PARALLEL DISTRIBUTED PROCESSING OF REALTIME TELEMETRY DATA

Donald P. Murphy
Syndetix, Inc.
2801 East Missouri Ave., Suite 36
Las Cruces, New Mexico 88001

ABSTRACT

An architecture is described for Processing Multiple digital PCM telemetry streams. This architecture is implemented using a collection of Motorola mono-board microprocessor units (MPUs) in a single chassis called an Intermediate Processing Unit (IPU). Multiple IPUs can be integrated using a common input data bus. Each IPU is capable of processing a single PCM digital telemetry stream. Processing, in this context, includes conversion of raw sample count data to engineering units; computation of derived quantities from measurement sample data; calculation of minimum, maximum, average and cyclic [(maximum - minimum)/2] values for both measurement and derived data over a pre-selected time interval; out-of-limit, dropout and wildpoint detection; strip chart recording of selected data; transmission of both measurement and derived data to a high-speed, large-capacity disk storage subsystem; and transmission of compressed data to the host computer for realtime processing and display. All processing is done in realtime with at most two PCM major frames time latency.

Keywords: realtime telemetry data acquisition, distributed processing, parallel computation.

INTRODUCTION

In the course of developing general-purpose telemetry data acquisition systems, one is eventually confronted with the realization that considerable sacrifices must be made regarding the quantity and timeliness of data available for realtime decision-making and analysis. This is due, in part, to the sharpness of the boundary between telemetry front-end equipment (TFE) and the host realtime computer systems. Typical TFE-to-host data paths are rigid, direct hardware connections; all flexibility must be programmed into the host. Most realtime host computer systems are required to perform other functions in addition to data acquisition and therefore tend to be less specialized than in the past. As a result, they

are sometimes inadequate at very high-speed data acquisition or at performing highly repetitive computations in conjunction with other realtime processing.

In a different vein, it is my contention that realtime software engineers prefer to design code that can operate on a variety of data structures which are in some sense natural with respect to processing requirements and thereby independent of the particular idiosyncrasies of telemetry stream structures. I also contend that it is highly desirable to remove from the host the complex operations inherent in preparing telemetry data for analysis, and the intricacies involved in mapping individual telemetry stream characteristics to the common data structures required for efficient realtime manipulation.

Toward these ends, we have designed, developed, and implemented a device referred to as an Intermediate Processing Unit (IPU). This device is a collection of parallel, distributed microprocessors positioned between the TFE and the host computer system. The salient features of such a device are

1) isolation and concentration of telemetry stream computations
2) reduction of the load on the host computer system
3) programmability, flexibility, and modularity
4) mechanisms for data routing
5) performance of repetitive functions not usually amenable to a general-purpose host computer system
6) comparability with other system elements
7) comparability with system design philosophy

The balance of this paper will be concerned with a presentation of the architecture (both hardware and software) of the IPU.

SPECIFIC REQUIREMENTS

The particular IPU design being presented was created for the Fixed Base Data System for the McDonnell Douglas Helicopter Company. As such, it is required to perform the following functions:

Engineering Unit Conversion. All analog telemetry count data (up to 16 bits/word) are converted to 32-bit floating point engineering units compatible with the host data format. This conversion is done using table look-up for 10-bit/word or less count data and a mixture of table look-up and calculation for count data greater than 10-bit/word, depending on allocated memory. All binary coded decimal (BCD) data is also converted. Certain data (e.g., discrete and time) can be passed through unaltered.

Derived Calculation. A collection of additional data is to be computed from input telemetry data. Virtually any algorithm can be implemented for such computation. Typically, multi-linear, trigonometric, exponential, and logarithmic functions are used.

Data Compression. Telemetry measurement and derived data are to be averaged over a user-defined time interval. These data are usually those which are presented to the host computer system for monitoring and display. In addition, the minimum, maximum, and cyclic value [i.e., (maximum - minimum)/2] are computed for each measurement and derived quantity over this interval. It is these compressed data which are used principally for realtime decision making.

Varied Formats. The IPU must be capable of processing a variety of different PCM telemetry streams structures. These include different sub-commutation, sub-sub-commutation, and super-commutation strategies, different minor and major frame sizes, and different telemetry word formats.

Data Routing. The IPU is capable of producing various data products. As such, these products can be transmitted to different components of the realtime system. Typically, converted sample data (both measurement and derived) are routed to a fast-disk subsystem, compressed data (minimum, maximum, cyclic, and average over a specified time interval) to the host, and selected measurement and derived data to digital-to-analog converters (DACs) for recording on strip charts and monitoring on event displays.

Dropout Detection. Since the host is one step removed from the TFE, telemetry signal dropout it detected by the IPU and this information passed to the host.

Wildpointing. Any number of wildpoint algorithms can be implemented. Typically, a forward or central difference is used.

Limit Detection. Since the primary function of the system of which the IPU is a part is flight test, safety issues are paramount. To assist in this function, the IPU can detect limit threshold violations of measurement and derived data. Any violations are signaled to the host via status information transmitted as part of the compressed data.

Programmability. The IPU is highly reprogrammable because the system of which it is a part is designed for flexibility. In this broader context, the IPU data and program structures originate from a complex database specifically constructed to allow the rapid generation of products bound to the details of individual test conditions.

Phase Alignment. Because of the sampling delays inherent in many telemetry systems, it is necessary for the IPU to maintain data and program structures to ensure that all sample (and corresponding derived quantities) are aligned correctly in time.

Multiple Streams. An IPU is designed to process a single telemetry stream. Therefore, by inserting several IPUs between the TFE and the host, multiple streams can be processed.

Although capable of performing a wide variety of tasks, the IPU described herein is designed with the above requirements in mind.

FUNCTIONAL DESCRIPTION

The IPU is composed of seven functional components (See Figure 1): 1) a Global Input Processor (GIP) and its corresponding memory, the Global Input Memory (GIM); 2) a TFE-to-GIP interface; 3) several Data Stream Processors (DSPs); 4) a DAC Output Processor (DOP); 5) a Global Output Processor (GOP) and its corresponding memory, the Global Output Memory (GOM); 6) a GOP-to-disk interface; and 7) a GOP-to-host interface. Each of these is discussed in turn.

Global Input Processor (GIP). The GIP is responsible for obtaining the digital telemetry data from the TFE. In its operation it is much akin to a video frame grabber; i.e, it assembles a complete PCM telemetry major frame before signaling the remainder of the IPU components. Using this strategy, the GIP can properly construct a phase-aligned PCM major frame, ensure completeness of the data, and ensure correct data order. By detecting dropout in the input stream (done with internal timers), the GIP can signal the other IPU components so that they can perform their necessary initialization functions. In addition, the GIP is connected directly to the host via an RS-232 interface so that it can respond directly to user-initiated commands and pass them, if necessary, to other IPU components. Among these are start/stop data acquisition, inhibit sample data transmission (i.e., transmit compressed data only), and reset. (See Figure 2)

TFE-GIP Interface. This is a custom interface between the GIP private bus and the TFE bus. It accepts 16 bits of tag and 32 bits of data from the TFE bus and translates the tag into a 16-bit address using an on-board 64K RAM. This address is combined with a programmed 16-bit base address register to form a 32-bit address and is presented along with the 32-bit data in GIP address space. The GIP uses the address to vector to the subprogram responsible for processing the data.

Data Stream Processors (DSPs). The DSPs perform the hulk of the work in the IPU. Each is programmed to process a selected set of measurements and deriveds independently

of the others, so that they operate in parallel. This processing includes: 1) all engineering unit conversion; 2) wild point detection; 3) derived Parameter computation; 4) compressed data computation; and 5) limit threshold detection. (See Figure 3)

DAC Output Processor (DOP). The DOP is responsible for transferring selected measurements and derived sample data to strip chart recording devices. This component can be independently controlled from the host via its own RS-232 interface, thereby allowing realtime changes to be made to the DAC/strip chart assignments. (See Figure 4)

Global Output Processor (GOP). The GOP performs two essential functions. First, it is responsible for data routing--EU-converted sample data is transferred to a fast-disk subsystem and compressed data is transferred to the host. Second, it acts as the boot loader for the entire IPU. In its reset state, it expects to receive a collection of load modules for the other IPU components from the host. (See Figure 5)

GOP-Host Interface. This is a standard DMA interface between GOP memory and host memory. It is bidirectional, used both to download software from the host and to transmit compressed data to the host.

GOP-Disk Interface. The GOP-to-disk interface is a custom interface card between the GOP private bus and the disk controller (which is actually an I/O computer--see below) private bus. This interface is not DMA (since the GOP private bus does not support this), so transfers through this interface must be performed under program control.

SOFTWARE ARCHITECTURE

Software for all IPU components is constructed from a collection of library subprograms on a per-test basis. This allows for the flexibility, maintainability, and modifiability of all IPU code. There are three primary software categories for each IPU component. First, each has a initial program loader in onboard ROM. This is used to download other code from the host. Second, each loadable code module has a high-level executive routine which is responsible for supervising the execution of the test specific code. This routine is independent of test-specific details; therefore, it is the same for all tests. The remainder of each loadable code module consists of test-specific code and data structures operating in the environment established by the executive.

GIP Software. GIP code it essentially a collection of small subprograms bound together via a vector table. The executive is responsible for loading the GIP-to-TFE interface RAM with measurement-tag/subprogram-address pairs (obtained from a data packet bound at load time). This defines the measurement tag-to-subprogram binding. After loading, the executive turns control over to the major frame synchronization subprogram. Subsequent

executive activity is limited to interrupt service when responding to host commands (via the RS-232 interface) or dropout (via internal minor and major frame timers).

DOP Software. The DOP code consists of an executive together with collections of tables used for translating measurement and derived sample data to DAC/strip chart count data. In addition, there are data structures which map measurements and derived quantities to DAC channel and strip chart number. These mapping structures can be modified in realtime by host commands to the DOP executive via an RS-232 interface. The DOP-to-DAC interface is identical to the TFE-to-GIP interface since the DAC channels are connected to a second TFE bus having the same architecture as the input telemetry bus.

DSP Software. DSP code consists of an executive routine together with a collection of data structures and threaded code defined by the specific computational requirements of each DSP. The data structures consist of a linked list of computational packets, each of which contains a 32-byte header followed by a set of entries. The packet header contains a vector table whose entries point to the processing subprograms associated with the packet. These include the packet initialization, computation, and compression subprograms. All entries in a packet must be capable of being processed by the same code. Thus, measurements and deriveds of like type are collected together in the same packet. It is important that the processing subprograms be efficient so that the MC68020 instruction cache is utilized to the maximum possible extent.

When the DSP executive receives a “major frame ready” interrupt from the GIP, it determines from the common data area which processing is required. These are initialize and compute, compute only, and compute and compress. It then proceeds through the linked packet list, vectoring through each packet header to the appropriate code sets. When the end of the list is reached, the executive returns to its dormant state, waiting for the next interrupt.

From this discussion, it should be clear that whatever a DSP does must be accomplished in one major frame cycle. Therefore, load balancing the various DSPs is a critical issue. This is done a priori by host software for each test.

GOP Software. The GOP code is very similar to that of the GIP except that it is less sensitive to changes in telemetry stream structure. The primary function of the GOP executive is to determine, using the common data area, when it is necessary to initiate transfer of the sample data buffers to the fast-disk subsystem and compressed data to the host. The GOP executive must also collect status information (DSP and telemetry stream) and pass this to the host and fast-disk subsystem controller.

THEORY OF OPERATION

The operational details of the IPU are relatively easy to understand. Each component executes independently from and in parallel with the others. They coordinate their activities by using a global interrupt and a common data area (in the GOM, since it is visible from all processors). The GIP assembles telemetry major frames in GIM via the TFE-to-GIP interface using its private bus. After each is successfully assembled, the GIP generates the global interrupt to signal other components and proceeds with the next frame. During this Process, the GIP is also monitoring the count of major frames in the compression interval. When this interval expires, the GIP indicates this event in the common data area. Telemetry stream dropout is also communicated to other IPU components via the common data area.

When the global “frame ready” signal is generated, each DSP should be dormant. Upon interruption, they can proceed through four different paths, dependent upon the contents of the common data area. When the first major frame is ready after the initial load or after recovery from a dropout, the DSPs initialize all of the required items in the data structures and then execute their normal computation code. For successive major frames, they simply execute the computation code until the compression interval has expired, at which time they also execute the compression code. Dropout indication is ignored since initialization must eventually be performed when the telemetry is re-acquired. Finally, a reset may be indicated, in which case the DSPs branch into their initial program loader (IPL) located in onboard ROM. It should be noted that DSP code obtains sample count data from the GIM via the global IPU bus while the GIP is accessing the GIM via its private bus. This is critical for throughput considerations since it is important not to saturate the global bus.

As the DSPs execute their computation code, they are assembling in the GOM (via the global bus) an output buffer containing EU-converted sample data plus derived data. When the “frame ready” interrupt is received by the GOP, it interprets this to mean that the previous output buffer (i.e., the one filled by the DSPs in the previous frame cycle) is ready for transmission to the fast-disk subsystem, and initiates this activity. Should expiration of the compression interval also be indicated, the GOP initiates DMA transfer to the host. It is important here also to note that transmission of sample data to the fast disk is achieved using the GOP private bus, to reduce the global bus load. However, DMA transfer of compressed data cannot be done across the private bus since it has no DMA capability. A separate bus (having the same architecture as the global bus) with DMA capability is used instead.

From the above description, there is clearly a two major-frame latency through the IPU. The GIP is assembling the “current” one, DSPs are processing the previous one, and the GOP is transmitting the one previous to that.

IMPLEMENTATION

The above IPU architecture is implemented using Motorola MC68020/MC68881 mono-processing units (MPUs). The GIP, GOP, and DOP are 16.67 MHz MVME130 MPUs with the VMX private bus and VME global bus interface. The GIM, GOM, and DOM are MVME214 dual-ported (VMX/VME) 512KB static RAM. The DSPs are 16.67 MHz MVME133 MPUs with 1MB onboard DRAM and VME bus interface. The GOP-to-host interface is a VMIC VME-to-DEC DR11-W DMA interface card. The TFE is an Aydin Monitor System 2000 with two Mercury (Hg) buses. The TFE-to-GIP interface was designed and built by the Physical Science Laboratory (PSL) of New Mexico State University to interface the Hg and VMX buses. The fast-disk subsystem is an Ibis hosted by an Aptec 2400 I/O Computer. The GOP-to-fast disk interface was also designed and built by PSL to interface the VMX bus and the Aptec private bus. (See Figures 6 and 7)

PERFORMANCE BENCHMARKS

The following timing benchmark data has been determined.

1) Analog EU conversion (using table lookup), wildpointing, new maximum, new minimum, running sum and band edge (sample counts all zeros or all ones) detection takes from 10.5 to 16 microseconds per sample.

2) Derived computation, new maximum, new minimum, and running sum takes from 14 microseconds (linear equation) to 85 microseconds (modulus) per sample.

3) Compression (minimum, maximum, average, and cyclic) plus seven limit threshold detection takes 65 microseconds per measurement or derived.

4) Aggregate IPU throughput is from 2 - 4 MB per second.

These figures were obtained by using both simulated and live PCM telemetry data.

FUTURE ENHANCEMENTS

There are several enhancement areas for the IPU architecture. First, there are faster processors, co-processors, buses, and memories already available which are compatible with the existing software structure. With these products, it is possible to incorporate more intelligence into the IPU, particularly in the DSPs. Secondly, it would be desirable to develop an Ada-like structured language in which IPU software could be written (it is currently written in assembly language). This would be very beneficial for development and would fit in with other imbedded computer languages. Finally, there are several

VME-compatible storage devices available. Incorporating these into the IPU could enable implementation of other recording strategies.

ACKNOWLEDGEMENTS

The author would like to thank Jay Lory, Garen Saulsberry, Bruce Carter, Lori Kelly, and Troy Scoughton of the Physical Science Laboratory/New Mexico State University and Fritz Lawrence of Syndetix for their participation in this effort.

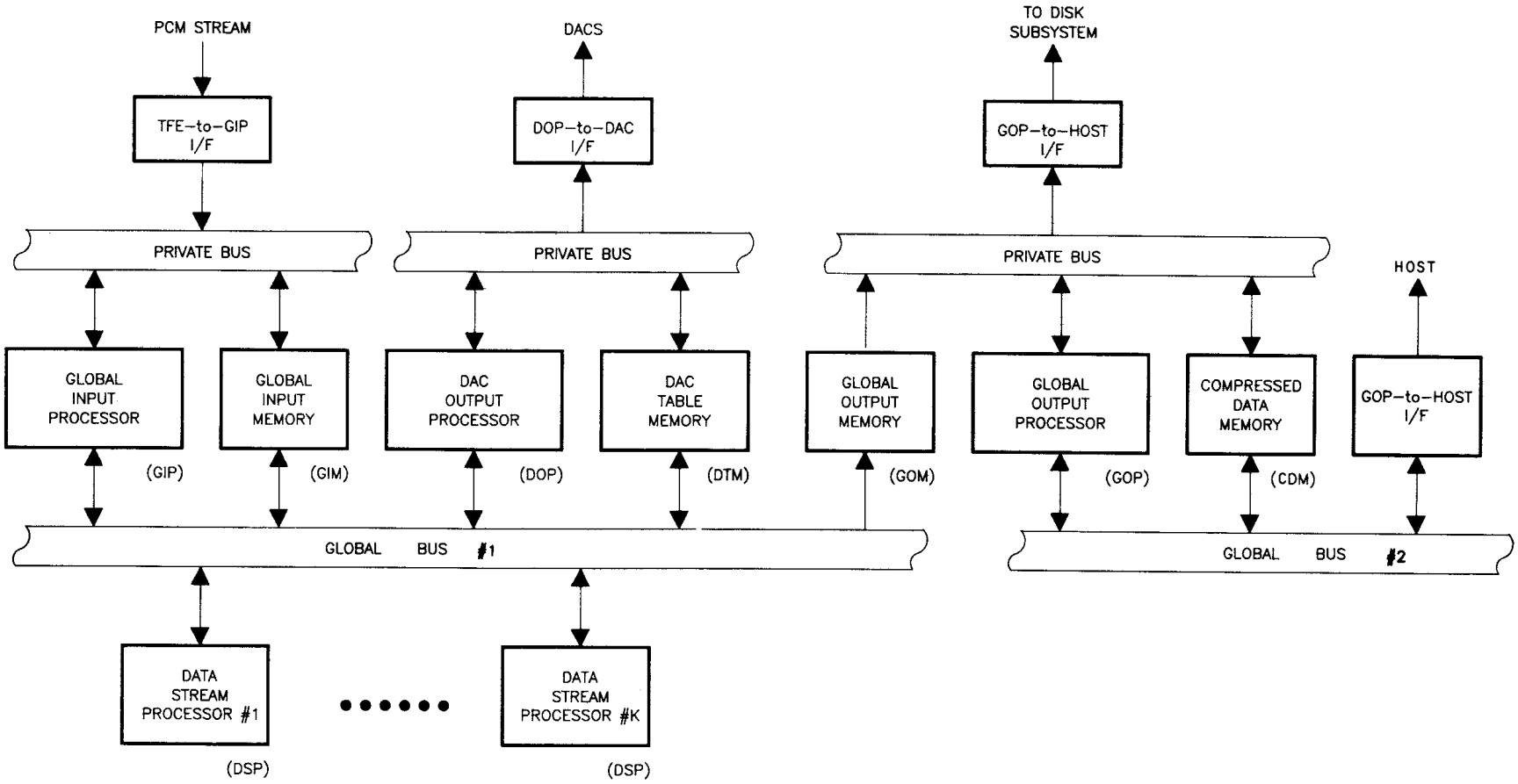

Figure 1. IPU Functional Components

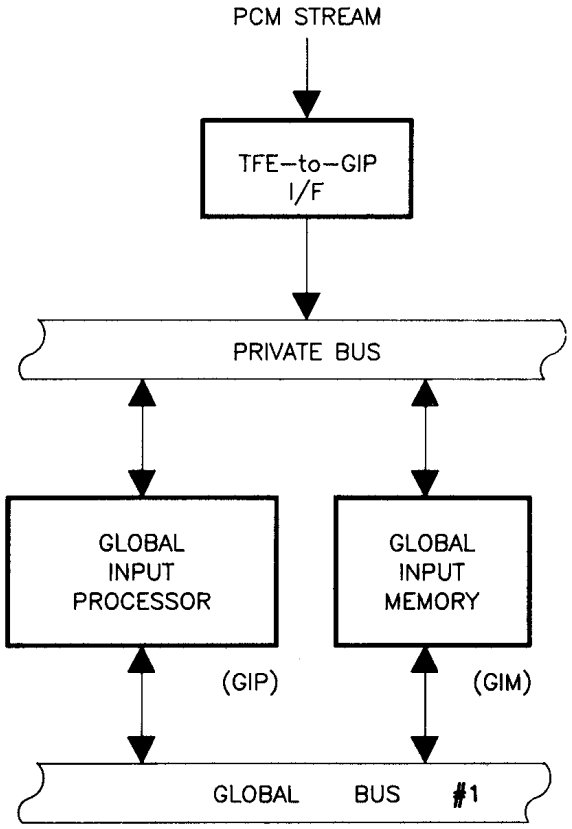

Figure 2. GIP Component

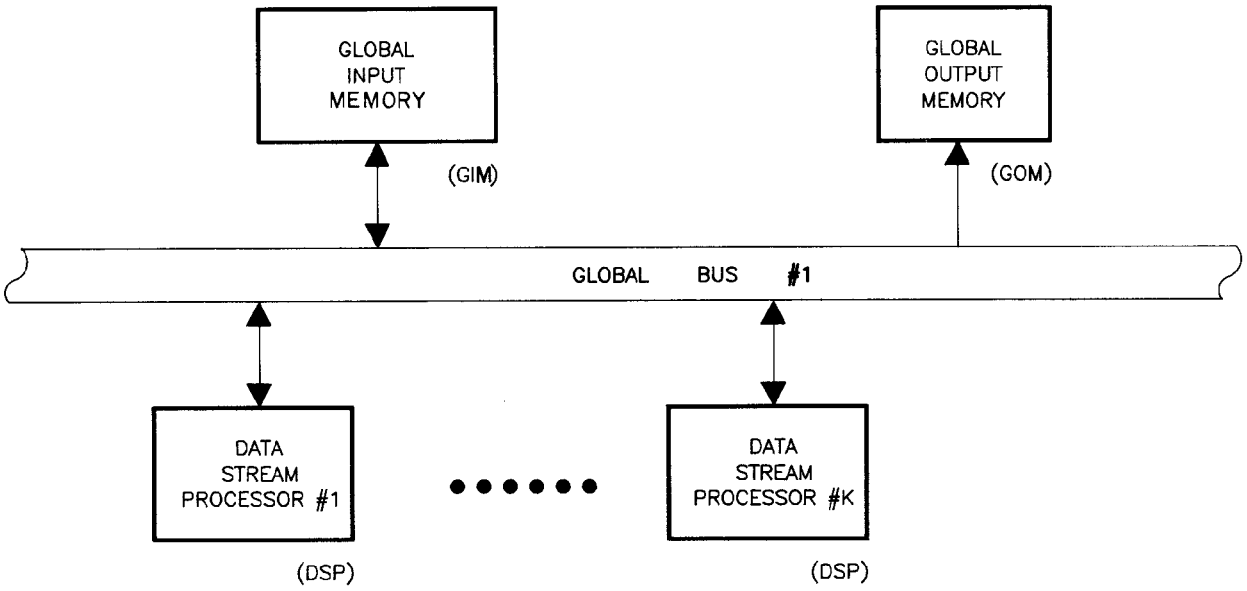

Figure 3. DSP Component

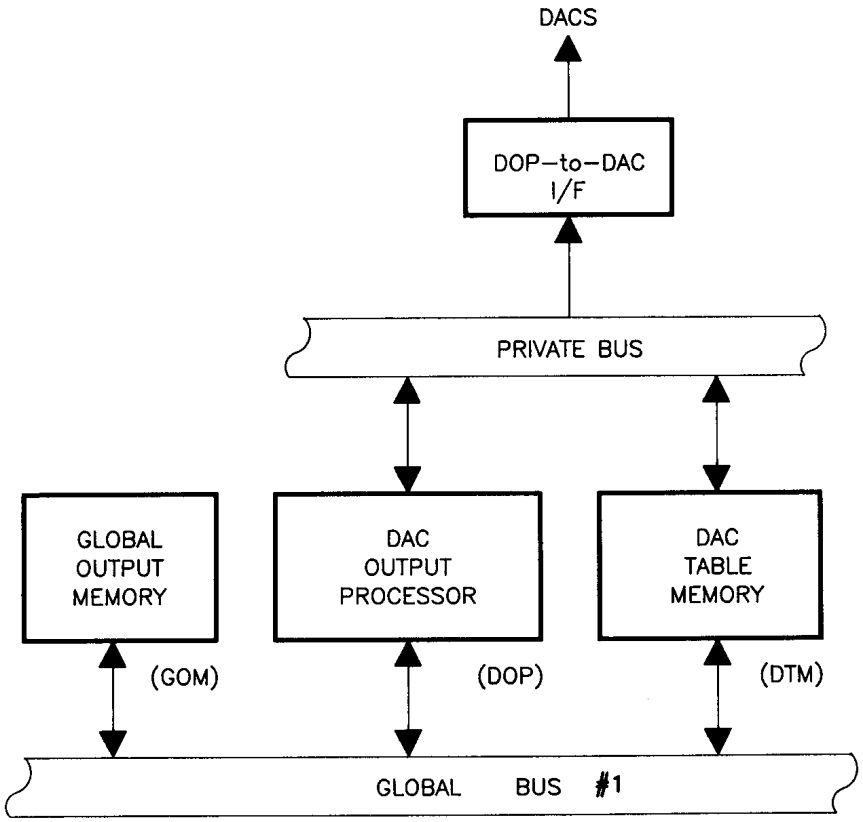

Figure 4. DOP Component

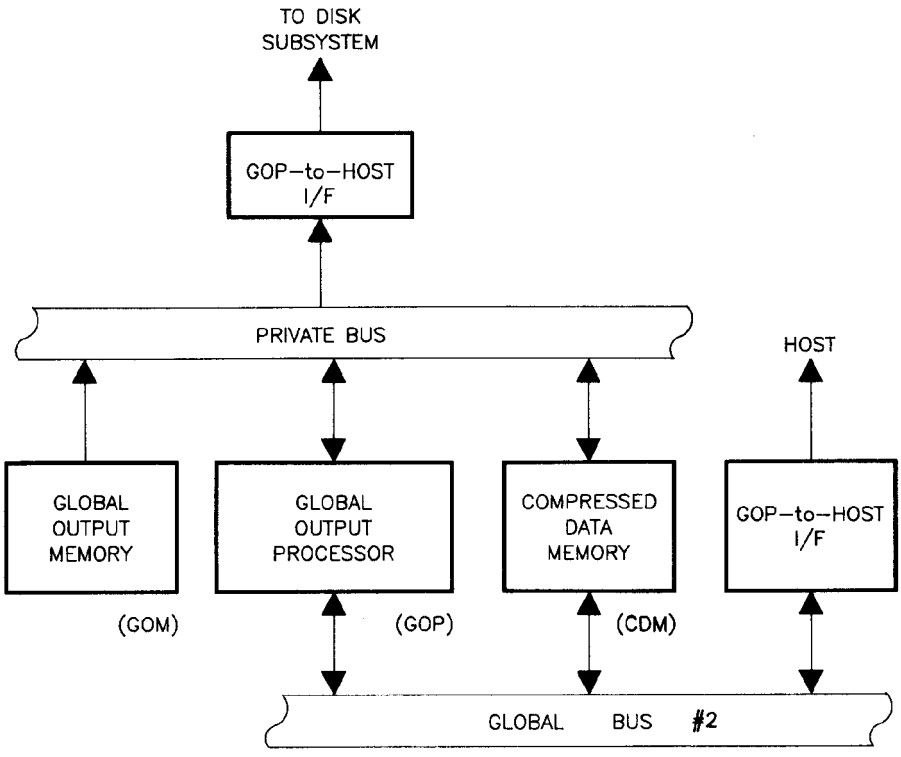

Figure 5. GOP Component

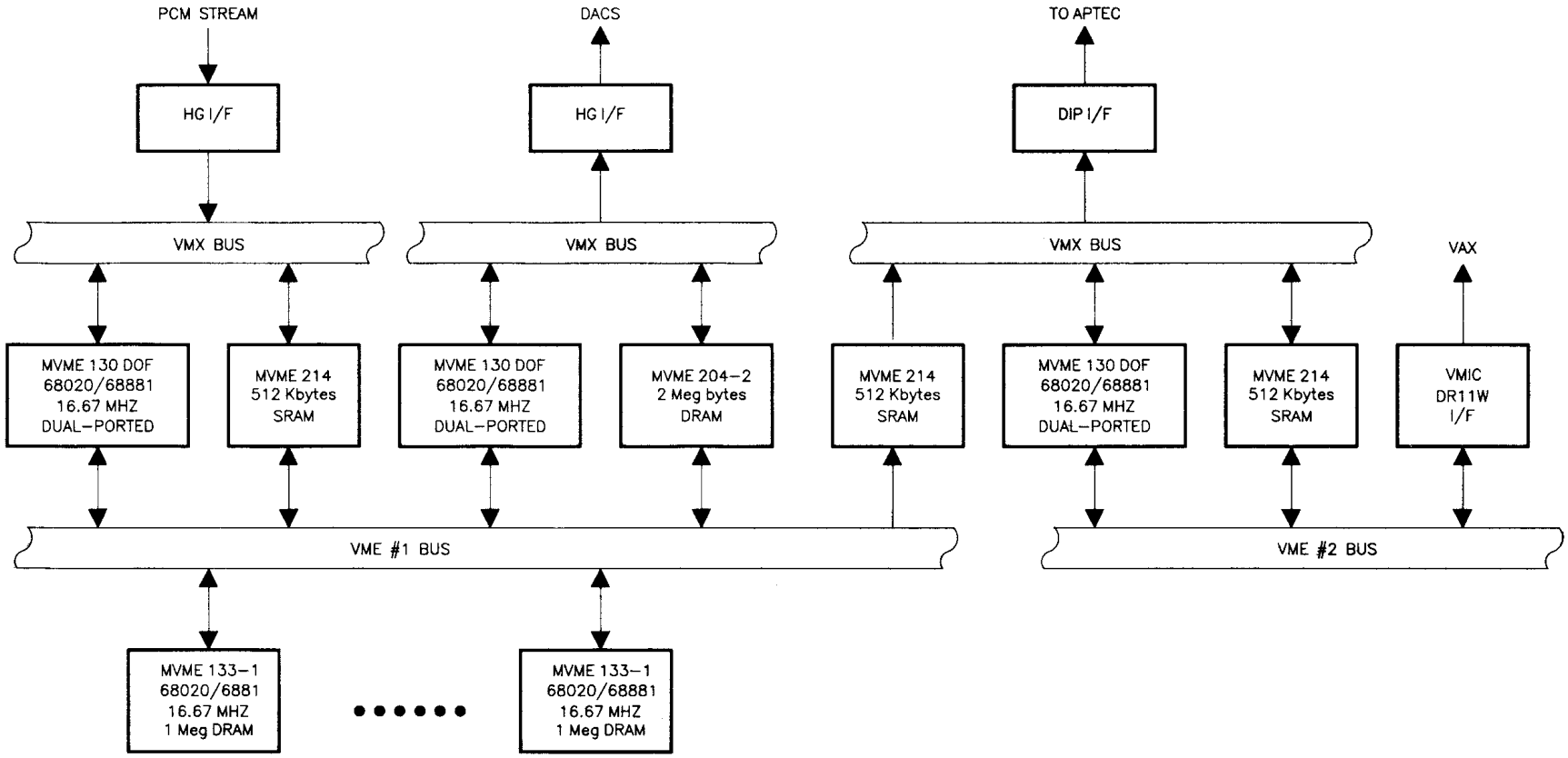

Figure 6. IPU Implementation

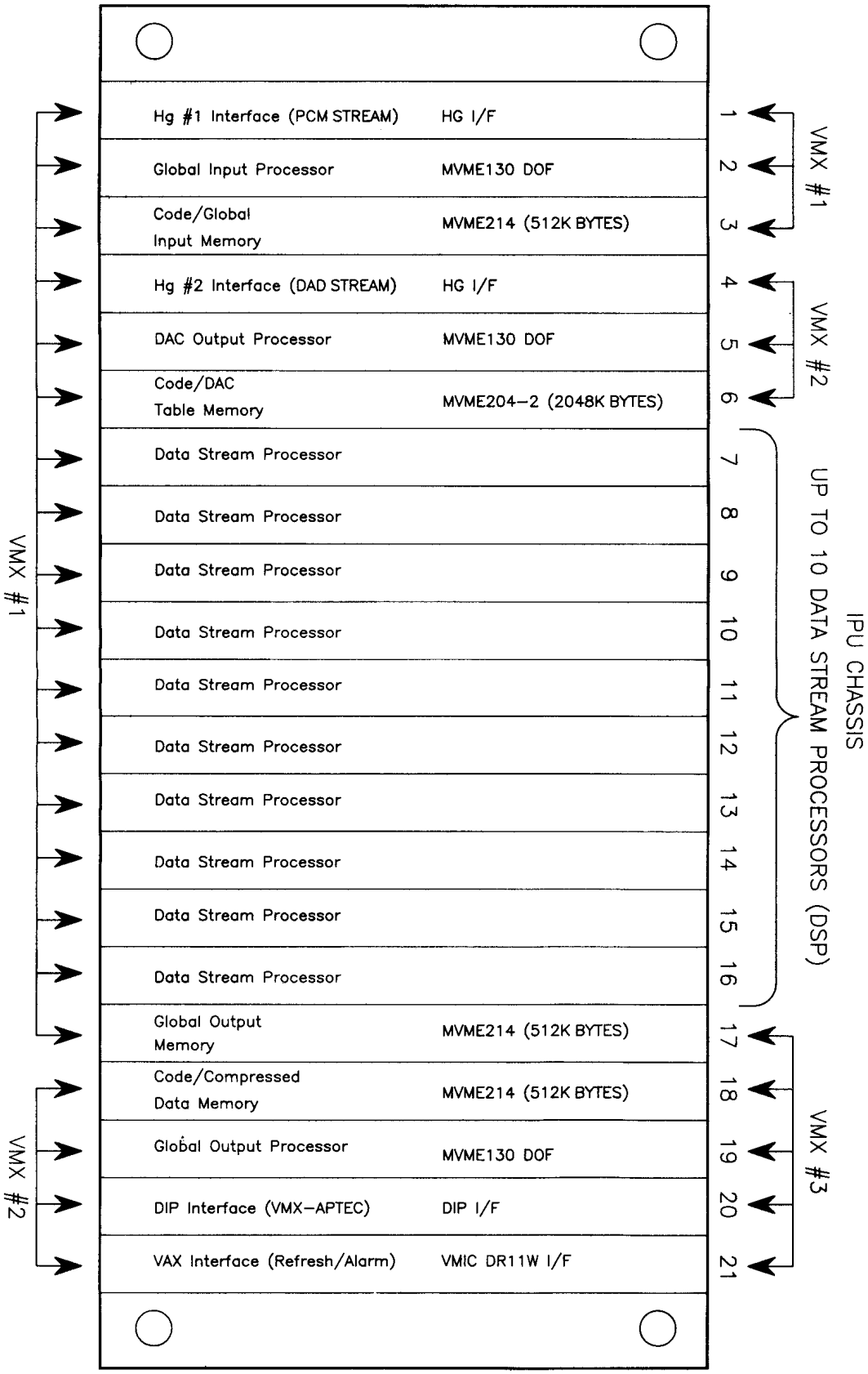

Figure 7. IPU Implementation