be scheduled at this PC value. Consecutive is defined as being in sequence in the PC Register RAM. If the contexts that are to be scheduled at the same PC value are not consecutive, then another word(s) is required in order to specify them appropriately. This field is included as a means to more efficiently schedule SASMMU-RCV contexts which run common routines such as receipt of data from a sibling, for example.

4. First PC Register RAM Address (5 Bits) — the address in PC Register RAM that corresponds to the first context that is to be scheduled at this PC address. The first context is defined as the context with the lowest PC Register RAM address to be scheduled at the PC value identified in C.2 above. Note that the PC Register RAM address is done according to active context code number and not the context priority.

D. CP Register OR Mask Entry

1. Jump to Internal ROM (1 Bit) — this bit is a flag which determines whether or not a jump to internal ROM is to be executed. This bit will be cleared in normal operation and will be set when the processor is to execute a reload. In this way, a software reset can be performed. The timed context will jump to FFFE0H and begin executing the internal ROM code, which will in turn load in the external

ROM software, performing any necessary QC tests and the loading procedure.

2. CP Register OR Mask (CP Bits to Set) (34 Bits) — the mask for the timed context CP register 1024 that is provided for the contexts that are to be scheduled and other bits which may concurrently need to be set (for example, when loading a H2 Program, the Immobilize Bit). For contexts that are to be scheduled and for other bits in the CP register which are to be set, the bits corresponding to the position in the timed context CP register should be true. Any bits that are false indicate that those bits are to be kept in the same state they are currently in. This information requires one word each time the Time Enforcer routine is executed.

At this point, it might be helpful to illustrate a further example of the pseudo-queue method of operation according to one form of the present invention. In this regard, each of the 24 contexts of HI/O microprocessor 1012 preferably has its own "Pseudo-Queue" (PQ) pointer contained in Local RAM (LRAM) memory 1140 at the address corresponding to that context's active context code. This concept is illustrated in Figure 5A, which diagrammatically shows a series of addresses in LRAM 1140 and a label corresponding to the contents at each of these addresses.

Figure 5B diagrammatically shows 3 sets of memory addresses in a portion of the private program and data memory circuits 1049 for the HI/O microprocessor 1012. As will be seen in the example below, a portion of the time context (Time Enforcer) pseudo-queue data is stored at address locations 01057 through 01062. Similarly, a portion of the SASMMU context receive pseudo-queue data is stored at address locations 02F57 through 02F5C and a one word portion of the shared context transfer pseudo-queue data is stored at address location 05BB6. These small pseudo-queue sections are illustrated to provide an execution example of how a data block might be received and stored into the private program and data memory circuits 1049 by SASMMU Receiver 1 at t=285.594274 ms, and then transferred to shared data memory circuit 1042 by the shared context at t=285.761045 ms, all under the control of the timed context.

Assume that the timed context has just timed out at t=285.594274 ms and its PQ (pseudo-queue) pointer (at Ø1 hex in LRAM 1140) has the value 0105Bh. The comparator 1202 of timer circuit 1200 in Figure 4A has just generated a HIGH timeout signal, which is directed to the priority encoder circuit shown in Figure 6. Since the timed context has the highest priority of all the contexts, it will win the bidding process, become the "new" context during the next instruction period and then become the "active" context on the next fundamental instruction period of the HI/O microprocessor 1012. While not all operations listed below are necessarily performed on one instruction, the execution sequence is illustrative of the pseudo-queue method of operation according to the present invention.

- The timed context reads the word stored in the private program and data memory circuits 1049 (DMem) at PQ pointer value (0105Bh), then increments its pseudo-queue pointer to 0105Ch, and writes the data memory value read to a MMED register FD hex (timeout register mapped into one of the upper 128 locations of the local RAM address space thus updating the next timeout value). The pointer stored at address "01" in LRAM 1140 will be changed from 0105B to 0105C. The data value, written to FD hex, will be the hex equivalent of t=285.761045 ms in the 1 second timeline of a repeating process control cycle.

- Timed context reads the next PQ value from DMem (0105Ch), increments its PQ pointer, ANDs the PQ value read with its CP register 1204, turning off SASMMU1 and writing out the new CP (all in one instruction period). It should be noted here that these timed context operations will not be absolutely time sequential in terms of instruction-to-instruction execution within the HI/O microprocessor 1012, as the timed context is not permitted to monopolize all of the microprocessor execution time. (Only 50 percent maximum is permitted).

- Timed context reads DMem at PQ pointer value (0105Dh), increments its PQ pointer and uses 20 bits of the PQ value data to set the PC of SASMMU1 to a "receive" routine address. Only SASMMU1 will be scheduled at this time at this PC (bits 9:5 of the data word read=00001b, indicating that only one context should be scheduled at this PC). SASMMU1's active context code (00001b) is also contained in this data

word. Finally, the most significant bit in this data word is set to indicate that this is the last PC value word corresponding to this PQ time segment. As will be seen in a subsequent example, more than one processing context can be concurrently operating and even use the same software routine in the process.

- Timed context reads DMem at its PQ pointer (0105Eh), increments its PQ pointer, and ORs the PQ value with its CP register in order to schedule SASMMU1 (CP bit 1). The timed context then switches its interrupt source to "Timeout" and is inactive until the timeout at t=285.761045. Accordingly, it should be appreciated that the timed context effectively sets an alarm clock, issues marching orders to one or more processing contexts, and simply goes to sleep until the alarm goes off. Each of the other 23 processing contexts are going about their tasks, if so ordered, on an independent basis.

- The SASMMU1 context reads DMem at PQ pointer (02F57h), increments its PQ pointer and uses the PQ value as an AND mask for the SASMMU configuration MMed register in the LRAM address space. Thus, as shown in Figure 5B, a pseudo-queue may be provided for both the timed and device processing contexts.

- The SASMMU1 context reads DMem at PQ pointer (02F58h), increments its PQ pointer, and uses the PQ value as an OR mask for the SASMMU configuration MMed register in the LRAM address space. This mask combined with the AND mask will generate a new input/output mux configuration for the physical SASMMU1 circuit. (Generally, a SASMMU context should only modify the bits of this configuration register pertaining to its own multiplexer settings as bits of other SASMMU contexts are maintained in this register as well).

- The SASMMU1 context reads DMem at PQ pointer (02F59h), increments its PQ pointer and uses 20 bits of the PQ value as the length of the message it is to receive, with another bit indicating whether or not to decrypt the incoming data.

- SASMMU1 reads DMem at PQ pointer (02F5Ah), increments its PQ pointer and uses 20 bits of the PQ value as the destination address of the received data block and the other 20 bits as the PC value of the routine where the SASMMU1 context will jump to do the actual receiving. The SASMMU1 context then jumps to the receive routine and receives the data.

- After receiving the block of data of the length specified, the SASMMU1 context reads DMem at PQ pointer (02F5Bh), increments its PQ pointer and uses the PQ value as the "Expected Message Header."

- SASMMU1 reads DMem at PQ pointer (02F5Ch), increments its PQ pointer and uses the PQ value as the destination address for the message reception diagnostic bits. Some of the message reception diagnostic bits are generated by comparison of the expected message header to the actual message header and these bits are then written out to data memory at the address value stored at 02F5Ch.

- The SASMMU reception task provides two outputs as a result of its execution. These two outputs are: “reception diagnostics” and “received data”. The message diagnostics are provided in two separate places. One is intended to be short-term and the other is intended to be longer term. The short-term diagnostics are stored just preceding the received data and are utilized by the arbitration processing context in order to determine the validity of the data. The longer term diagnostics are stored at the HI/O DMem location specified in the pseudo-queue for this task. Both locations contain identical information. This information and their relative bit positions are as follows: Checksum Error Detected (MSB); Bad Message Header; Context Not Descheduled; No Data Received (LSB). The short-term diagnostics store this information in the lower 4 bit positions in the word. The longer term diagnostics store this information based on receiver number (it is not configurable via pseudo-queue). The bit positions which are occupied are as follows: Me (Receiver 3 or 4) — Bits 3-0; Sib1 (Receiver 1) — Bits 7-4; Sib2 (Receiver 2) — Bits 11-8.

- After the timeout at t=285.761045 ms, the timed context reads DMem at PQ pointer (0105Fh), writes the PQ value read to the LRAM MMed register (timeout register) at FDh (updating next timeout value and then incrementing its PQ pointer to 01060h).

- The timed context reads DMem at PQ pointer (01060h), increments its PQ pointer, ANDs the PQ value read with its CP register, turning-off the shared context (and also turning-off SASMMU1 if not already off). The Most Significant Bit (MSB) is equal to zero in the mask data word (NOT CP register manipulation only).

- The timed context reads DMem at PQ pointer (01061h), increments its PQ pointer, and uses 20 bits of the PQ value to set the PC of the shared context to a "write transfer" routine address. Only one context (the shared context) will be scheduled at this time at this PC (bits 9:5 of the data word read=00001b). The shared context's active context code (01011b) is also contained in this data word in bits 4-0. Finally, the MSB is set in this data word to indicate that this is the last PC value word corresponding to this PQ time segment.

- The timed context reads DMem at PQ pointer (01062h), increments its PQ pointer, and ORs the PQ value with its CP register in order to schedule shared (CP register bit 16). The timed context then switches its interrupt source to "Timeout" and is inactive until the next timeout.

- The shared context reads its first PQ pointer from LRAM, uses this value to read DMem at PQ pointer (05BB6h), increments its PQ pointer, and uses 20 bits of the PQ value as the source address of HI/O DMem from where the data block is to be read. Some of the other bits are used as the length of the data block. (It should be noted that the shared context maintains two separate PQ pointers in LRAM, one to point to items in HI/O program and data memory, the other to point to items in shared memory).

- The shared context reads its second PQ pointer from LRAM and uses this value to read a transfer parameter stored in shared memory at the PQ pointer. It then increments its PQ pointer and uses 24 bits of the PQ value (transfer parameter) as the shared memory destination address of the

data block to be transferred, with 1 more bit to indicate whether this block is to go to the shared data memory circuit 1042 or to H2's program memory circuit 1032 (both of which are preferably mapped into the shared memory space). The shared context then transfers the data block and deschedules itself (clears its CP bit 15).

More than one context can be scheduled at any PQ time segment (turning on all SASMMU Receive contexts, for example) and, if desired, each can run the same code. Contexts running the same code can determine their "context specific" parameters by first reading their active context ID from an LRAM MMed (memory mapped) register and then using this information to generate, for example, data memory address locations, LRAM address locations, and offsets. While running several contexts in parallel, care should be taken to not over-schedule the processor, as asynchronous data (SASMMU, Serial) may be lost if contexts related to asynchronous data do not win priority to run when they need to.

There are a number of variables which will affect the execution timing required by a time enforcement routine. Note that a task scheduler must be aware of the duration required for the time enforcement routine, as this time will limit the times at which a context may be scheduled. The task scheduler cannot choose to schedule contexts at every 112 ns boundary.

The following factors will influence the execution timing of the time enforcement routine:

A. Whether CP register manipulation is the only function which is needed, or if PC values are also required to be written to specific contexts.

B. The number of consecutive contexts which are to be scheduled at the same PC value. (The number of consecutive contexts that will be scheduled at a given PC value will be referred to as n. This value is equal to the number which is stored in the PC value word).

C. The number of groups of contexts. (A single group is a number of consecutive contexts which are to be scheduled at a given PC value and will be referenced as m. One group corresponds to a single word in the pseudo-queue).

Additionally, in this particular embodiment, the number of execution cycles required if only CP manipulation is required is 11. Accordingly, in this embodiment, the number of execution cycles required if PC values are to be written to contexts is:

$$12+7m+\sum_{m=1}^{m}(n(m\text{-}1))$$

The conclusions that can be drawn about this equation are:

(1) Scheduling a single context requires 19 instructions.

(2) Each context which requires a different PC value (or is not consecutive with the first) requires 7 additional instructions.

(3) Each consecutive context which is scheduled at the same PC value requires 1 additional instruction.

The following provides two examples which further illustrate the concept of concurrently running more than one processing context using the same software routines for at least some of these contexts. In the first of these two examples, the SASMMU-RCV 1 and SASMMU-RCV 2 circuits will be employed to receive corresponding data from sibling microprocessors, while the SASMMU-RCV 3 circuit will be employed to receive data from a network level microprocessor. Then, in the second example, the process of arbitrating between the data received by the SASMMU-RCV 1 and SASMMU-RCV 2 circuits will be initiated.

Commencing then with the first of these examples, it will be assumed that the timeout for the next event is 145Ah. No contexts are to be descheduled at this time and no other bits in the CP register 1204 need to be cleared. The contexts that are currently to be scheduled are SASMMU-RCV 1 (to receive data from Sibling 1), SASMMU-RCV 2 (to receive corresponding data from Sibling 2), and SASMMU-RCV 3 (to receive data from the network), while no other bits in the timed context CP register need to be set. All 3 SASMMU-RCV contexts will be running the common SASMMU-RCV software, which has an initial PC address of 0586h. It should be noted that each of the addresses provided herein are fictitious and are given simply to facilitate understanding of the methods of operation for the present invention. The pseudo-queue entries for this event are as follows:

A. Timeout Value Entry

1. The timeout for the next event is 145Ah, therefore the initial word in the pseudo-queue for this event contains this value.

B. CP Register AND MASK Entry

1. There are (3) contexts which are going to be scheduled, therefore the CP register Manipulation Only bit is false.

2. Since the SASMMU-RCV 1, SASMMU-RCV 2 and SASMMU-RCV 3 contexts will have their PCS written-to, they will have their scheduler bits cleared prior to the writing to their PCS. This clearing prior to the writing of the CPs is the preferred practice. However, it should be understood that this and other similarly situated steps may be modified in the appropriate application.

C. PC Value Entry

1. The PC for the contexts that are to be scheduled is the initial address at which the SASMMU-RCV common routine is stored in the program and data memory 1049 (PMem).

2. The PC Register RAM address for the first context (which is SASMMU-RCV 1's active context code) is 00h.

3. The number of contexts to schedule is 3, therefore, the number specified in the (Number of Contexts to Schedule) field must be 03h.

4. This is the final PC address word, therefore the Final PC Address Word bit must be true.

D. CP Register OR Mask Entry

1. The contexts that are to be scheduled are SASMMU-RCV 1 (whose scheduler bit is in bit position 0), SASMMU-RCV 2 (whose scheduler bit is in bit position 1) and SASMMU-

RCV 3 (whose scheduler bit is in bit position 2). Therefore, the lower 3 bits in the word will be true and all the others will be false.

In the second of these two examples, it will be assumed that the timeout value for the next event is 2495h. As indicated above, data has been exchanged with the siblings and arbitration is now to occur. Also, this example will include the task of moving a previously arbitrated message to the shared DMem 1042. Accordingly, the SASMMU-RCV 1, SASMMU-RCV 2 and SASMMU-RCV 3 contexts are to be descheduled and the Activity Change bit is also to be cleared at this time. The initial PC for the arbitration common routine is 0642h and the initial PC for transferal of data to shared memory is 0824h. The pseudo-queue entries for this event are as follows:

A. Timeout Value Entry

1. The timeout for the next event is 2495H, therefore the initial word in the pseudo-queue for this event contains this value.

B. CP Register AND Mask Entry

1. There are contexts which require that their PC Register be written, therefore the CP Manipulation Only bit is false.

2. SASMMU-RCV 1, SASMMU-RCV 2 and SASMMU-RCV 3 contexts are to be descheduled, as well as the Arbitration and Shared&Host context. The Activity Change bit is to be cleared, therefore bit 29 will also be false.

C1. PC Value Entry (Word 1)

1. The PC for the contexts that are to be scheduled is the initial address at which the Arbitration common routine is stored in PMem: 0642h.

2. The PC Register RAM address for the first context (which is arbitration context's active context code) is 16h.

3. The number of contexts to schedule at this PC Address is 1, therefore, the number specified in the (Number of Contexts to Schedule) field must be 01h.

4. This is not the final PC Address Word, therefore the Final PC Address Word bit must be false.

C2. PC Value Entry (Word 2)

1. The PC for the contexts that are to be scheduled is the initial address at which the Shared&Host common routine is stored in PMem: 0825h.

2. The PC Register RAM address for the first context (which is the Shared&Host Context) is 0Bh.

3. The number of contexts to schedule at this PC Address is one, therefore, the number specified in the field must 01h.

4. This is the final PC Address word, therefore, the Final PC Address word bit must be true. Accordingly, it should be clear by this point that a series of PC Value Entry Words may be sequentially stored in the program and data memory (DMem) 1049, in contrast to the example of only one such word in Figure 5B.

D. CP Register OR Mask Entry

1. The arbitration (bit 21) and the shared memory (bit 16) contexts are to be scheduled, therefore, the CP register OR Mask will contain those two bits true, while all other bits will be false.

Some special consideration is required in the handling of the nonscheduler bits in the timed context CP register 1204. These considerations are handled on a bit-by-bit basis as described below.

H2 Interrupt (Bit 23) — The H2 Interrupt line is preferably handled in the time enforcer routine. This bit will be set, when required by the CP Register OR Mask, which has

been described above. The clearing of this interrupt line is done one instruction after the CP register bits are set in order to assure that the generation of the interrupt is not interpreted by the H2 microprocessor 1014 as "multiple interrupts". The clearing of this line is preferably done every time the timed context executes this routine.

Interrupt Vector (Bit 24) — This line is handled by the time enforcer routine. It will be initialized to its proper state either during the writing of the CP Register AND Mask (for clearing the interrupt vector so that it is in the zero state) or during the writing of the CP Register OR Mask (for setting the interrupt vector to a 1 state). This Interrupt Vector need only be altered when a H2 interrupt is to be generated. It should be left in the state of the last interrupt until it is rewritten.

Unprotect PC Register RAM (Bit 25) — This bit is set when the timed context is required to write to the PCs of the other contexts. This bit is an integral part of the time enforcer routine, so that the timed context can initialize contexts which are required to perform particular tasks. The setting and clearing of this bit is done independently from the AND mask and the OR mask which is provided in the timed context pseudo-queue.

HI/O Watchdog (Bit 26) — The HI/O Watchdog bit is ANDed with the H2 Watchdog bit and this result is an output signal from the HI/O Microprocessor 1012. The bit can be cleared by the CP Register AND Mask or set by the CP Register OR Mask. The determination of the conditions for the setting and clearing of the HI/O Watchdog bit is not the prerogative of the time enforcer.

Immobilize (Bit 27) — This bit is set when HI/O microprocessor 1012 is to load a program into PMem (program memory circuit 1032) of the H2 microprocessor 1014. The setting of the Immobilize bit will be handled concurrently with the setting of the other bits in the CP register by the CP register OR Mask. It will be cleared by the CP Register AND Mask.

Timeout (Bit 28) — This bit reflects the state of a timeout and is true if a timeout has occurred and false if a timeout has not occurred. This bit is read-only and therefore does not allow any software manipulation.

Activity Change (Bit 29) — As discussed above, the time enforcer routine bases its operation on timeouts and not on the asynchronous nature of an activity change

(which indicates that a context has descheduled itself). Consequently, the manipulation of this bit is not necessary.

Received UTC Pulse (Bit 30) — The time enforcer software handles this bit the same as any of the other bits in the CP register (that is, it can be set or cleared with the CP register masks provided in the pseudo-queue). In the preferred embodiment the bit is set by a leading edge detected on an input pin of the HI/O processor.

H2 Reset (Bit 31) — This bit will be set, when required, by the CP Register OR Mask, which is described above. The setting of this bit will cause the H2 microprocessor 1014 to be reset (that is, the H2 microprocessor will have its PC set to zero). The clearing of this reset line is done one instruction after the CP bits are set since this will cause the H2 microprocessor to reset. The clearing of this line is preferably done every time the timed context executes this routine. The H2 microprocessor should be Reset for a minimum of two instruction cycles.

Enable Sibling 1 Reset (Bit 32) and Enable Sibling 2 Reset (Bit 33) — These bits are set via the PC Register OR Mask when it is the intent to enable the reset of one or both of the sibling microprocessors.

The present invention has been described in an illustrative manner. In this regard, it is evident that those skilled in the art once given the benefit of the foregoing disclosure, may now make modifications to the specific embodiments described herein without department from the spirit of the present invention. Such modifications are to be considered within the scope of the present invention which is limited solely by the scope and spirit of the appended claims.

What Is Claimed Is:

1. A dedicated context-cycling microprocessor (1012) having at least one computational unit (1134) for performing mathematical or logical functions or a combination thereof, a plurality of circuits (1100-1106, 1108, 1110) for receiving and transmitting information, and a bus structure for transferring instructions and data with external memory devices, characterized by:

a plurality of processing contexts (16) each having an individual set of dedicated registers (22) in which a distinct processing context is provided for each of said plurality of said circuits, said processing contexts including a timed context for concurrently scheduling multiple processing contexts into a schedule and for enforcing time constraints associated with said schedule; and

a multiplexer circuit (1146) for moving data between said circuits, said dedicated registers, and said computational unit, said multiplexer circuit having a move bus (109) coupled to said bus structure.

2. A dedicated context-cycling microprocessor (1012) comprising:

a plurality of input/output circuits (1100-1106, 1108, 1110) for receiving and transmitting information;

a plurality of processing contexts (16) each having an individual set of dedicated registers (22) in which a distinct processing context is provided for each of said plurality of said input/output circuits, said processing contexts including a timed context for concurrently scheduling multiple processing contexts into a schedule and for enforcing time constraints associated with said schedule;

at least one computational unit (1134) for performing mathematical performing mathematical or logical functions or a combination thereof;

a bus structure for transferring instructions and data with external memory devices; and

a multiplexer circuit (1146) for moving data between said input/output circuits, said dedicated registers, and said computational unit, said multiplexer circuit having a move bus (109) coupled to said bus structure.

3. The invention according to Claim 2 further comprising a local RAM (1140) and a program memory and wherein at least said timed context has a pseudo-queue list whose pointers are stored in said local RAM, said pseudo-queue list representing an ordered set of pointers to data and program memory words to be used for scheduling said processing contexts.

4. The invention according to Claim 2 wherein said input/output circuits include at least one serial and shared memory management unit (1110) comprised of a parallel-to-serial converter and a transmitter circuit which is capable of transmitting a plurality of serial word frames without any spaces between them, said serial and shared memory management unit having its own processing context.

5. A common core computer (1000) comprising:

a first microprocessor (1014);

a first private memory system (1026) connected to said first microprocessor;

a second microprocessor (1012);

a second private memory system (1048) connected to said second microprocessor; and

a shared memory system (1016) connected to both said first and second microprocessors;

said second microprocessor including a plurality of input/output circuits (1100-1106, 1108, 1110) for receiving and transmitting information with external communication devices and timing means for controlling the operation of said first microprocessor and access to said shared memory system.

6. The invention according to Claim 5 wherein said first microprocessor has a Harvard architecture and said second microprocessor has an opcode width corresponding to a data word width of said first microprocessor.

7. The invention according to Claim 5 wherein said second microprocessor includes:

an individual set of dedicated registers (22) for each of a plurality of processing contexts (16) in which a distinct processing context is provided for each of a plurality of said input/output circuits, said processing contexts also including a timed context for concurrently scheduling multiple processing contexts and enforcing time constraints associated with said schedule;

at least one computational unit (1134) for performing mathematical or logical functions or a combination thereof, said computation unit being coupled to at least one of said input/output circuits;

a bus structure for transferring instructions and data with said shared memory system and said second private memory system; and

multiplexer circuit means for moving data between said input/output circuits, said dedicated registers, and said computational unit, said multiplexer circuit means having a move bus (109) coupled to said bus structure.

8. The invention according to Claim 7 wherein at least some of said processing contexts have their own plurality of interrupt signal inputs.

9. The invention according to Claim 1 or Claim 2 or Claim 7 wherein each of said sets of dedicated registers includes at least one general purpose register and a program counter.

10. The invention according to Claim 9 wherein said timed context has access to at least the program counter (18) of said dedicated registers (22) in all of the other of said processing contexts.

11. The invention according to Claim 1 or Claim 2 or Claim 7 wherein said processing contexts include an idle state in which the context is waiting to be scheduled, an inactive state in which a scheduled context is waiting for an event to occur, a bidding state in which a scheduled context is competing for service, and an active state in which the scheduled context has been granted the highest priority and is now executing a fetched instruction.

12. The invention according to Claim 11 wherein a scheduler latch (1206) is provided for each of said processing contexts except for said timed context and a context purpose register is provided for said timed context, with each of said scheduler latches being

mapped into said context purpose register for said timed context in order to schedule one or more of said processing contexts for service.

13. The invention according to Claim 11 wherein said processing contexts have preassigned priority levels and at least some high priority processing contexts are capable of lowering their own priority.

14. The invention according to Claim 11, wherein said microprocessor having said processing contexts further includes a priority encoding circuit which ensures that one instruction of an active processing context must be executed before the active processing context is permitted to bid again for service.

15. The invention according to Claim 7 further comprising a local RAM (1140) and a program memory, and wherein at least said timed context has a pseudo-queue list stored in said local RAM which represents an ordered set of pointers to data and program memory words to be used for scheduling said processing contexts.

16. The invention according to Claim 7 wherein said input/output circuits include at least one serial and shared memory management unit (1110) comprising a parallel-to-serial transmitter circuit which is capable of transmitting a plurality of serial word frames without deadtime, said serial and shared memory management unit having its own processing context.

17. The invention according to Claim 4 or Claim 16 wherein said input/output circuits also include a plurality of SCSI interfaces (1100-1106) with each of said SCSI interfaces having its own programming context.

18. A method of processing input/output signals of different types by a process control computer, comprising:

defining a plurality of functional contexts corresponding to said different types of input/output signals;

configuring an input/output processor to have a dedicated program counter and at least one dedicated general purpose register for each of said functional contexts; and

operating said input/output processor to cycle from one context to another such that the processing state of a first context is saved in the dedicated program counter

and dedicated general purpose register of that first context while switching to process a second context by accessing the program counter of that second context.

19. The method of Claim 18 wherein said input/output processor operates in synchronism with system clock that defines a fundamental clock interval, and

wherein said step of operating said input/output processor is performed such that one context is active during a given fundamental clock interval.

20. The method of Claim 19 further comprising subdividing said fundamental clock interval into a first component and a second component that follows said first component, and

wherein the processing state of said first context is saved in the dedicated general purpose register of the first context during the first component of a first fundamental clock interval, and

wherein the program counter of the second context is accessed in preparation to process the second context during the second component of the said first fundamental clock interval.

21. The method of Claim 18 further comprising configuring a second processor to have a memory space that is shared with said input/output processor and using said input/output processor to communicate information about said input/output signals to said second processor by storing said information as data in said shared memory space.

22. The method of Claim 18 further comprising configuring a second processor to have a memory space that is shared with said input/output processor and using said input/output processor to supply timing information to said second processor.

23. The method of Claim 18 wherein said plurality of contexts includes a timed context that oversees the order in which context switching is performed.

24. The method of Claim 18 wherein said functional contexts are assigned different priorities and wherein said step of operating said input/output processor is performed by selecting said second context based on priority bidding.

25. A multi-protocol data processing computer for communicating with multiple processes comprising:

an input/output (I/O) processor (1012) for communicating with multiple processes using different protocols;

said I/O processor having context cycling system for defining a plurality of independent operating-state contexts, each context devoted to one of said protocols;

said context cycling system having a plurality of program counter registers and a plurality of general purpose registers each dedicated to said contexts such that each context has its own program counter register and its own general purpose registers;

said context cycling system having priority encoder for selecting one of said contexts as the operative context, such that the program counter register and general purpose registers of the operative context are identified to said I/O processor as the operative program counter register and the operative general purpose registers, and the remaining program counter registers and general purpose registers are treated by said I/O processor as inoperative;

said I/O processor communicating with one of said multiple processes by accessing the operative program counter register and the operative general purpose registers, such that communication is effected in accordance with the operative context;

said I/O processor saving the state of the contexts not selected as operative by leaving the contents of the inoperative program counters and general purpose registers unchanged while communication is effected in accordance with the operative context.

26. A multi-protocol multiprocessor data processing system for communicating with and controlling multiple processes comprising:

an input/output (I/O) processor (1012) for communicating with multiple processes using different protocols, said I/O processor having first private memory (1048) accessed by said I/O processor in communicating with said multiple processes;

a second processor (1014) for executing process control instructions to control said multiple processes, said second processor having second private memory (1026) accessed by said second processor in executing process control instructions;

a shared memory separate from said first and second private memories and accessed by both said I/O processor and said second processor for establishing communication between said I/O processor and said second processor;

said I/O processor having context cycling system for defining a plurality of independent operating-state contexts, each context devoted to one of said protocols;

said context cycling system having a plurality of program counter registers and a plurality of general purpose registers each dedicated to said contexts such that each context has its own program counter register and its own general purpose registers;

said context cycling system having priority encoder for selecting one of said contexts as the operative context, such that the program counter register and general purpose registers of the operative context are identified to said I/O processor as the operative program counter register and the operative general purpose registers, and the remaining program counter registers and general purpose registers are treated by said I/O processor as inoperative;

said I/O processor communicating with one of said multiple processes by accessing the operative program counter register and the operative general purpose registers, such that communication is effected in accordance with the operative context;

said I/O processor saving the state of the contexts not selected as operative by leaving the contents of the inoperative program counters and general purpose registers unchanged while communication is effected in accordance with the operative context.

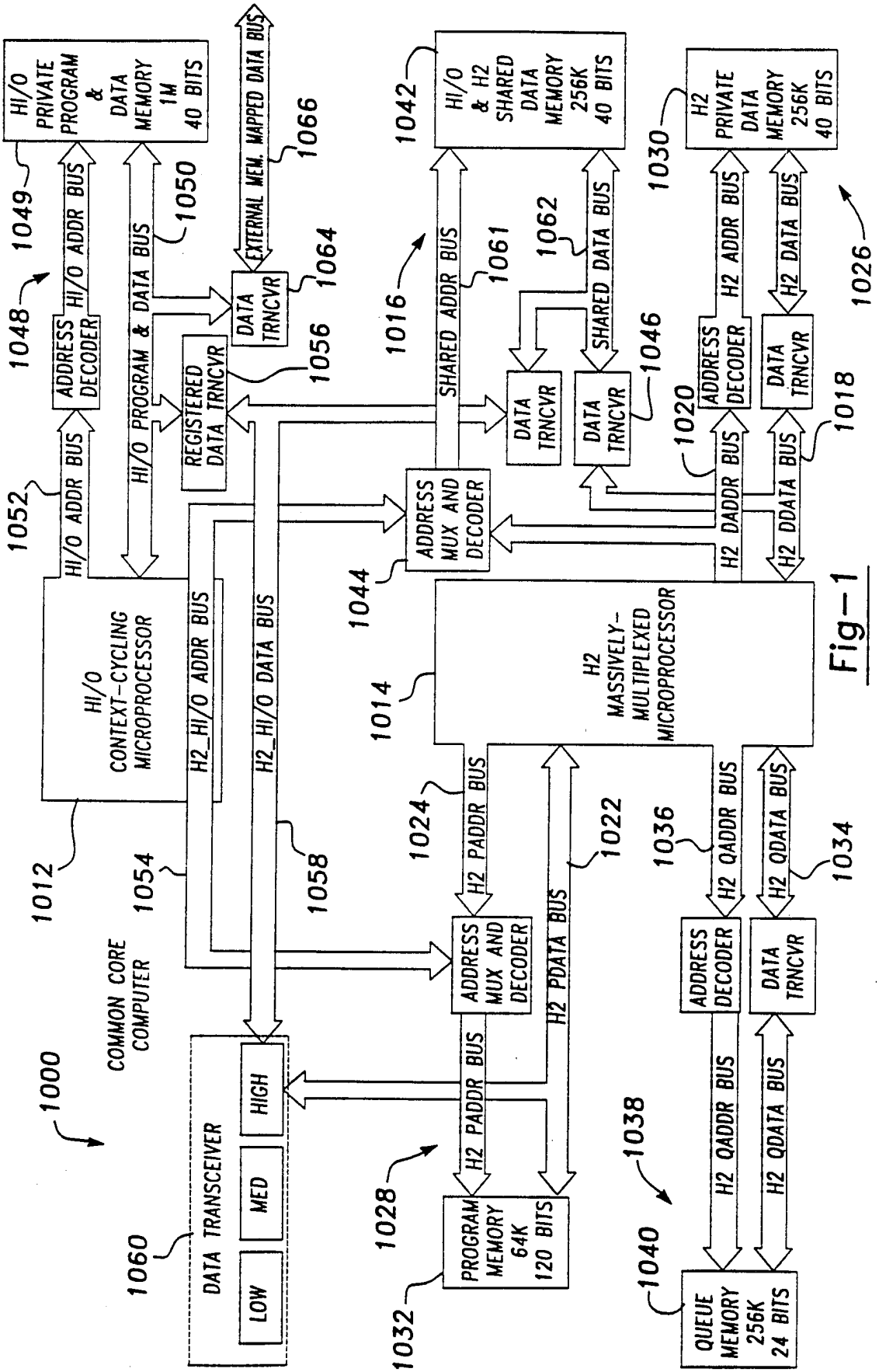

HI/O PRIVATE PROGRAM & DATA MEMORY 1M 40 BITS
1049
1048
ADDRESS DECODER
HI/O ADDR BUS
HI/O PROGRAM & DATA BUS
1050
EXTERNAL MEM. MAPPED DATA BUS
1066
DATA TRNCVR
1064
1056
REGISTERED DATA TRNCVR
1052
HI/O ADDR BUS
HI/O CONTEXT-CYCLING MICROPROCESSOR
1012
H2_HI/O ADDR BUS
H2_HI/O DATA BUS
1058
1054
COMMON CORE COMPUTER
1000
DATA TRANSCEIVER
HIGH
MED
LOW
1060
1028
PROGRAM MEMORY 64K 120 BITS
1032
H2 PADDR BUS
H2 PDATA BUS
ADDRESS MUX AND DECODER
1024
1022
1014
H2 MASSIVELY-MULTIPLEXED MICROPROCESSOR
1044
ADDRESS MUX AND DECODER
SHARED ADDR BUS
1061
1016
1042
HI/O & H2 SHARED DATA MEMORY 256K 40 BITS
1062
SHARED DATA BUS
DATA TRNCVR
DATA TRNCVR
1046
H2 DADDR BUS
H2 DDATA BUS
1020
1018
1030
H2 PRIVATE DATA MEMORY 256K 40 BITS
ADDRESS DECODER
H2 ADDR BUS
DATA TRNCVR
H2 DATA BUS
1026
1036
H2 QADDR BUS
H2 QDATA BUS
1034
1038
ADDRESS DECODER
DATA TRNCVR
H2 QADDR BUS
H2 QDATA BUS
1040
QUEUE MEMORY 256K 24 BITS

Fig-1

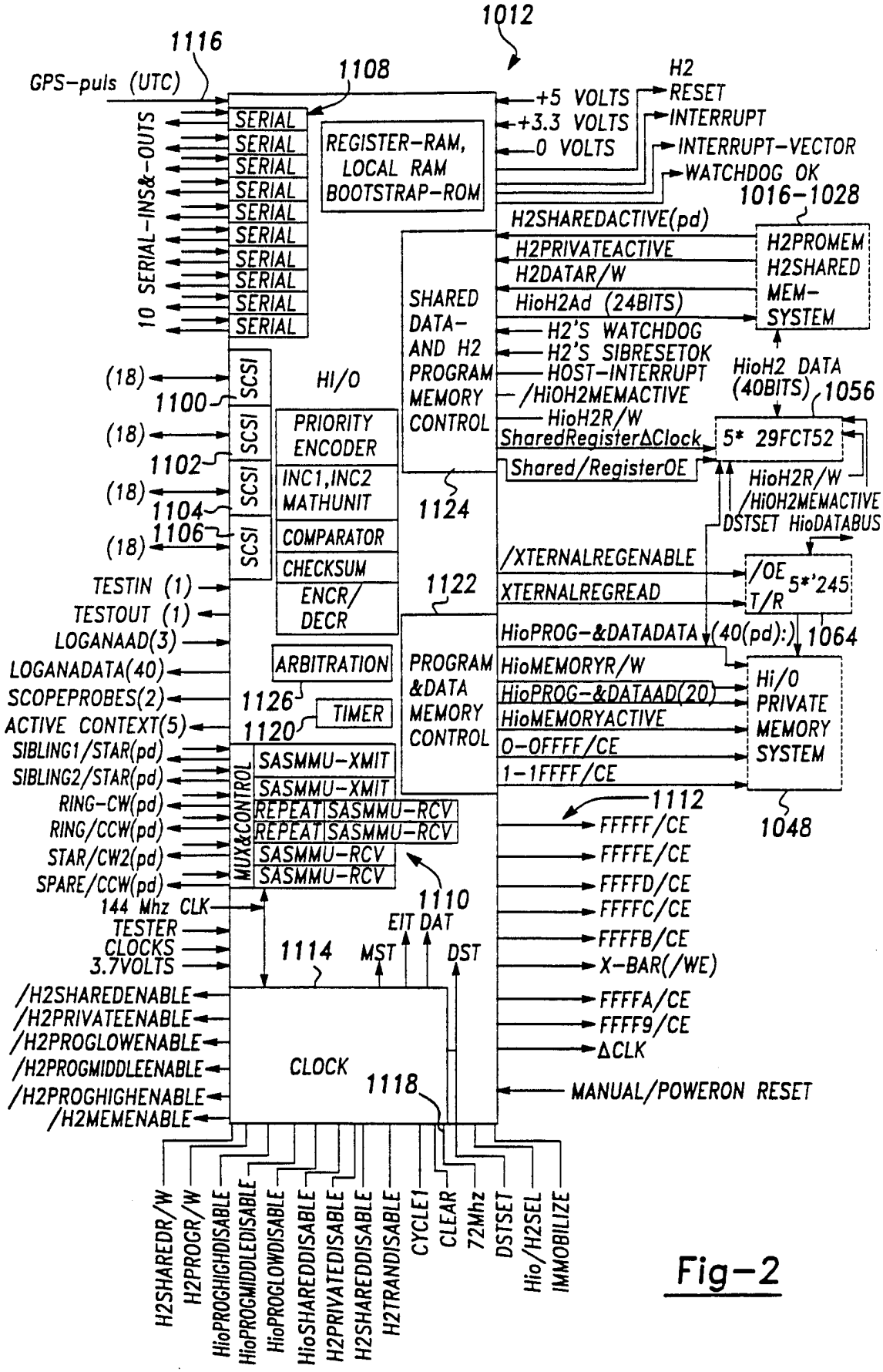

1012
1116
1108
GPS-puls (UTC)
10 SERIAL-INS&-OUTS
SERIAL
REGISTER-RAM, LOCAL RAM BOOTSTRAP-ROM
+5 VOLTS
+3.3 VOLTS
0 VOLTS
H2 RESET
INTERRUPT
INTERRUPT-VECTOR
WATCHDOG OK
1016-1028
H2SHAREDACTIVE(pd)
H2PRIVATEACTIVE
H2DATAR/W
HioH2Ad (24BITS)
H2PROMEM H2SHARED MEM-SYSTEM
SHARED DATA- AND H2 PROGRAM MEMORY CONTROL
H2'S WATCHDOG
H2'S SIBRESETOK
HOST-INTERRUPT
/HiOH2MEMACTIVE
HioH2R/W
SharedRegisterΔClock
Shared/RegisterOE
HioH2 DATA (40BITS)
1056
5* 29FCT52
HioH2R/W /HiOH2MEMACTIVE
DSTSET HioDATABUS
1124
(18)
1100
1102
1104
1106
SCSI
HI/O
PRIORITY ENCODER
INC1,INC2 MATHUNIT
COMPARATOR
CHECKSUM
ENCR/ DECR
/XTERNALREGENABLE
XTERNALREGREAD
/OE T/R 5*'245
1122
HioPROG-&DATADATA (40(pd):)
1064
PROGRAM &DATA MEMORY CONTROL
HioMEMORYR/W
HioPROG-&DATAAD(20)
HioMEMORYACTIVE
0-0FFFF/CE
1-1FFFF/CE
Hi/O PRIVATE MEMORY SYSTEM
1048
1112
FFFFF/CE
FFFFE/CE
FFFFD/CE
FFFFC/CE
FFFFB/CE
X-BAR(/WE)
FFFFA/CE
FFFF9/CE
ΔCLK
MANUAL/POWERON RESET
TESTIN (1)
TESTOUT (1)
LOGANAAD(3)
LOGANADATA(40)
SCOPEPROBES(2)
ACTIVE CONTEXT(5)
SIBLING1/STAR(pd)
SIBLING2/STAR(pd)
RING-CW(pd)
RING/CCW(pd)
STAR/CW2(pd)
SPARE/CCW(pd)
144 Mhz CLK
TESTER CLOCKS
3.7VOLTS
ARBITRATION
1126
TIMER
1120
MUX&CONTROL
SASMMU-XMIT
SASMMU-XMIT
REPEAT SASMMU-RCV
REPEAT SASMMU-RCV
SASMMU-RCV
SASMMU-RCV
1110
EIT DAT
1114
MST
DST
/H2SHAREDENABLE
/H2PRIVATEENABLE
/H2PROGLOWENABLE
/H2PROGMIDDLEENABLE
/H2PROGHIGHENABLE
/H2MEMENABLE
CLOCK
1118
H2SHAREDR/W
H2PROGR/W
HioPROGHIGHDISABLE
HioPROGMIDDLEDISABLE
HioPROGLOWDISABLE
HioSHAREDDISABLE
H2PRIVATEDISABLE
H2SHAREDDISABLE
H2TRANDISABLE
CYCLE1
CLEAR
72Mhz
DSTSET
Hio/H2SEL
IMMOBILIZE

Fig-2

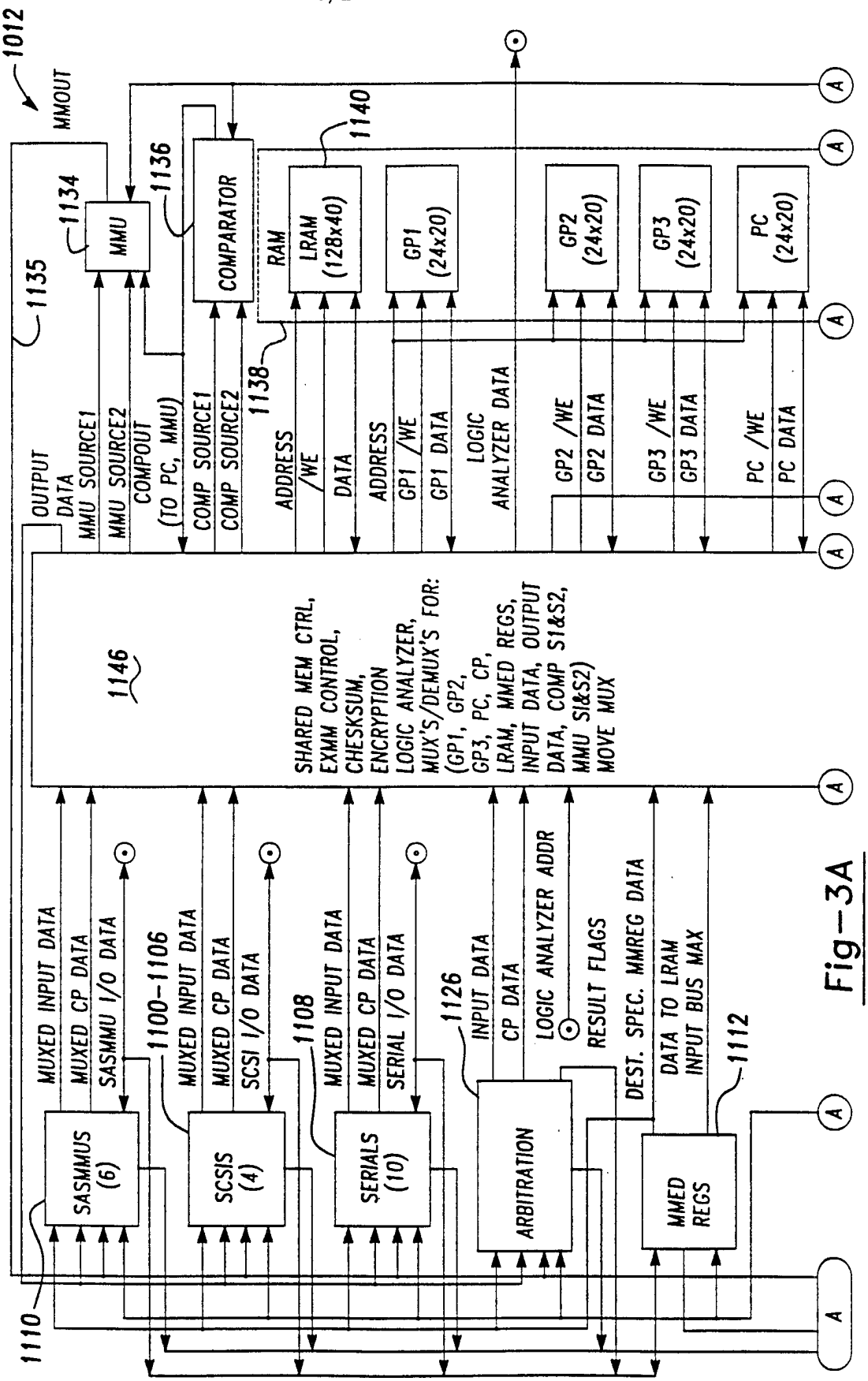

1012
MMOUT
1135
1134
MMU
1136
COMPARATOR
1140
RAM
LRAM (128x40)
GP1 (24x20)
GP2 (24x20)
GP3 (24x20)
PC (24x20)
OUTPUT DATA
MMU SOURCE1
MMU SOURCE2
COMPOUT (TO PC, MMU)
COMP SOURCE1
COMP SOURCE2
1138
ADDRESS /WE
DATA
ADDRESS
GP1 /WE
GP1 DATA
LOGIC ANALYZER DATA
GP2 /WE
GP2 DATA
GP3 /WE
GP3 DATA
PC /WE
PC DATA
1146
SHARED MEM CTRL, EXMM CONTROL, CHESKSUM, ENCRYPTION LOGIC ANALYZER, MUX'S/DEMUX'S FOR: (GP1, GP2, GP3, PC, CP, LRAM, MMED REGS, INPUT DATA, OUTPUT DATA, COMP S1&S2, MMU S1&S2) MOVE MUX
MUXED INPUT DATA
MUXED CP DATA
SASMMU I/O DATA
1100-1106
SASMMUS (6)
SCSI I/O DATA
SCSIS (4)
1108
SERIAL I/O DATA
SERIALS (10)
1126
INPUT DATA
CP DATA
LOGIC ANALYZER ADDR
RESULT FLAGS
ARBITRATION
DEST. SPEC. MMREG DATA
DATA TO LRAM INPUT BUS MAX
1112
MMED REGS
1110
A

Fig-3A

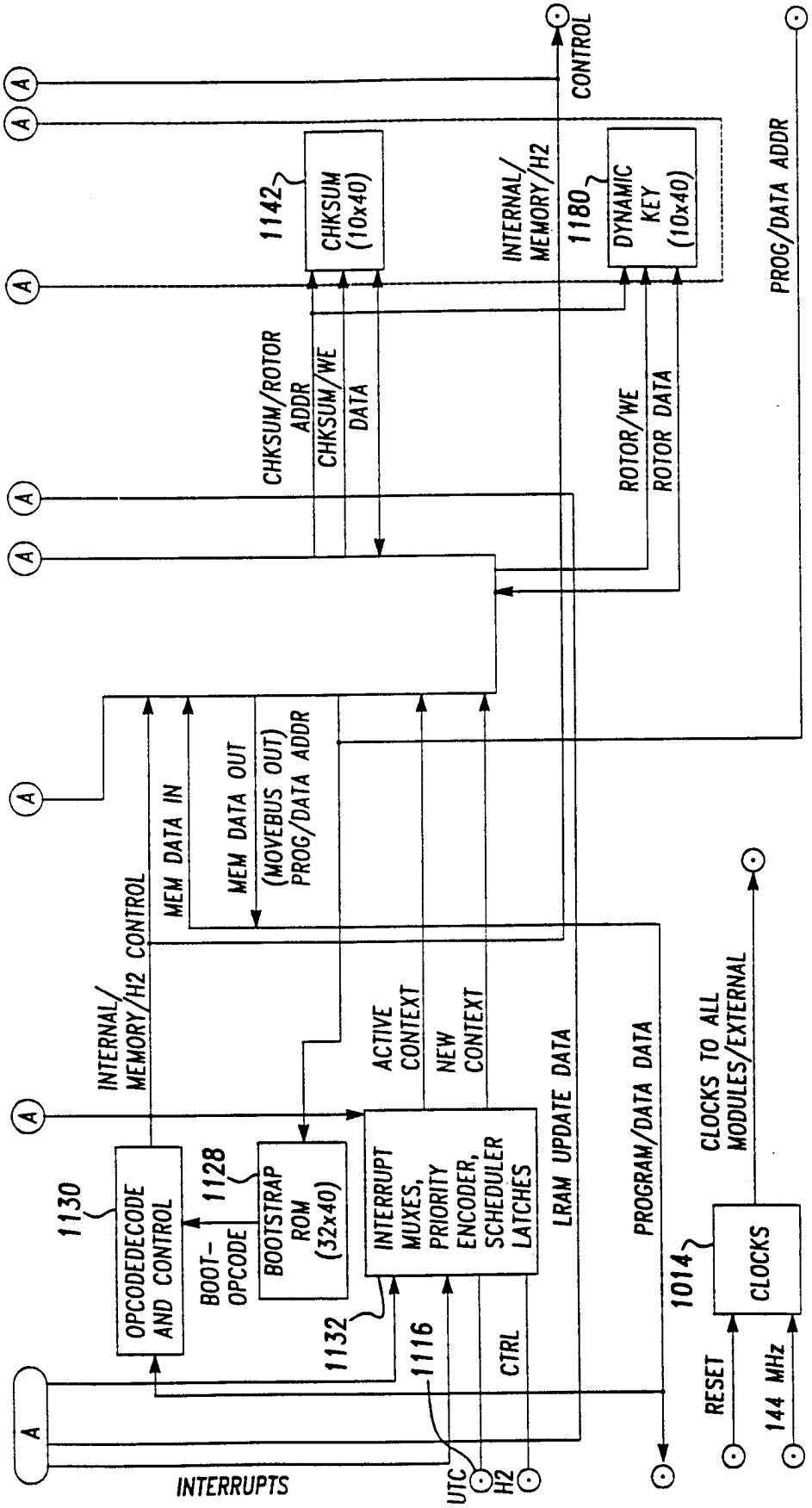

CONTROL
PROG/DATA ADDR
1142
CHKSUM (10x40)
INTERNAL/ MEMORY/H2
1180
DYNAMIC KEY (10x40)
CHKSUM/ROTOR ADDR
CHKSUM/WE
DATA
ROTOR/WE
ROTOR DATA
MEM DATA IN
MEM DATA OUT
(MOVEBUS OUT) PROG/DATA ADDR
INTERNAL/ MEMORY/H2 CONTROL
ACTIVE CONTEXT
NEW CONTEXT
LRAM UPDATE DATA
PROGRAM/DATA DATA
CLOCKS TO ALL MODULES/EXTERNAL
1130
OPCODEDECODE AND CONTROL
1128
BOOT-OPCODE
BOOTSTRAP ROM (32x40)
INTERRUPT MUXES, PRIORITY ENCODER, SCHEDULER LATCHES
1132
1116
CTRL
1014
CLOCKS
RESET
144 MHz
INTERRUPTS
UTC
H2
A

Fig-3B

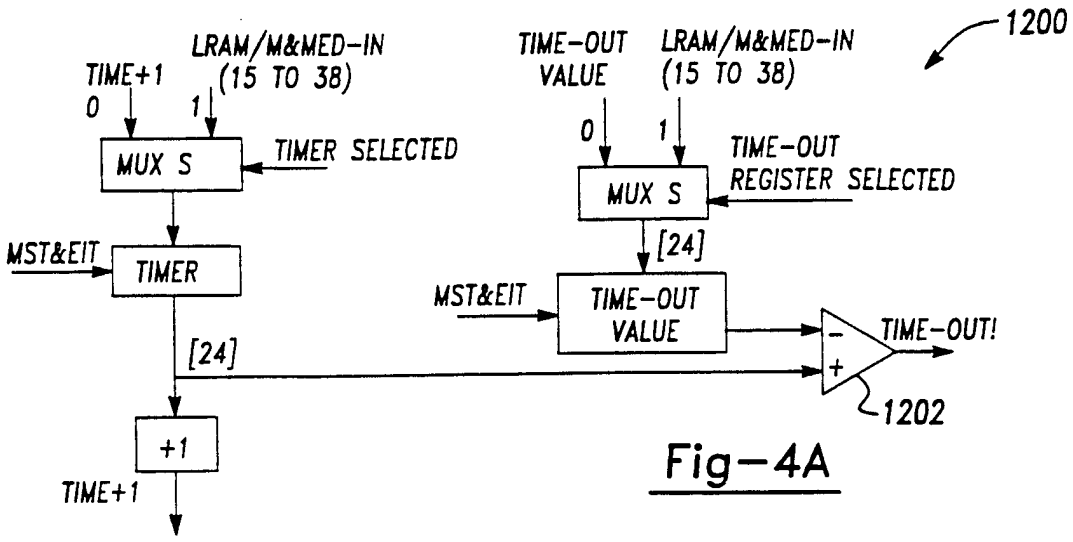
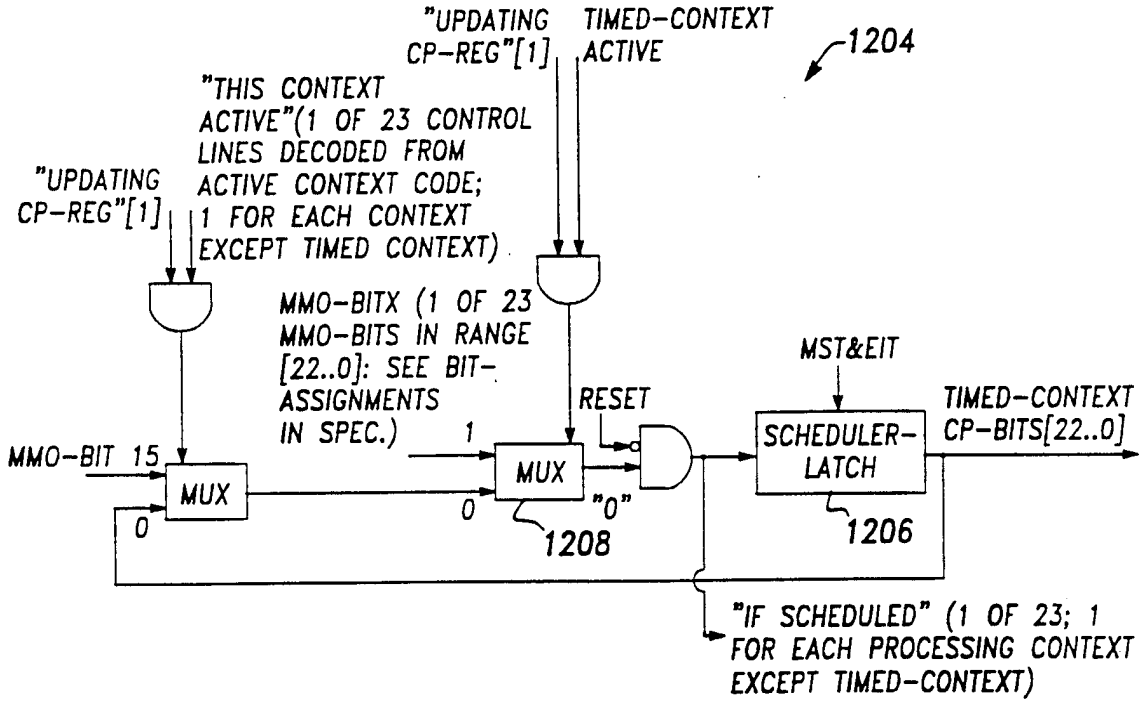


Fig-4A



(NOTE: EACH BIT OF TIMED-CONTEXT'S CP-BITS[22..0] IS CO-MAPPED INTO THE CP-BIT15 POSITION OF THE RESPECTIVE CONTEXT FOR WHICH THE BIT ACTS AS THE SCHEDULER BIT.)

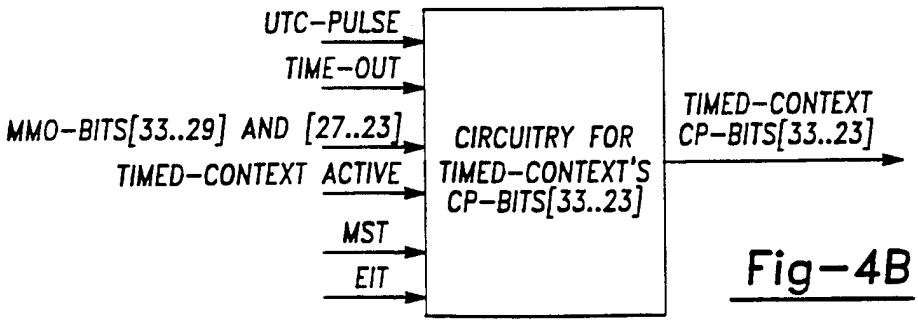


Fig-4B

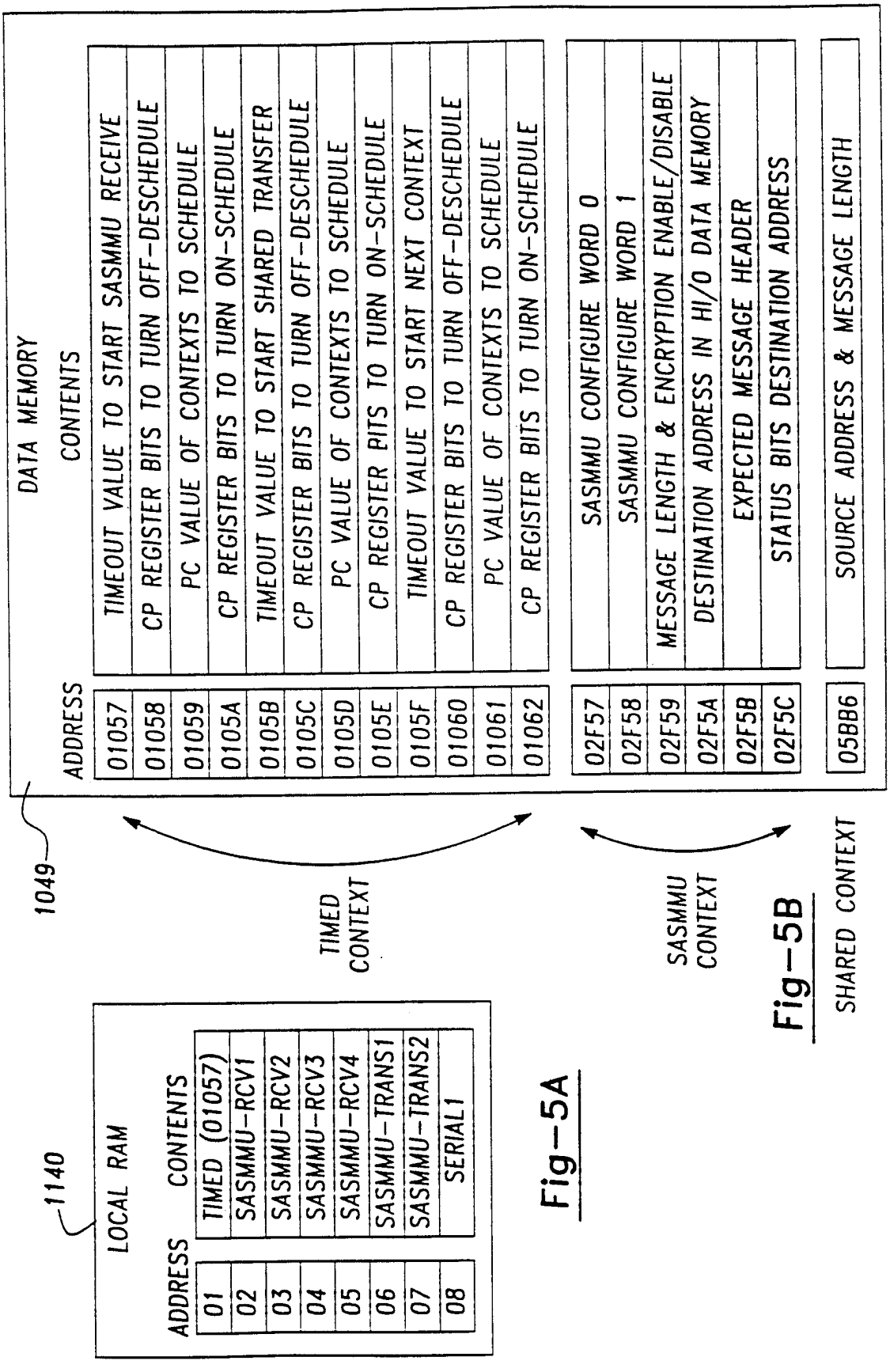

1140
LOCAL RAM
ADDRESS
CONTENTS
01
TIMED (01057)
02
SASMMU-RCV1
03
SASMMU-RCV2
04
SASMMU-RCV3
05
SASMMU-RCV4
06
SASMMU-TRANS1
07
SASMMU-TRANS2
08
SERIAL1
1049
DATA MEMORY
ADDRESS
CONTENTS
01057
TIMEOUT VALUE TO START SASMMU RECEIVE
01058
CP REGISTER BITS TO TURN OFF-DESCHEDULE
01059
PC VALUE OF CONTEXTS TO SCHEDULE
0105A
CP REGISTER BITS TO TURN ON-SCHEDULE
0105B
TIMEOUT VALUE TO START SHARED TRANSFER
0105C
CP REGISTER BITS TO TURN OFF-DESCHEDULE
0105D
PC VALUE OF CONTEXTS TO SCHEDULE
0105E
CP REGISTER BITS TO TURN ON-SCHEDULE
0105F
TIMEOUT VALUE TO START NEXT CONTEXT
01060
CP REGISTER BITS TO TURN OFF-DESCHEDULE
01061
PC VALUE OF CONTEXTS TO SCHEDULE
01062
CP REGISTER BITS TO TURN ON-SCHEDULE
TIMED CONTEXT
02F57
SASMMU CONFIGURE WORD 0
02F58
SASMMU CONFIGURE WORD 1
02F59
MESSAGE LENGTH & ENCRYPTION ENABLE/DISABLE
02F5A
DESTINATION ADDRESS IN HI/O DATA MEMORY
02F5B
EXPECTED MESSAGE HEADER
02F5C
STATUS BITS DESTINATION ADDRESS
SASMMU CONTEXT
05BB6
SOURCE ADDRESS & MESSAGE LENGTH
SHARED CONTEXT

Fig-5A

Fig-5B

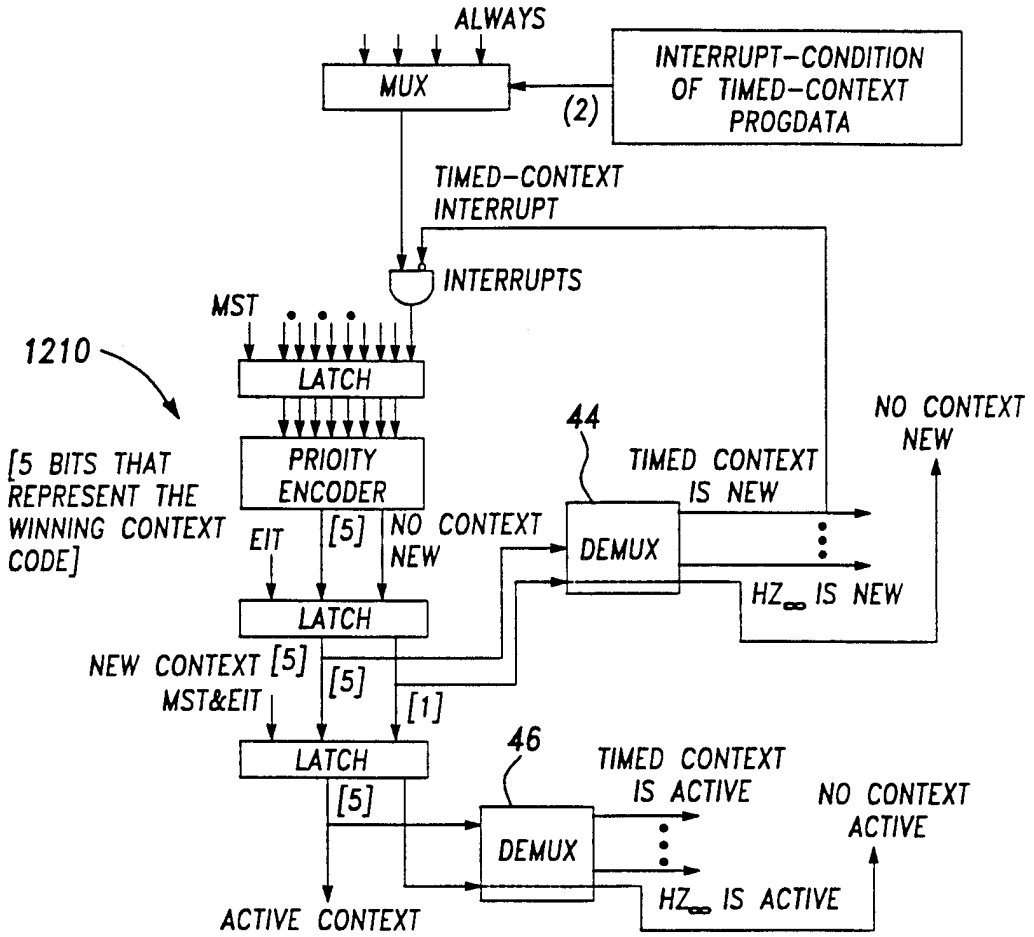

ALWAYS
MUX
(2)
INTERRUPT-CONDITION OF TIMED-CONTEXT PROGDATA
TIMED-CONTEXT INTERRUPT
INTERRUPTS
MST
1210
LATCH
[5 BITS THAT REPRESENT THE WINNING CONTEXT CODE]
PRIOITY ENCODER
EIT
[5]
NO CONTEXT NEW
44
TIMED CONTEXT IS NEW
NO CONTEXT NEW
DEMUX
HZ∞ IS NEW
LATCH
NEW CONTEXT [5]
MST&EIT
[5]
[1]
46
LATCH
[5]
TIMED CONTEXT IS ACTIVE
NO CONTEXT ACTIVE
DEMUX
HZ∞ IS ACTIVE
ACTIVE CONTEXT

Fig-6

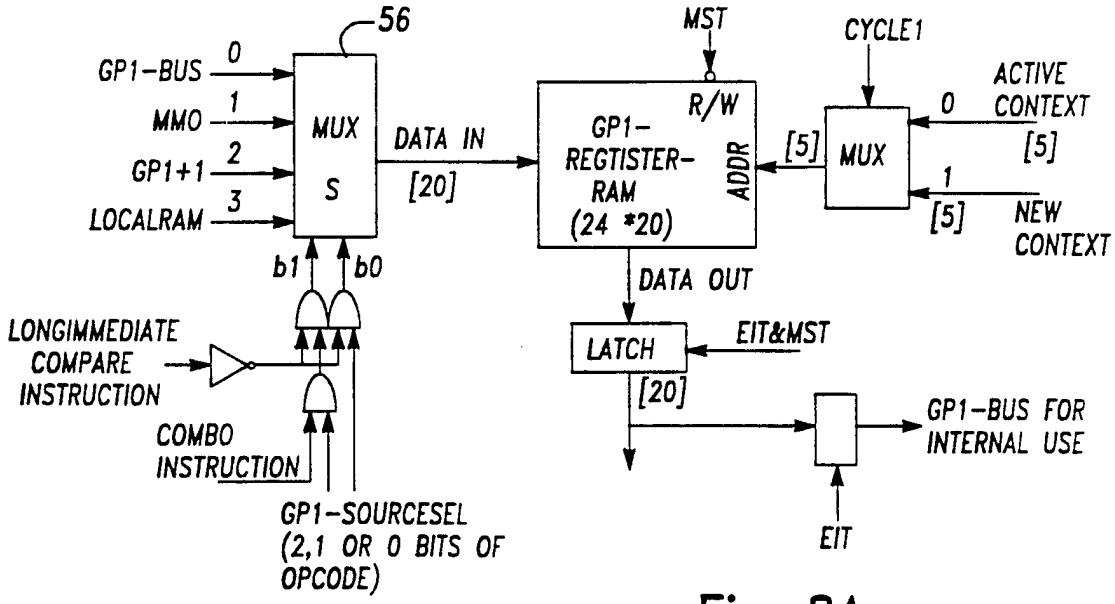

56
GP1-BUS
MMO
GP1+1
LOCALRAM
0
1
2
3
MUX
S
DATA IN
[20]
b1
b0
LONGIMMEDIATE COMPARE INSTRUCTION
COMBO INSTRUCTION
GP1-SOURCESEL (2,1 OR 0 BITS OF OPCODE)
MST
R/W
GP1-REGTISTER-RAM (24 *20)
ADDR
[5]
CYCLE1
MUX
0
ACTIVE CONTEXT
[5]
1
[5]
NEW CONTEXT
DATA OUT
LATCH
EIT&MST
[20]
GP1-BUS FOR INTERNAL USE
EIT

Fig-8A

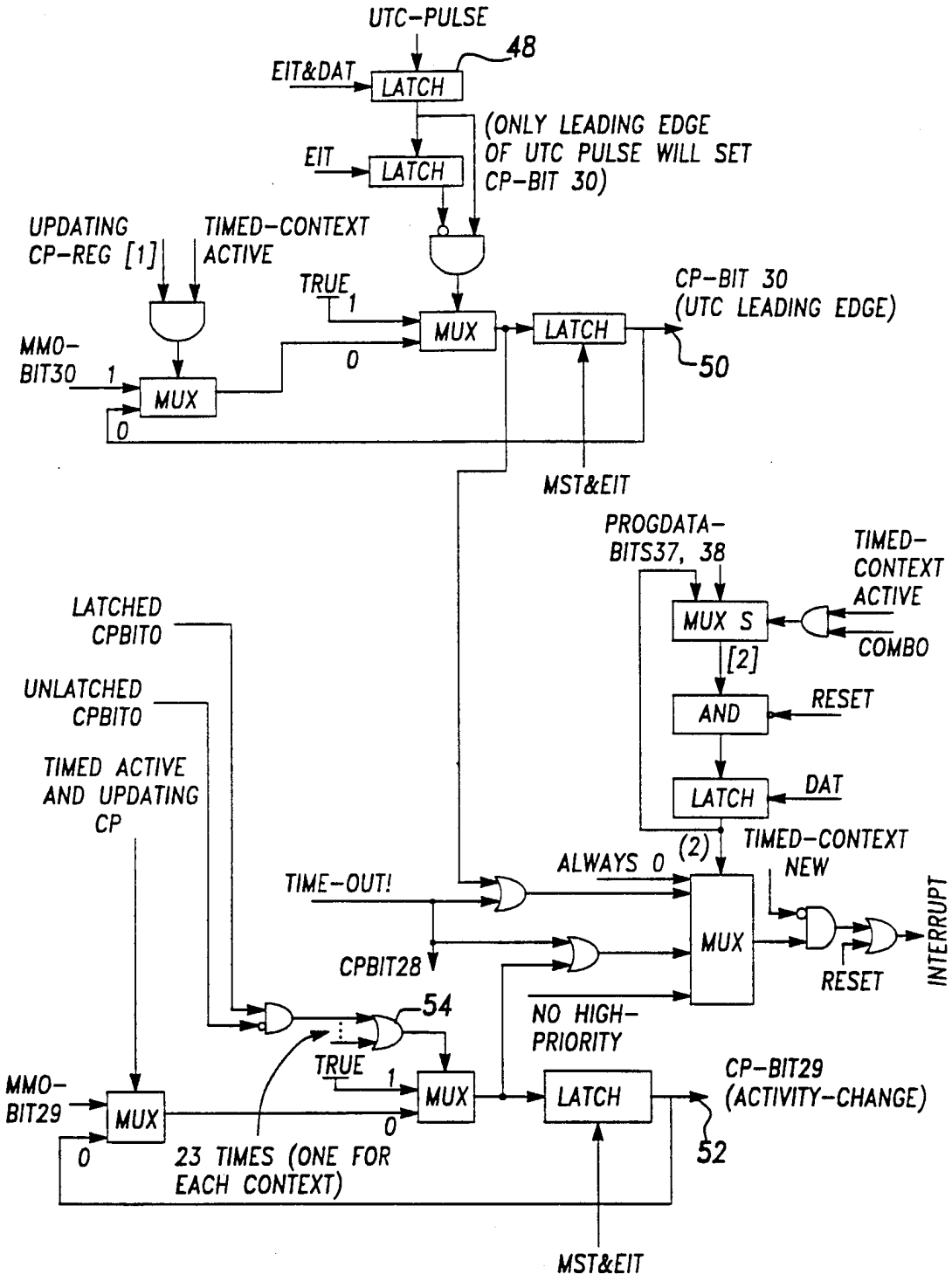

UTC-PULSE
48
EIT&DAT
LATCH
(ONLY LEADING EDGE OF UTC PULSE WILL SET CP-BIT 30)
EIT
LATCH
UPDATING CP-REG [1]
TIMED-CONTEXT ACTIVE
TRUE
1
MUX
LATCH
CP-BIT 30 (UTC LEADING EDGE)
50
MMO-BIT30
1
MUX
0
0
MST&EIT
PROGDATA-BITS37, 38
TIMED-CONTEXT ACTIVE
MUX S
COMBO
[2]
AND
RESET
LATCH
DAT
LATCHED CPBIT0
UNLATCHED CPBIT0
TIMED ACTIVE AND UPDATING CP
(2)
TIMED-CONTEXT NEW
ALWAYS 0
TIME-OUT!
MUX
INTERRUPT
RESET
CPBIT28
54
NO HIGH-PRIORITY
TRUE
1
MMO-BIT29
MUX
MUX
LATCH
CP-BIT29 (ACTIVITY-CHANGE)
0
0
52
23 TIMES (ONE FOR EACH CONTEXT)
MST&EIT

Fig-7

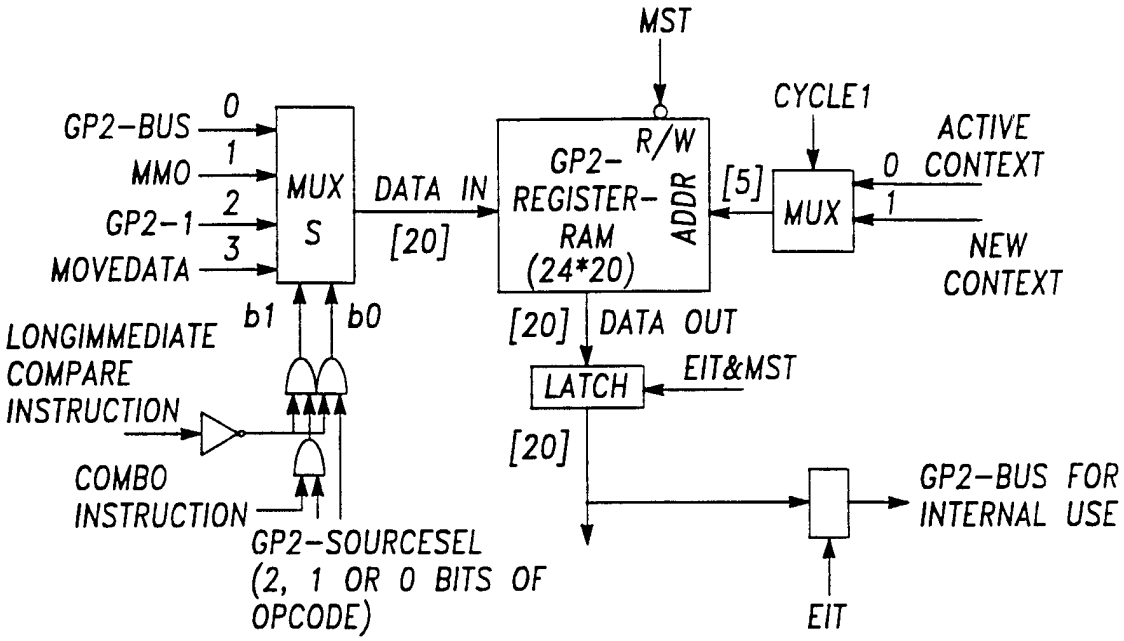

MST
GP2-BUS
MMO
GP2-1
MOVEDATA
0
1
2
3
MUX
S
DATA IN
[20]
GP2-
REGISTER-
RAM
(24*20)
R/W
ADDR
[5]
CYCLE1
MUX
0
1
ACTIVE
CONTEXT
NEW
CONTEXT
b1
b0
LONGIMMEDIATE
COMPARE
INSTRUCTION
COMBO
INSTRUCTION
GP2-SOURCESEL
(2, 1 OR 0 BITS OF
OPCODE)
[20]
DATA OUT
LATCH
EIT&MST
[20]
GP2-BUS FOR
INTERNAL USE
EIT

Fig-8B

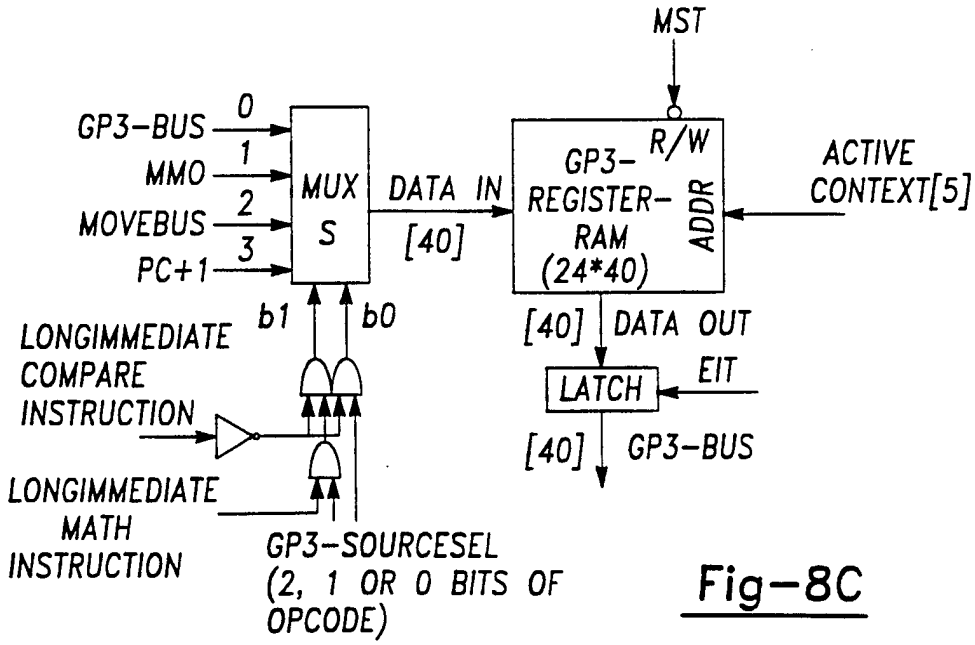

MST
GP3-BUS
MMO
MOVEBUS
PC+1
0
1
2
3
MUX
S
DATA IN
[40]
GP3-
REGISTER-
RAM
(24*40)
R/W
ADDR
ACTIVE
CONTEXT[5]
b1
b0
LONGIMMEDIATE
COMPARE
INSTRUCTION
LONGIMMEDIATE
MATH
INSTRUCTION
GP3-SOURCESEL
(2, 1 OR 0 BITS OF
OPCODE)
[40]
DATA OUT
LATCH
EIT
[40]
GP3-BUS

Fig-8C

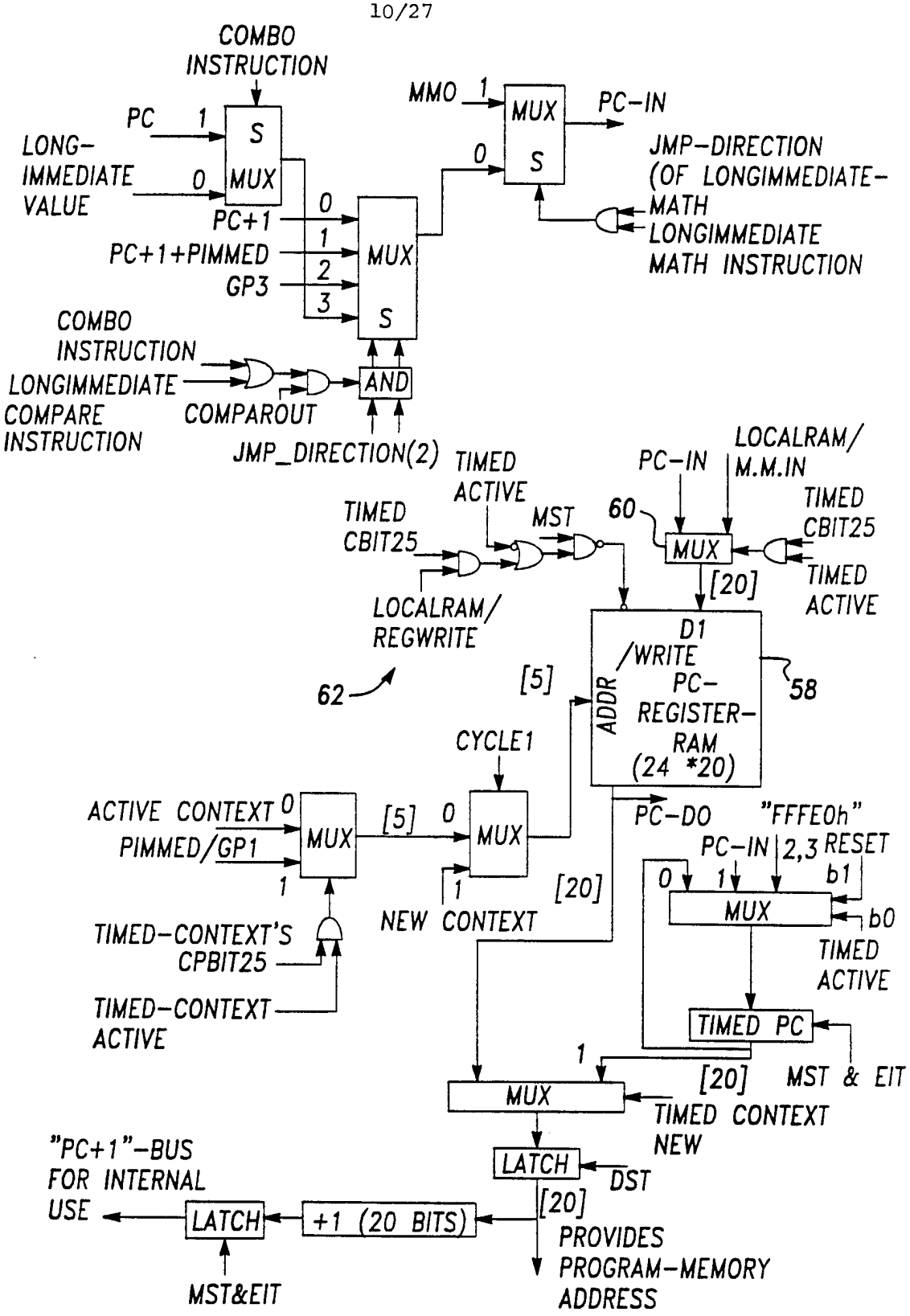

COMBO INSTRUCTION
PC
LONG-IMMEDIATE VALUE
MUX
PC+1
PC+1+PIMMED
GP3
MMO
PC-IN
JMP-DIRECTION (OF LONGIMMEDIATE-MATH LONGIMMEDIATE MATH INSTRUCTION
COMBO INSTRUCTION
LONGIMMEDIATE COMPARE INSTRUCTION
COMPAROUT
AND
JMP_DIRECTION(2)
TIMED ACTIVE
TIMED CBIT25
MST
LOCALRAM/ REGWRITE
62
PC-IN
LOCALRAM/ M.M.IN
60
TIMED CBIT25
TIMED ACTIVE
[20]
D1 /WRITE PC-REGISTER-RAM (24 *20)
ADDR
[5]
58
CYCLE1
ACTIVE CONTEXT
PIMMED/GP1
NEW CONTEXT
PC-DO
"FFFE0h"
PC-IN
2,3 RESET
b1
b0
TIMED-CONTEXT'S CPBIT25
TIMED-CONTEXT ACTIVE
TIMED PC
MST & EIT
TIMED CONTEXT NEW
"PC+1"-BUS FOR INTERNAL USE
LATCH
DST
+1 (20 BITS)
PROVIDES PROGRAM-MEMORY ADDRESS
MST&EIT

Fig-8D

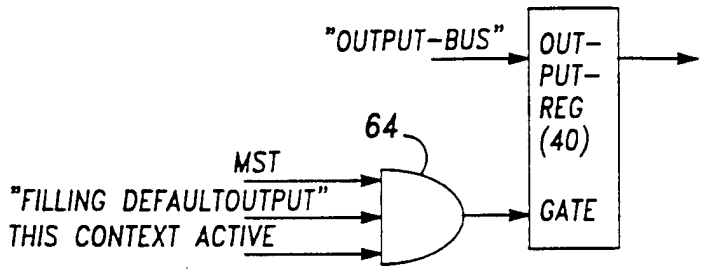


Fig-8E

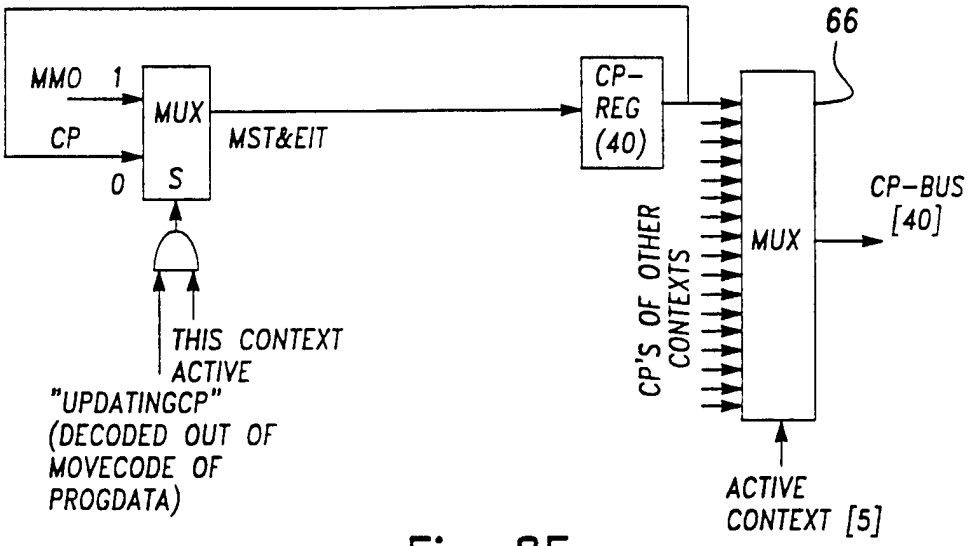


Fig-8F

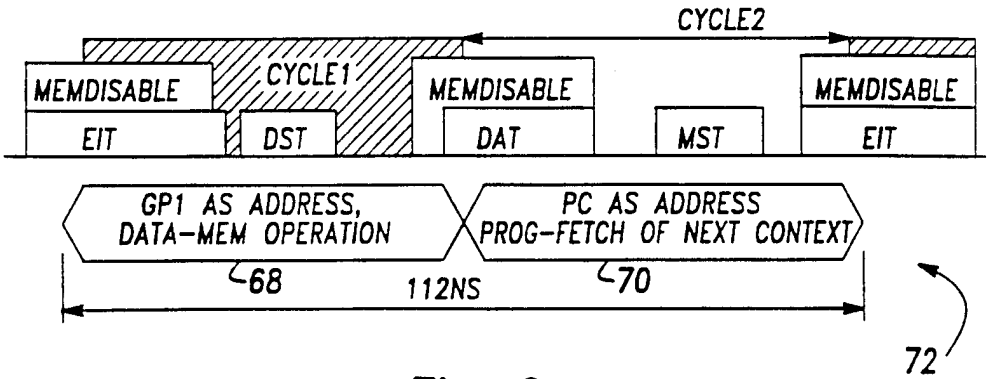


Fig-9

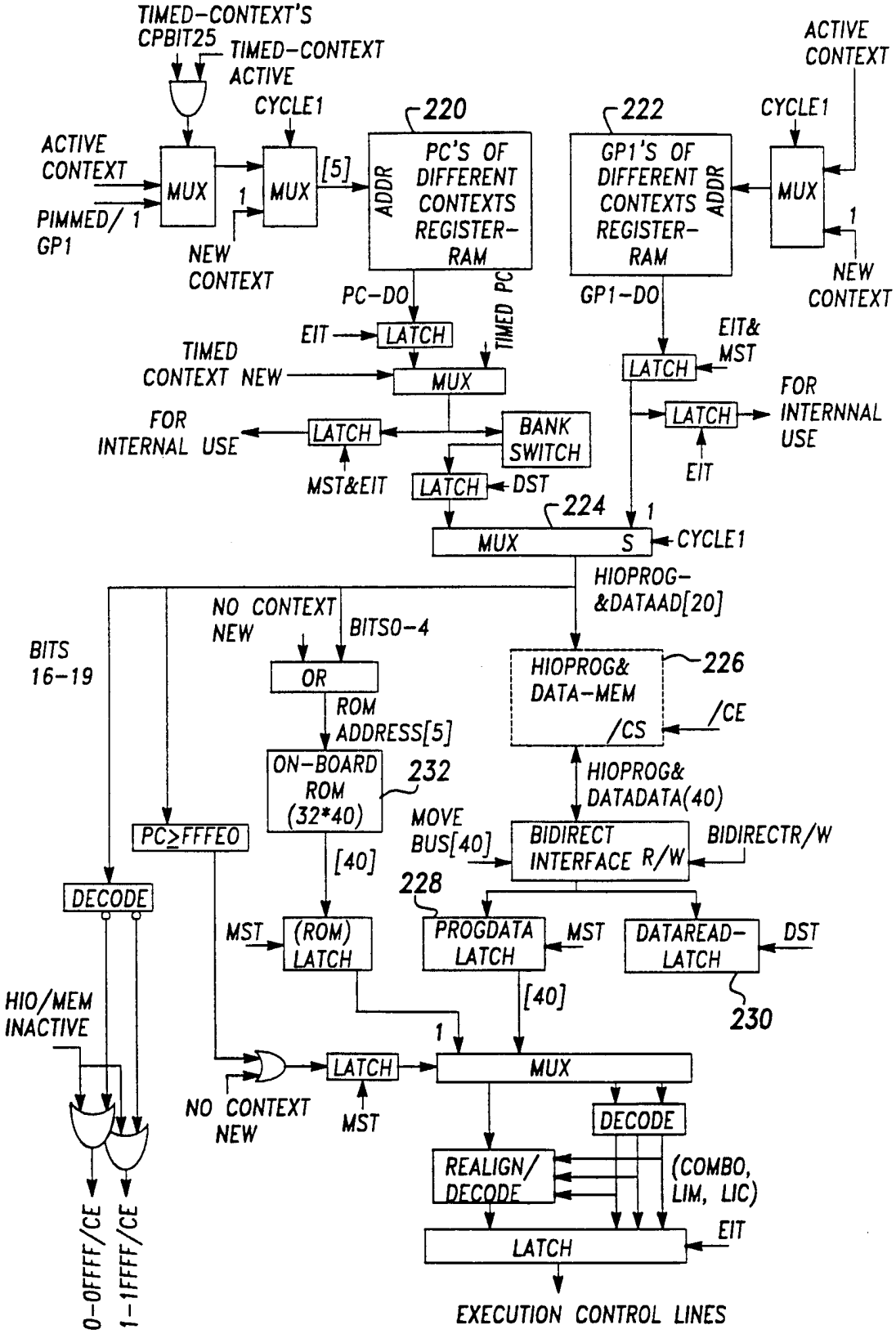

TIMED-CONTEXT'S CPBIT25
TIMED-CONTEXT ACTIVE
CYCLE1
220
PC'S OF DIFFERENT CONTEXTS REGISTER-RAM
ADDR
222
GP1'S OF DIFFERENT CONTEXTS REGISTER-RAM
ADDR
ACTIVE CONTEXT
CYCLE1
ACTIVE CONTEXT
MUX
MUX
[5]
PIMMED/ GP1
1
NEW CONTEXT
MUX
1
NEW CONTEXT
PC-DO
TIMED PC
GP1-DO
EIT
LATCH
TIMED CONTEXT NEW
MUX
EIT& MST
LATCH
FOR INTERNNAL USE
LATCH
FOR INTERNAL USE
LATCH
BANK SWITCH
MST&EIT
LATCH
DST
EIT
224
1
MUX
S
CYCLE1
HIOPROG-&DATAAD[20]
NO CONTEXT NEW
BITS0-4
BITS 16-19
OR
HIOPROG& DATA-MEM
226
/CS
/CE
ROM ADDRESS[5]
ON-BOARD ROM (32*40)
232
HIOPROG& DATADATA(40)
PC≥FFFE0
MOVE BUS[40]
BIDIRECT INTERFACE R/W
BIDIRECTR/W
[40]
228
DECODE
MST
(ROM) LATCH
PROGDATA LATCH
MST
DATAREAD-LATCH
DST
230
HIO/MEM INACTIVE
[40]
1
LATCH
MUX
NO CONTEXT NEW
MST
DECODE
REALIGN/ DECODE
(COMBO, LIM, LIC)
EIT
LATCH
0-0FFFF/CE
1-1FFFF/CE
EXECUTION CONTROL LINES

Fig-10A

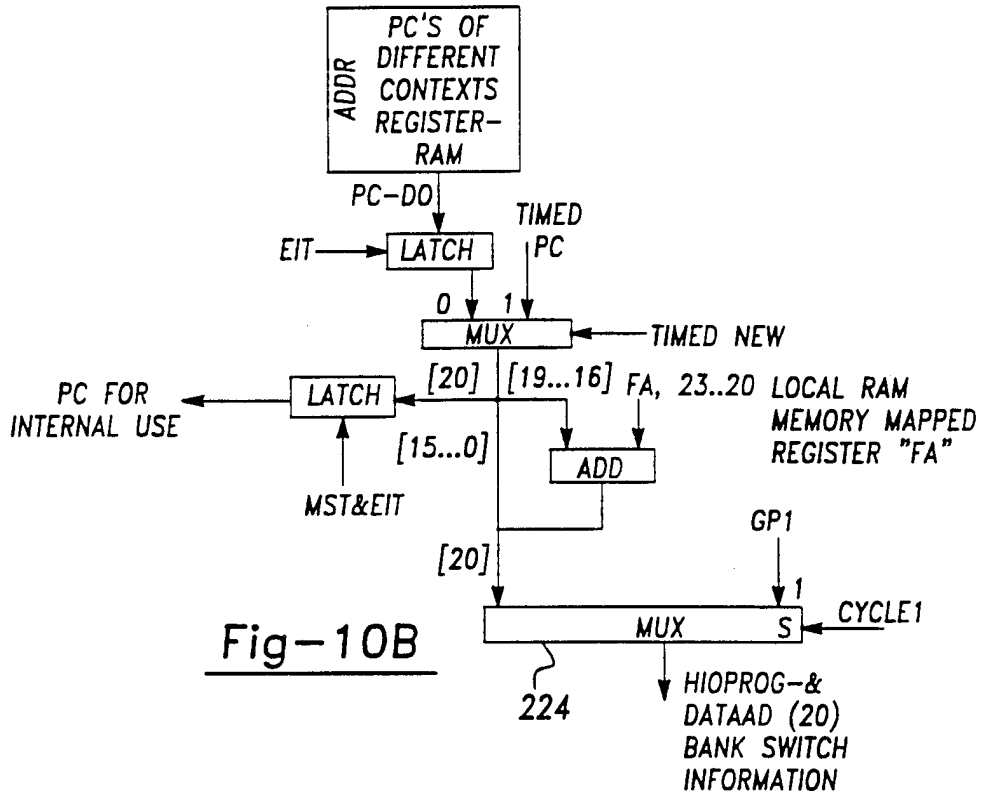


Fig-10B

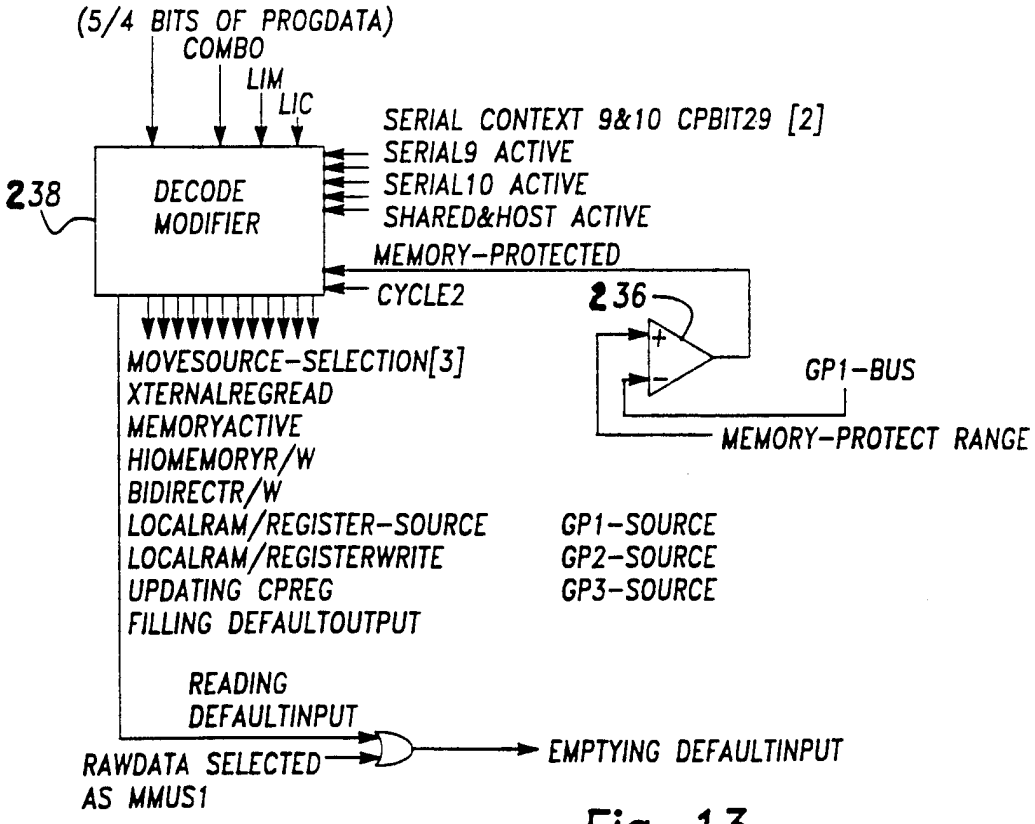


Fig-13

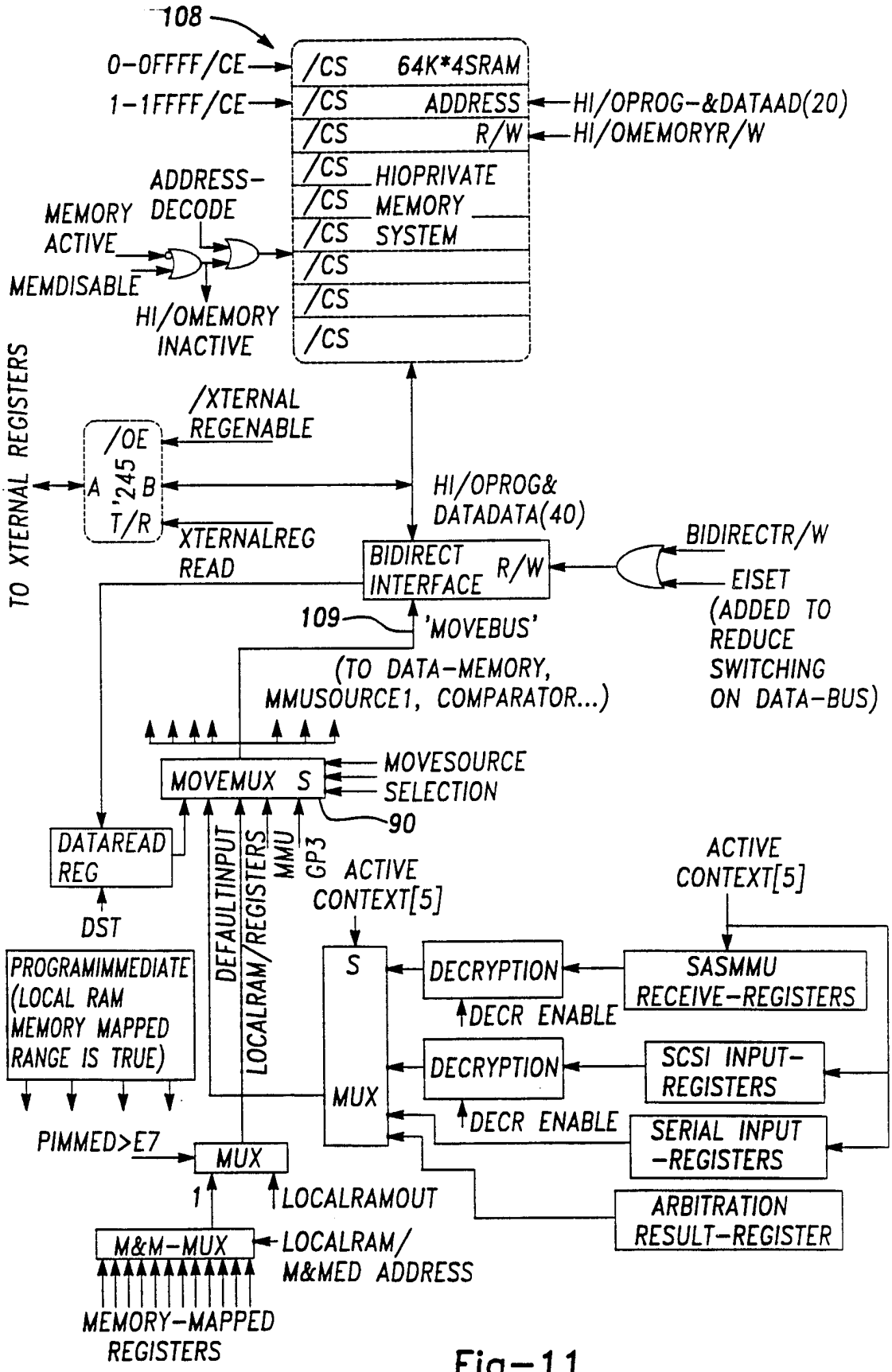

108
0-0FFFF/CE
1-1FFFF/CE
/CS 64K*4SRAM
/CS ADDRESS
/CS R/W
HI/OPROG-&DATAAD(20)
HI/OMEMORYR/W
/CS HIOPRIVATE
/CS MEMORY
/CS SYSTEM
ADDRESS-DECODE
MEMORY ACTIVE
MEMDISABLE
HI/OMEMORY INACTIVE
TO XTERNAL REGISTERS
/XTERNAL REGENABLE
/OE
A '245 B
T/R
XTERNALREG READ
HI/OPROG& DATADATA(40)
BIDIRECT INTERFACE R/W
BIDIRECTR/W
EISET
(ADDED TO REDUCE SWITCHING ON DATA-BUS)
109
'MOVEBUS'
(TO DATA-MEMORY, MMUSOURCE1, COMPARATOR...)
MOVEMUX S
MOVESOURCE SELECTION
90
DATAREAD REG
DST
DEFAULTINPUT
LOCALRAM/REGISTERS
MMU
GP3
ACTIVE CONTEXT[5]
ACTIVE CONTEXT[5]
PROGRAMIMMEDIATE (LOCAL RAM MEMORY MAPPED RANGE IS TRUE)
S
MUX
DECRYPTION
DECR ENABLE
SASMMU RECEIVE-REGISTERS
DECRYPTION
DECR ENABLE
SCSI INPUT-REGISTERS
SERIAL INPUT -REGISTERS
ARBITRATION RESULT-REGISTER
PIMMED>E7
MUX
1
LOCALRAMOUT
M&M-MUX
LOCALRAM/ M&MED ADDRESS
MEMORY-MAPPED REGISTERS

Fig-11

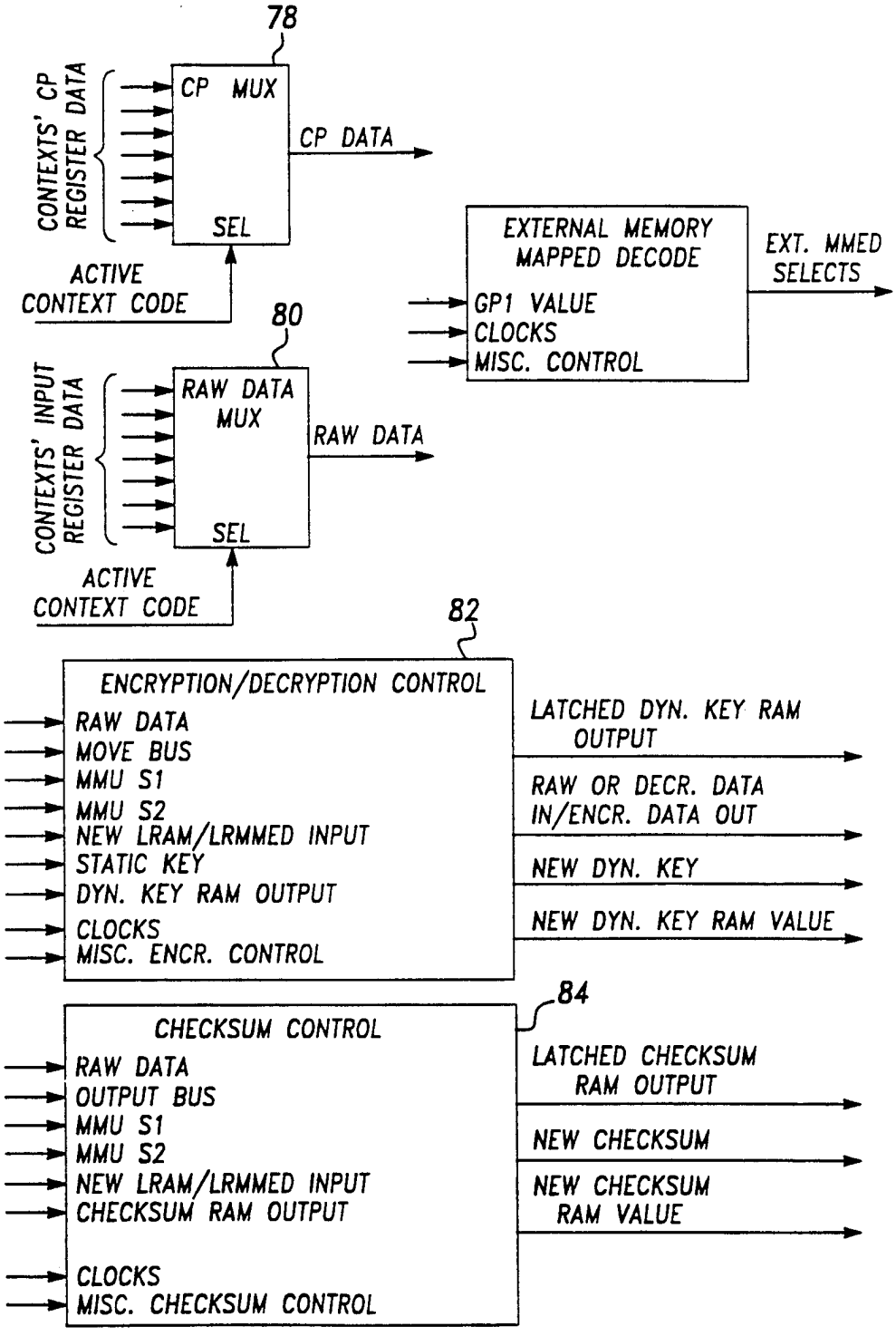

78
CONTEXTS' CP REGISTER DATA
CP MUX
CP DATA
SEL
ACTIVE CONTEXT CODE
EXTERNAL MEMORY MAPPED DECODE
EXT. MMED SELECTS
GP1 VALUE
CLOCKS
MISC. CONTROL
80
CONTEXTS' INPUT REGISTER DATA
RAW DATA MUX
RAW DATA
SEL
ACTIVE CONTEXT CODE
82
ENCRYPTION/DECRYPTION CONTROL
RAW DATA
MOVE BUS
MMU S1
MMU S2
NEW LRAM/LRMMED INPUT
STATIC KEY
DYN. KEY RAM OUTPUT
CLOCKS
MISC. ENCR. CONTROL
LATCHED DYN. KEY RAM OUTPUT
RAW OR DECR. DATA IN/ENCR. DATA OUT
NEW DYN. KEY
NEW DYN. KEY RAM VALUE
84
CHECKSUM CONTROL
RAW DATA
OUTPUT BUS
MMU S1
MMU S2
NEW LRAM/LRMMED INPUT
CHECKSUM RAM OUTPUT
CLOCKS
MISC. CHECKSUM CONTROL
LATCHED CHECKSUM RAM OUTPUT
NEW CHECKSUM
NEW CHECKSUM RAM VALUE

Fig-12A

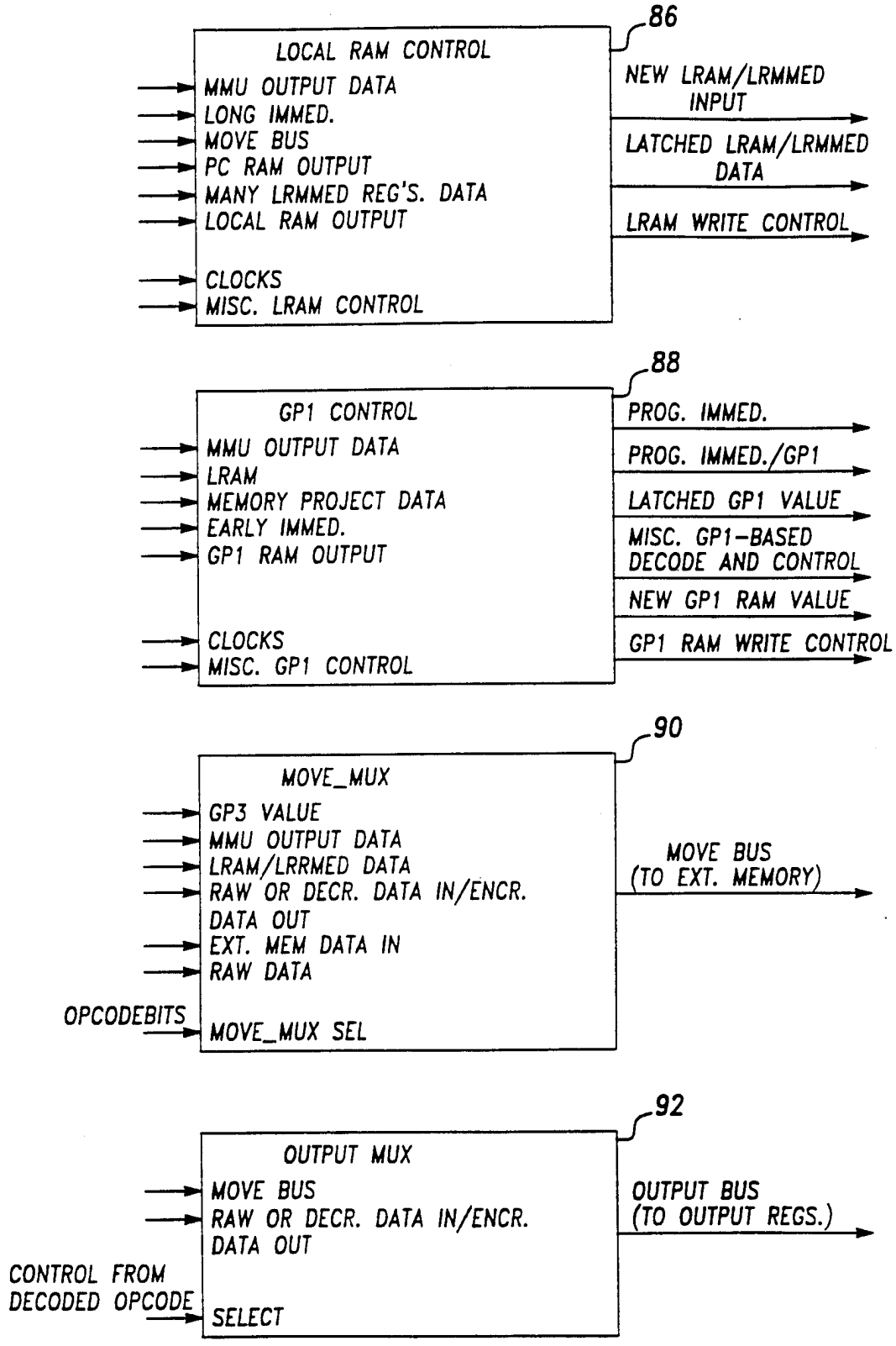

86
LOCAL RAM CONTROL
MMU OUTPUT DATA
LONG IMMED.
MOVE BUS
PC RAM OUTPUT
MANY LRMMED REG'S. DATA
LOCAL RAM OUTPUT
CLOCKS
MISC. LRAM CONTROL
NEW LRAM/LRMMED INPUT
LATCHED LRAM/LRMMED DATA
LRAM WRITE CONTROL
88
GP1 CONTROL
MMU OUTPUT DATA
LRAM
MEMORY PROJECT DATA
EARLY IMMED.
GP1 RAM OUTPUT
CLOCKS
MISC. GP1 CONTROL
PROG. IMMED.
PROG. IMMED./GP1
LATCHED GP1 VALUE
MISC. GP1-BASED DECODE AND CONTROL
NEW GP1 RAM VALUE
GP1 RAM WRITE CONTROL
90
MOVE_MUX
GP3 VALUE
MMU OUTPUT DATA
LRAM/LRRMED DATA
RAW OR DECR. DATA IN/ENCR. DATA OUT
EXT. MEM DATA IN
RAW DATA
OPCODEBITS
MOVE_MUX SEL
MOVE BUS (TO EXT. MEMORY)
92
OUTPUT MUX
MOVE BUS
RAW OR DECR. DATA IN/ENCR. DATA OUT
CONTROL FROM DECODED OPCODE
SELECT
OUTPUT BUS (TO OUTPUT REGS.)

Fig-12B

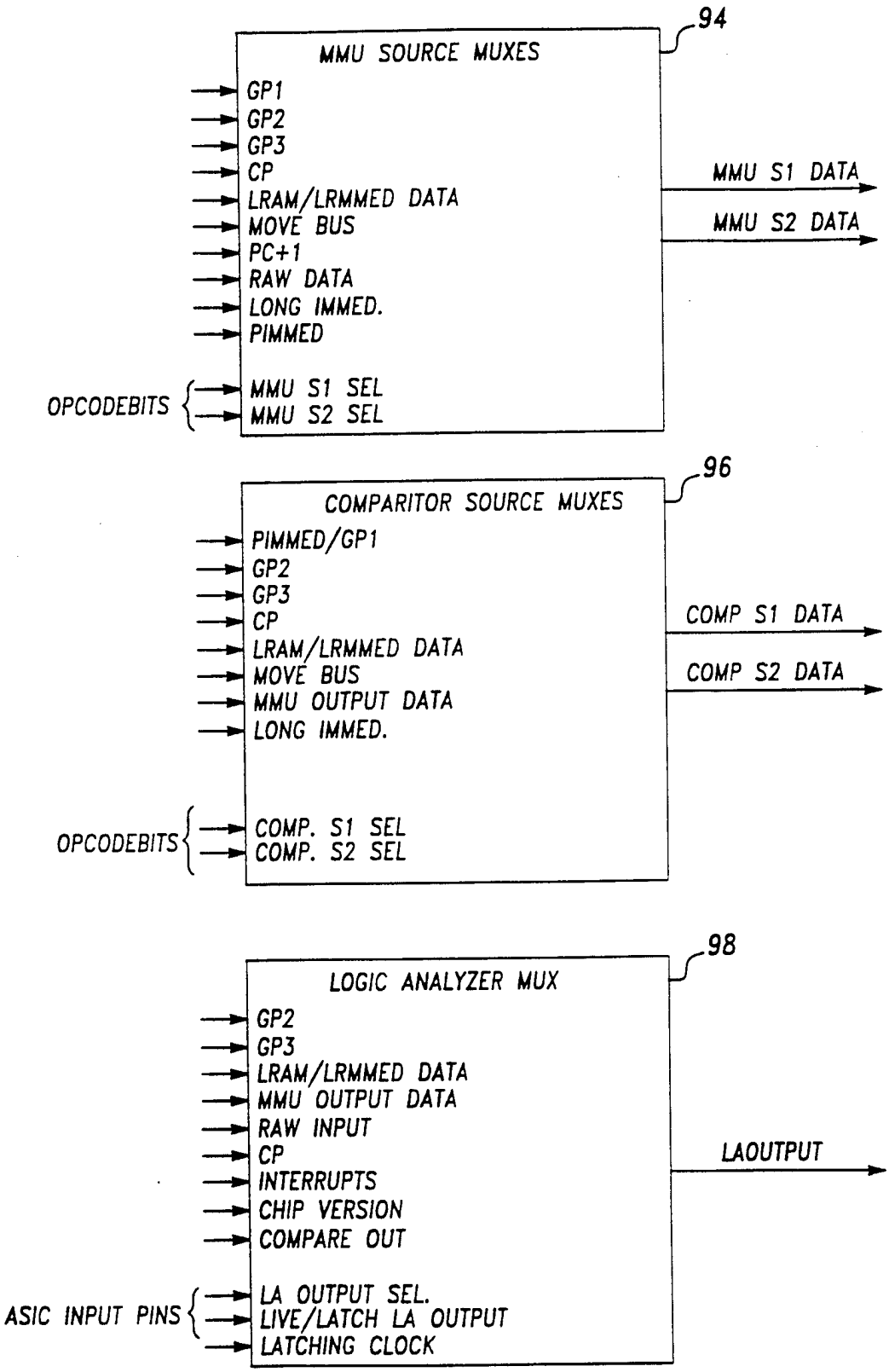

94
MMU SOURCE MUXES
GP1
GP2
GP3
CP
LRAM/LRMMED DATA
MOVE BUS
PC+1
RAW DATA
LONG IMMED.
PIMMED
OPCODEBITS
MMU S1 SEL
MMU S2 SEL
MMU S1 DATA
MMU S2 DATA
96
COMPARITOR SOURCE MUXES
PIMMED/GP1
GP2
GP3
CP
LRAM/LRMMED DATA
MOVE BUS
MMU OUTPUT DATA
LONG IMMED.
OPCODEBITS
COMP. S1 SEL
COMP. S2 SEL
COMP S1 DATA
COMP S2 DATA
98
LOGIC ANALYZER MUX
GP2
GP3
LRAM/LRMMED DATA
MMU OUTPUT DATA
RAW INPUT
CP
INTERRUPTS
CHIP VERSION
COMPARE OUT
ASIC INPUT PINS
LA OUTPUT SEL.
LIVE/LATCH LA OUTPUT
LATCHING CLOCK
LAOUTPUT

Fig-12C

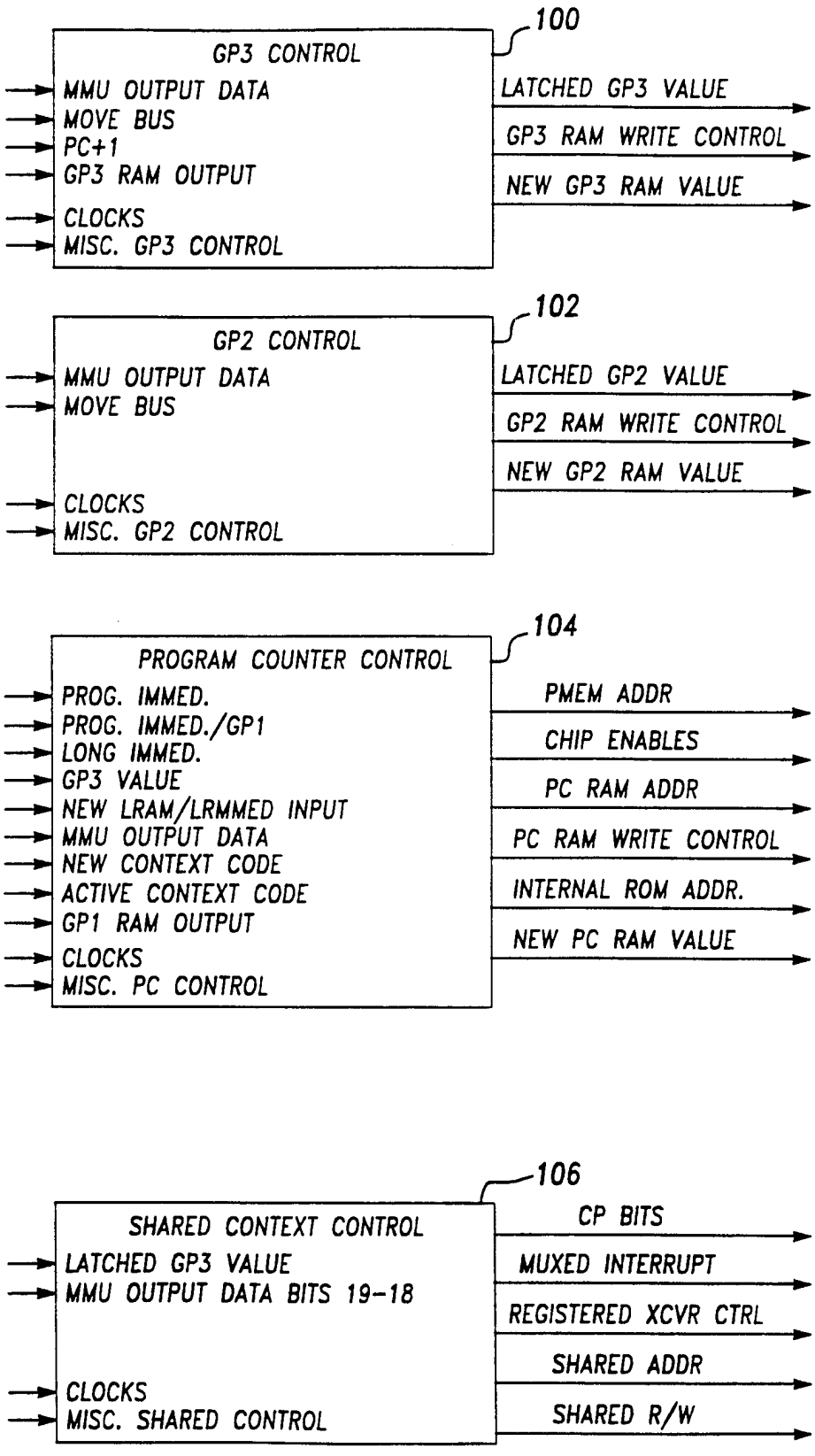
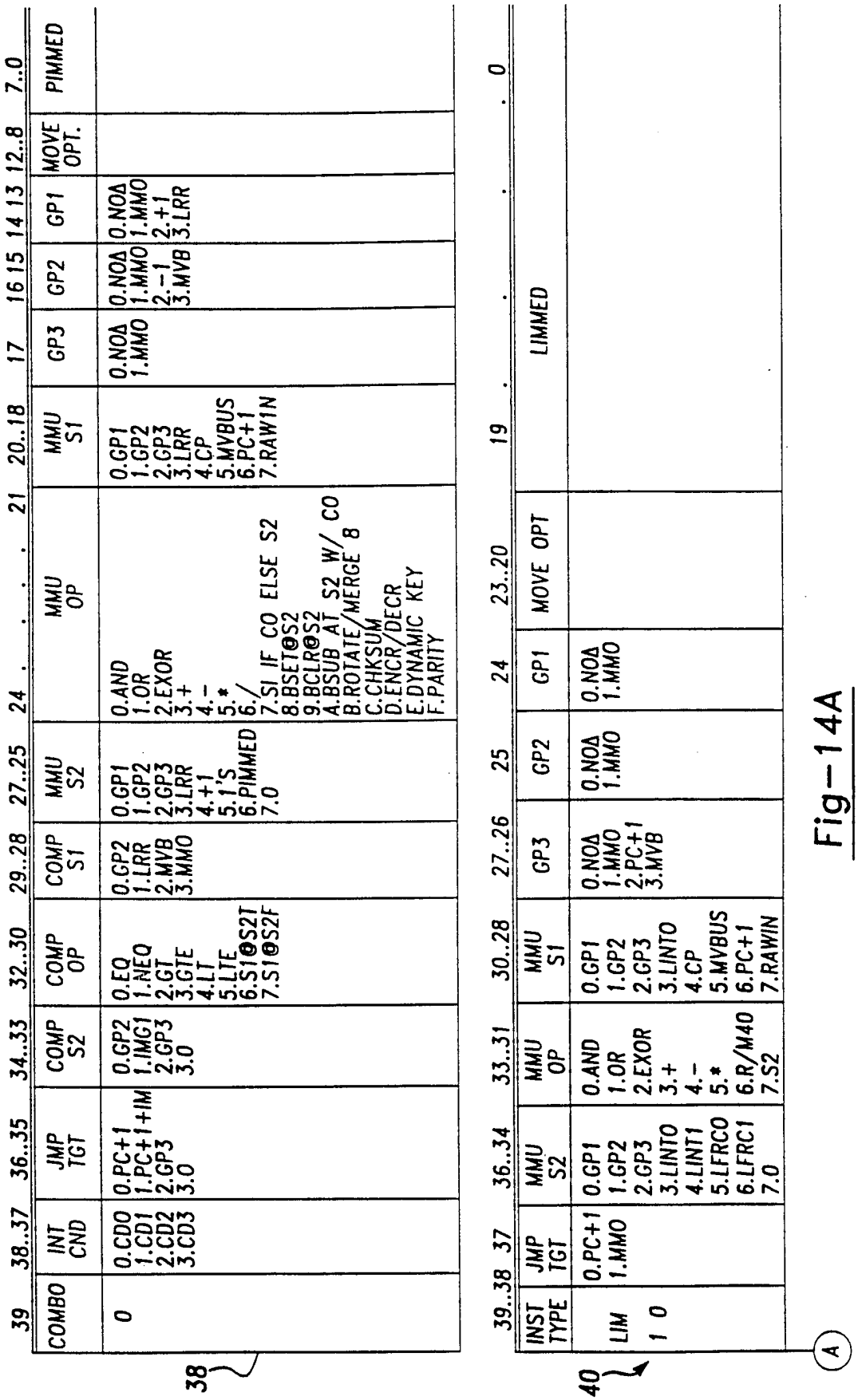

100
GP3 CONTROL
MMU OUTPUT DATA
MOVE BUS
PC+1
GP3 RAM OUTPUT
CLOCKS
MISC. GP3 CONTROL
LATCHED GP3 VALUE
GP3 RAM WRITE CONTROL
NEW GP3 RAM VALUE
102
GP2 CONTROL
MMU OUTPUT DATA
MOVE BUS
CLOCKS
MISC. GP2 CONTROL
LATCHED GP2 VALUE
GP2 RAM WRITE CONTROL
NEW GP2 RAM VALUE
104
PROGRAM COUNTER CONTROL
PROG. IMMED.
PROG. IMMED./GP1
LONG IMMED.
GP3 VALUE
NEW LRAM/LRMMED INPUT
MMU OUTPUT DATA
NEW CONTEXT CODE
ACTIVE CONTEXT CODE
GP1 RAM OUTPUT
CLOCKS
MISC. PC CONTROL
PMEM ADDR
CHIP ENABLES
PC RAM ADDR
PC RAM WRITE CONTROL
INTERNAL ROM ADDR.
NEW PC RAM VALUE
106
SHARED CONTEXT CONTROL
LATCHED GP3 VALUE
MMU OUTPUT DATA BITS 19-18
CLOCKS
MISC. SHARED CONTROL
CP BITS
MUXED INTERRUPT
REGISTERED XCVR CTRL
SHARED ADDR
SHARED R/W

Fig-12D


39
38..37
36..35
34..33
32..30
29..28
27..25
24
21
20..18
17
16 15
14 13
12..8
7..0
COMBO
INT CND
JMP TGT
COMP S2
COMP OP
COMP S1
MMU S2
MMU OP
MMU S1
GP3
GP2
GP1
MOVE OPT.
PIMMED
0
0.CD0
1.CD1
2.CD2
3.CD3
0.PC+1
1.PC+1+IM
2.GP3
3.0
0.GP2
1.IMG1
2.GP3
3.0
0.EQ
1.NEQ
2.GT
3.GTE
4.LT
5.LTE
6.S1⊕S2T
7.S1⊕S2F
0.GP2
1.LRR
2.MVB
3.MMO
0.GP1
1.GP2
2.GP3
3.LRR
4.+1
5.1'S
6.PIMMED
7.0
0.AND
1.OR
2.EXOR
3.+
4.–
5.*
6./
7.SI IF CO ELSE S2
8.BSET⊕S2
9.BCLR⊕S2
A.BSUB AT S2 W/ CO
B.ROTATE/MERGE 8
C.CHKSUM
D.ENCR/DECR
E.DYNAMIC KEY
F.PARITY
0.GP1
1.GP2
2.GP3
3.LRR
4.CP
5.MVBUS
6.PC+1
7.RAW1N
0.NOΔ
1.MMO
0.NOΔ
1.MMO
2.–1
3.MVB
0.NOΔ
1.MMO
2.+1
3.LRR
38
39..38
37
36..34
33..31
30..28
27..26
25
24
23..20
19
0
INST TYPE
JMP TGT
MMU S2
MMU OP
MMU S1
GP3
GP2
GP1
MOVE OPT
LIMMED
LIM
1 0
0.PC+1
1.MMO
0.GP1
1.GP2
2.GP3
3.LINTO
4.LINT1
5.LFRCO
6.LFRC1
7.0
0.AND
1.OR
2.EXOR
3.+
4.–
5.*
6.R/M40
7.S2
0.GP1
1.GP2
2.GP3
3.LINTO
4.CP
5.MVBUS
6.PC+1
7.RAWIN
0.NOΔ
1.MMO
2.PC+1
3.MVB
0.NOΔ
1.MMO
0.NOΔ
1.MMO
40
A

Fig–14A

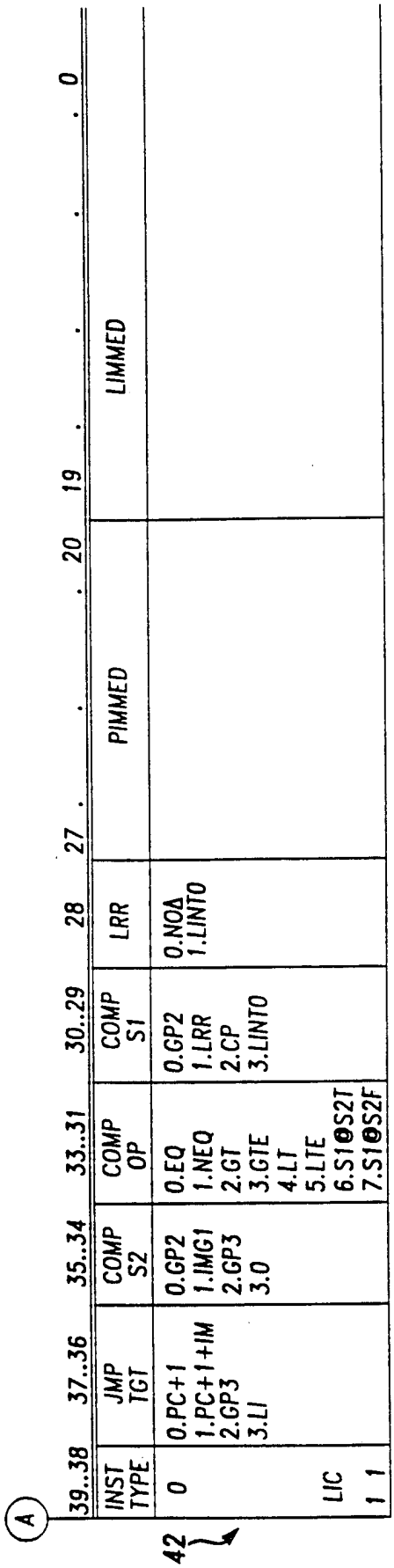

A
42
39..38
37..36
35..34
33..31
30..29
28
27
20
19
0
INST TYPE
JMP TGT
COMP S2
COMP OP
COMP S1
LRR
PIMMED
LIMMED
0
0.PC+1
1.PC+1+IM
2.GP3
3.LI
0.GP2
1.IMG1
2.GP3
3.0
0.EQ
1.NEQ
2.GT
3.GTE
4.LT
5.LTE
6.S1@S2T
7.S1@S2F
0.GP2
1.LRR
2.CP
3.LINTO
0.NOΔ
1.LINTO
LIC
1 1

Fig-14B

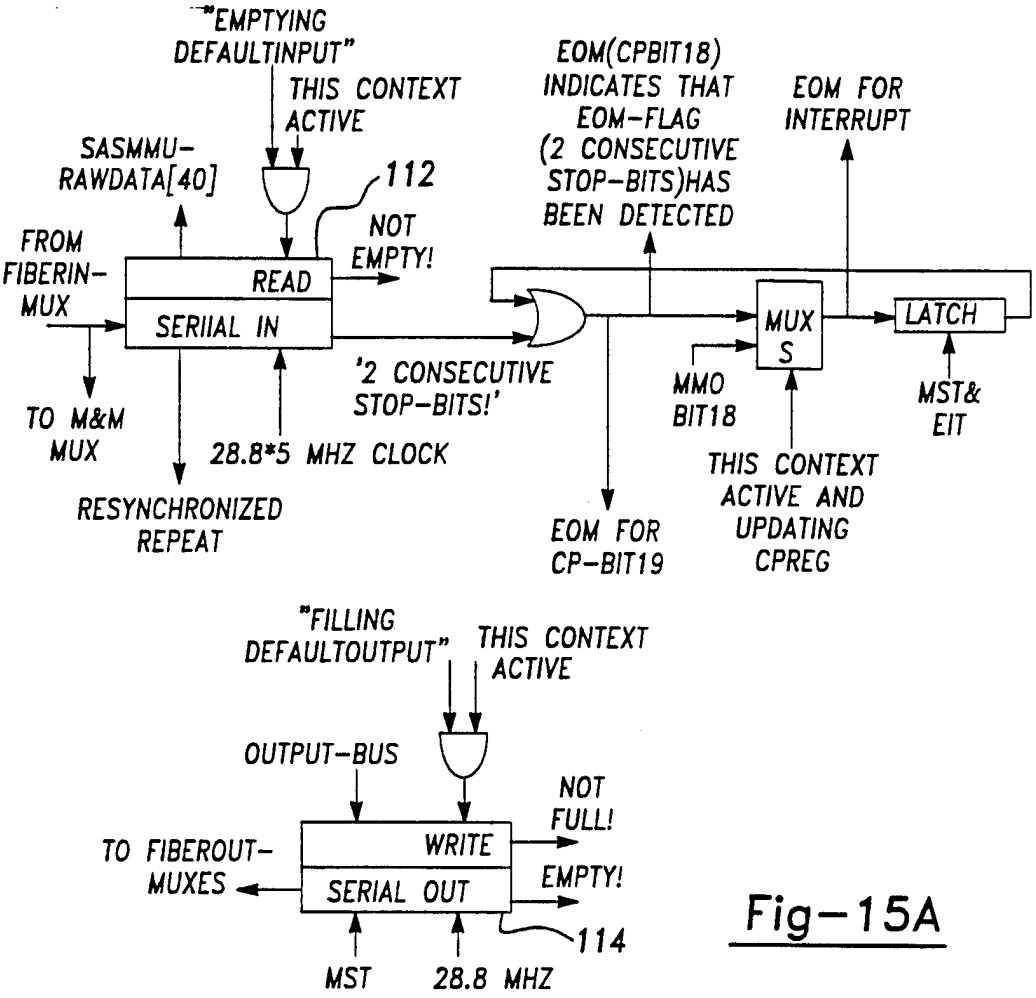


Fig-15A

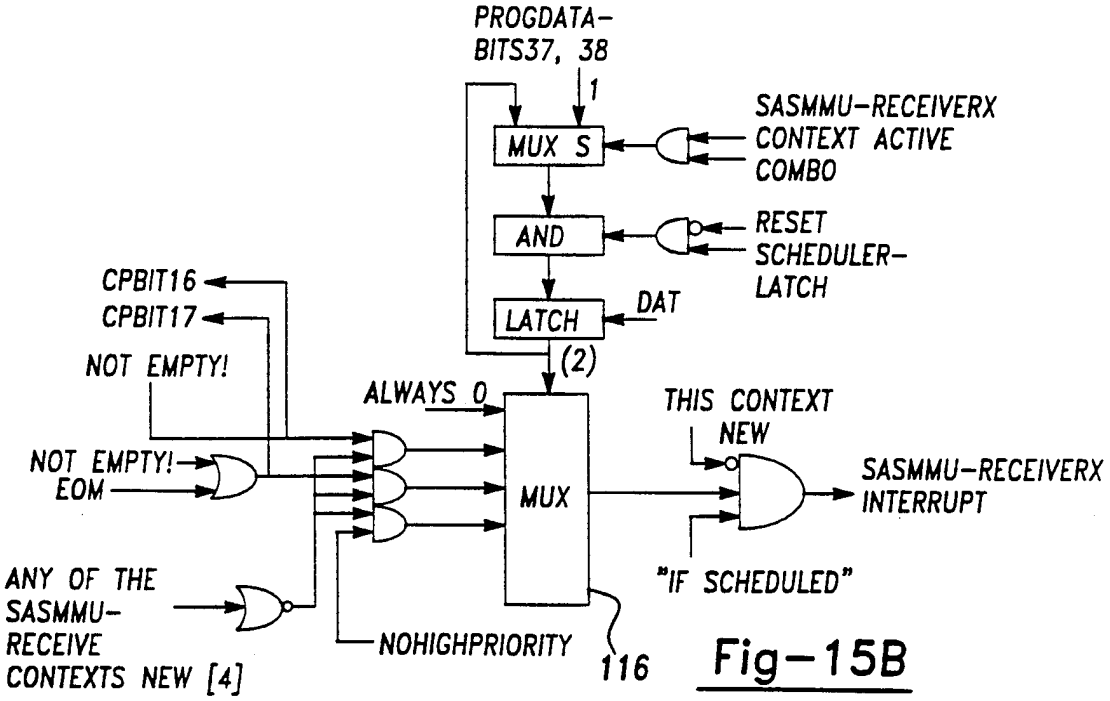


Fig-15B

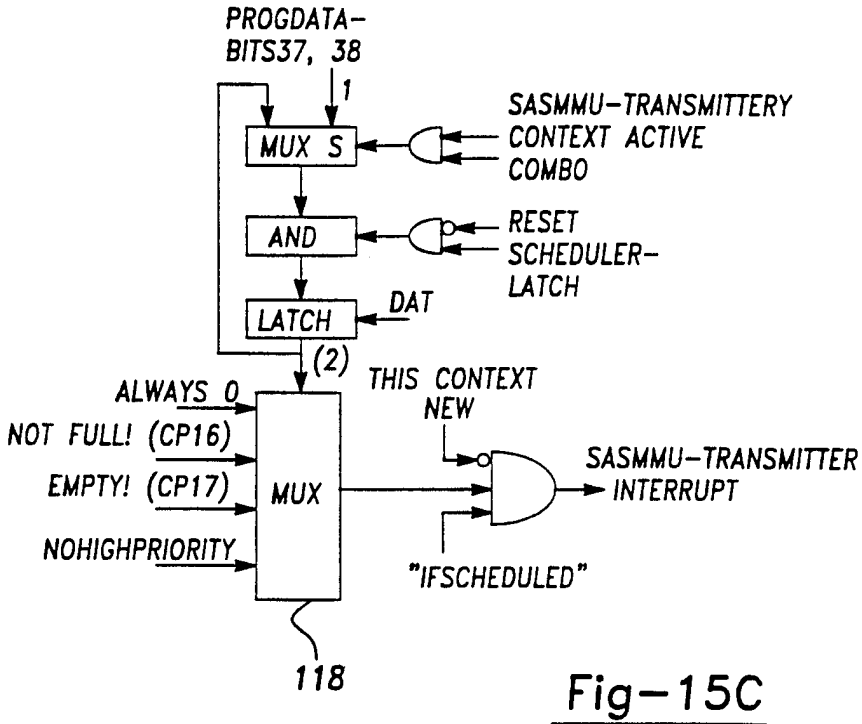


Fig-15C

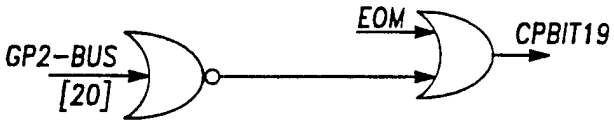


Fig-15E

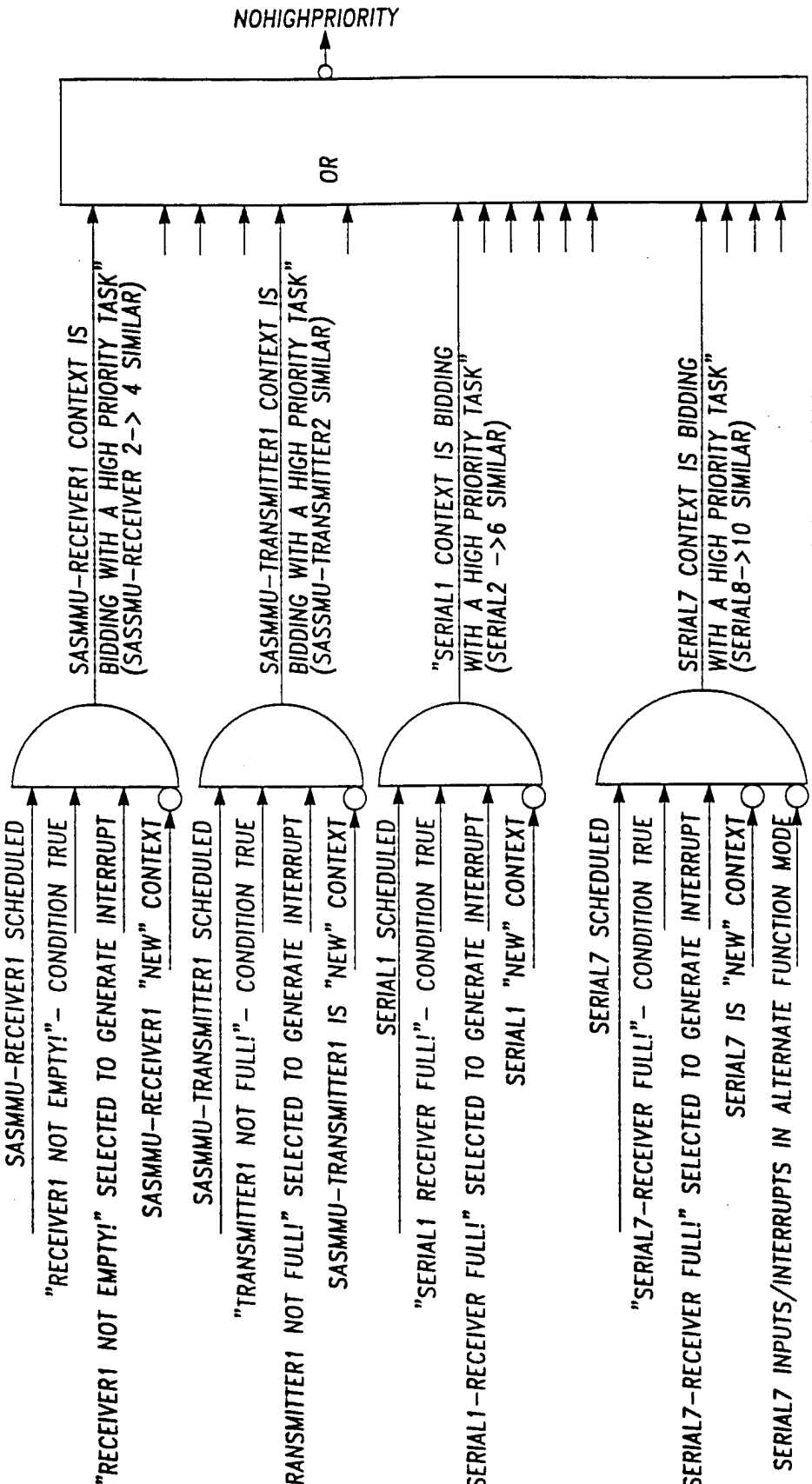

NOHIGHPRIORITY
OR
SASMMU-RECEIVER1 CONTEXT IS BIDDING WITH A HIGH PRIORITY TASK" (SASSMU-RECEIVER 2-> 4 SIMILAR)
SASMMU-TRANSMITTER1 CONTEXT IS BIDDING WITH A HIGH PRIORITY TASK" (SASSMU-TRANSMITTER2 SIMILAR)
"SERIAL1 CONTEXT IS BIDDING WITH A HIGH PRIORITY TASK" (SERIAL2 ->6 SIMILAR)
SERIAL7 CONTEXT IS BIDDING WITH A HIGH PRIORITY TASK" (SERIAL8->10 SIMILAR)
SASMMU-RECEIVER1 SCHEDULED
"RECEIVER1 NOT EMPTY!"- CONDITION TRUE
"RECEIVER1 NOT EMPTY!" SELECTED TO GENERATE INTERRUPT
SASMMU-RECEIVER1 "NEW" CONTEXT
SASMMU-TRANSMITTER1 SCHEDULED
"TRANSMITTER1 NOT FULL!"- CONDITION TRUE
"TRANSMITTER1 NOT FULL!" SELECTED TO GENERATE INTERRUPT
SASMMU-TRANSMITTER1 IS "NEW" CONTEXT
SERIAL1 SCHEDULED
"SERIAL1 RECEIVER FULL!"- CONDITION TRUE
"SERIAL1-RECEIVER FULL!" SELECTED TO GENERATE INTERRUPT
SERIAL1 "NEW" CONTEXT
SERIAL7 SCHEDULED
"SERIAL7-RECEIVER FULL!"- CONDITION TRUE
"SERIAL7-RECEIVER FULL!" SELECTED TO GENERATE INTERRUPT
SERIAL7 IS "NEW" CONTEXT
SERIAL7 INPUTS/INTERRUPTS IN ALTERNATE FUNCTION MODE

Fig-15D

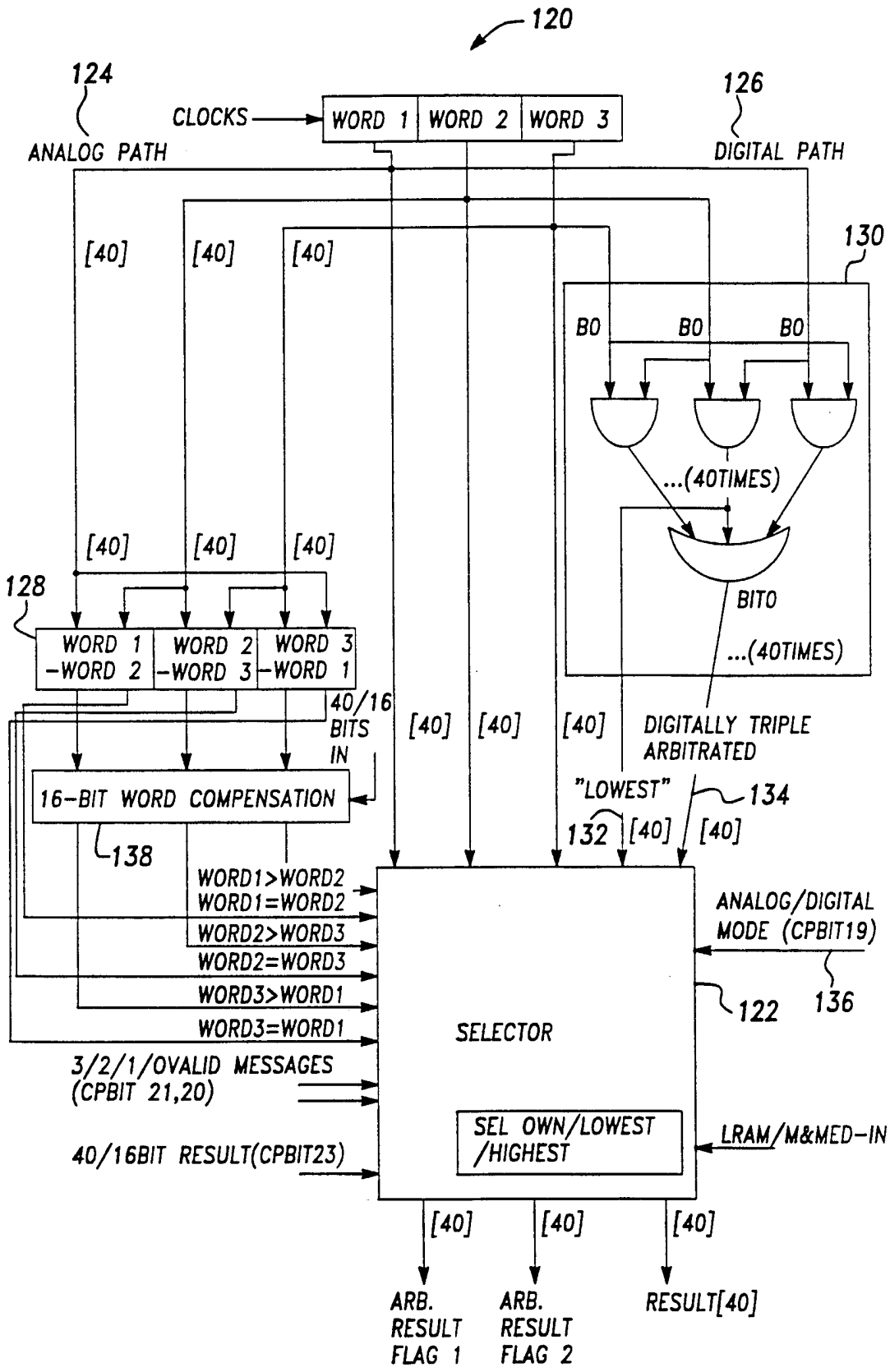

120
124
ANALOG PATH
CLOCKS
WORD 1
WORD 2
WORD 3
126
DIGITAL PATH
[40]
[40]
[40]
130
B0
B0
B0
...(40TIMES)
BIT0
...(40TIMES)
128
WORD 1 -WORD 2
WORD 2 -WORD 3
WORD 3 -WORD 1
40/16 BITS IN
16-BIT WORD COMPENSATION
138
DIGITALLY TRIPLE ARBITRATED
"LOWEST"
132
134
WORD1>WORD2
WORD1=WORD2
WORD2>WORD3
WORD2=WORD3
WORD3>WORD1
WORD3=WORD1
3/2/1/OVALID MESSAGES (CPBIT 21,20)
40/16BIT RESULT(CPBIT23)
SELECTOR
SEL OWN/LOWEST /HIGHEST
ANALOG/DIGITAL MODE (CPBIT19)
136
122
LRAM/M&MED-IN
ARB. RESULT FLAG 1
ARB. RESULT FLAG 2
RESULT[40]

Fig-16

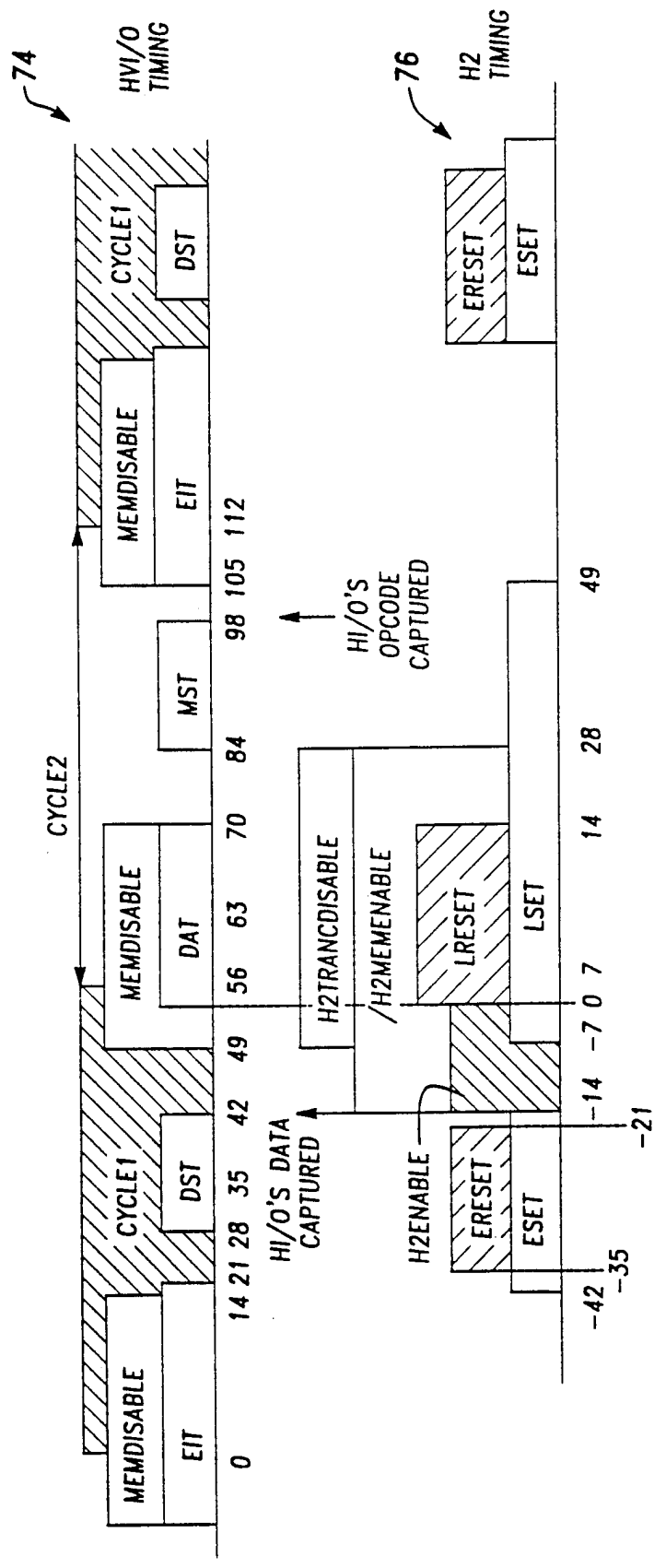

74
HVI/O TIMING
76
H2 TIMING
CYCLE1
DST
MEMDISABLE
EIT
CYCLE2
MST
DAT
MEMDISABLE
CYCLE1
DST
MEMDISABLE
EIT
ERESET
ESET
LRESET
LSET
ERESET
ESET
H2TRANCDISABLE
/H2MEMENABLE
HI/O'S OPCODE CAPTURED
HI/O'S DATA CAPTURED
H2ENABLE
0
14 21 28 35 42 49 56 63 70 84 98 105 112
49 28 14 0 7 -7 -14 -21 -35 -42

Fig-17

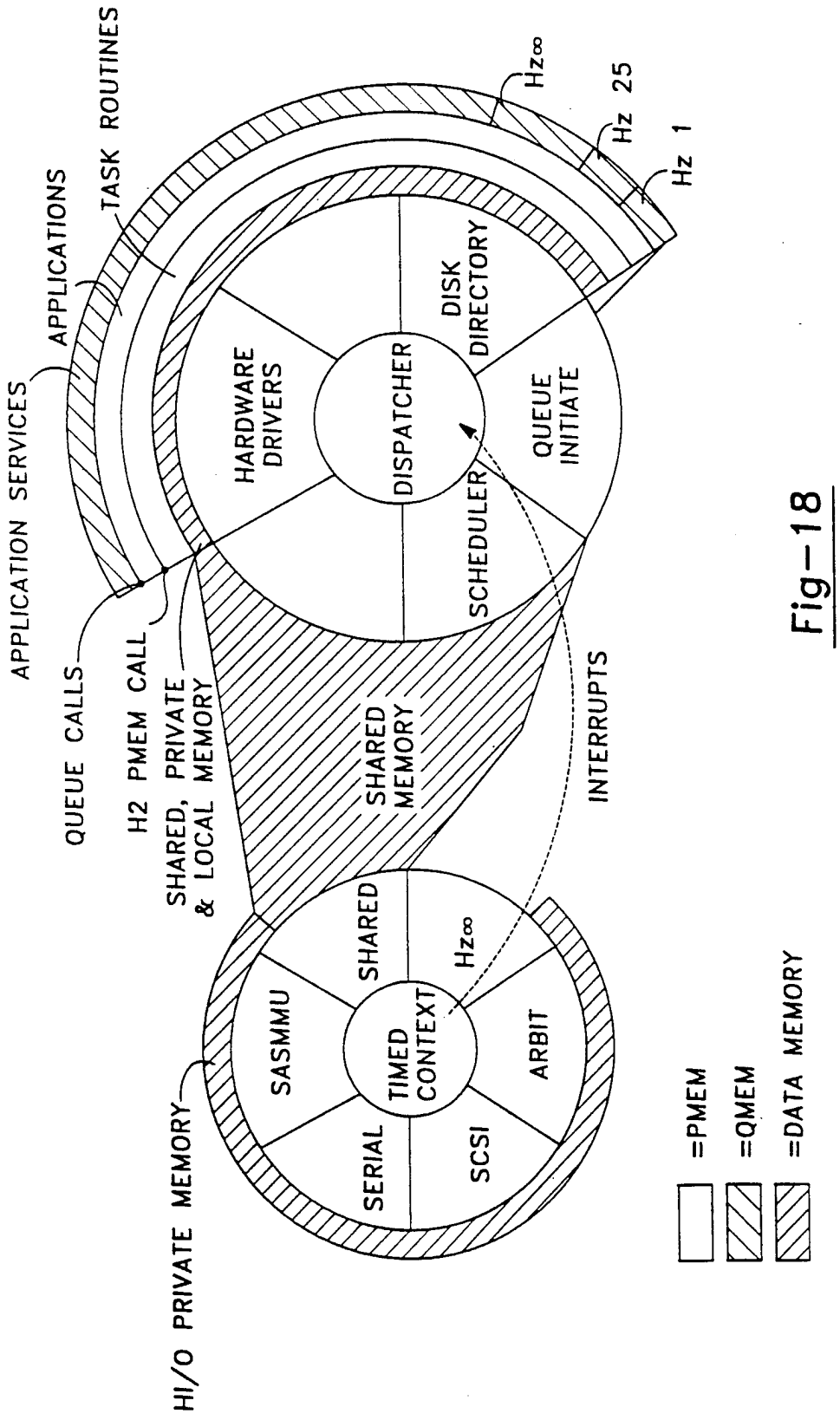

APPLICATION SERVICES
APPLICATIONS
TASK ROUTINES
QUEUE CALLS
H2 PMEM CALL
SHARED, PRIVATE & LOCAL MEMORY
HI/O PRIVATE MEMORY
SHARED MEMORY
INTERRUPTS
HARDWARE DRIVERS
DISPATCHER
DISK DIRECTORY
QUEUE INITIATE
SCHEDULER
Hz∞
Hz 25
Hz 1
SASMMU
SHARED
TIMED CONTEXT
ARBIT
SERIAL
SCSI
=PMEM
=QMEM
=DATA MEMORY

Fig-18

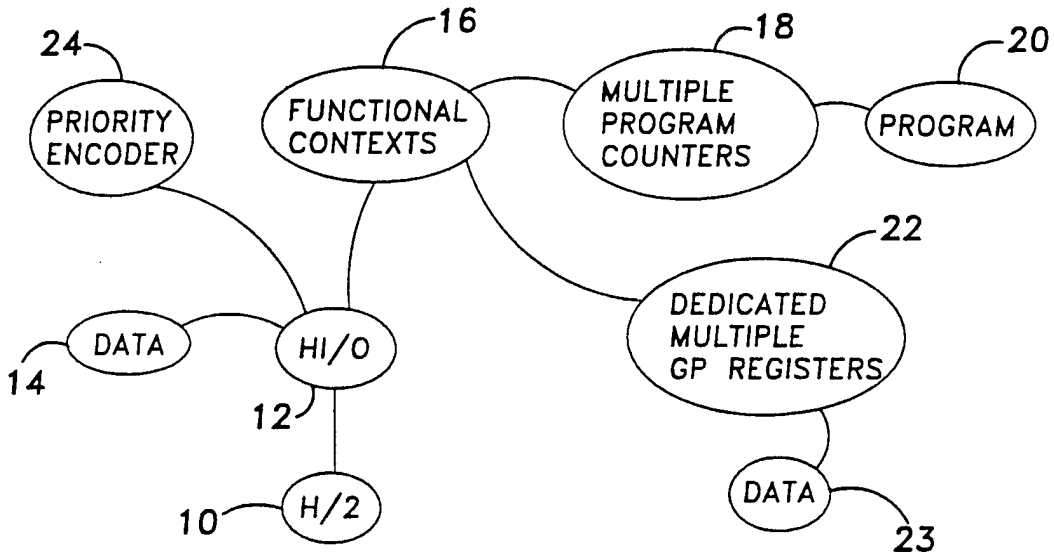

24
PRIORITY ENCODER
16
FUNCTIONAL CONTEXTS
18
MULTIPLE PROGRAM COUNTERS
20
PROGRAM
22
DEDICATED MULTIPLE GP REGISTERS
DATA
14
HI/O
12
H/2
10
DATA
23

Fig-19

INTERNATIONAL SEARCH REPORT

Inte. onal Application No
PCT/US 98/02016

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46

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 6 G06F

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)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	DE 43 01 117 A (SIEMENS AG) 5 August 1993 see abstract see column 2, line 22 - column 3, line 27 ---	1-3,5-9, 18,25,26
A	US 5 291 614 A (BAKER ROBERT G ET AL) 1 March 1994 see column 2, line 20 - line 68 see column 9, line 46 - column 13, line 35 --- -/--	1-3,5,7, 8,11-15, 18-26

[X] Further documents are listed in the continuation of box C.

[X] 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.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 May 1998	05/06/1998

Name and mailing address of the ISA	Authorized officer
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	Fournier, C

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inter onal Application No
PCT/US 98/02016

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	G. ANZINGER: "Managing PA-RISC Machines for Real-Time Systems" HEWLETT-PACKARD JOURNAL., vol. 44, no. 4, August 1993, PALO ALTO US, pages 31-37, XP000392979 see page 31, left-hand column, line 40 - right-hand column, line 26 ---	1,2,5, 11,13, 18,25,26
A	US 4 197 579 A (FORSMAN PETER E ET AL) 8 April 1980 see abstract ---	1,2,5,9, 18-20, 25,26
A	EP 0 617 361 A (DIGITAL EQUIPMENT CORP) 28 September 1994 see the whole document -----	1,2,5, 18,25,26

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter. nal Application No
PCT/US 98/02016

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
DE 4301117 A	05-08-1993	NONE	
US 5291614 A	01-03-1994	CA 2074633 A	04-03-1993
		JP 2514296 B	10-07-1996
		JP 5274276 A	22-10-1993
US 4197579 A	08-04-1980	NONE	
EP 0617361 A	28-09-1994	NONE	