OPEN ARCHITECTURE TELEMETRY PROCESSING SYSTEMS

Mark D. McMillen
AP Labs
6215 Ferris Square
San Diego, CA 92121

KEY WORDS

Processing Architectures, VME, Real-Time

ABSTRACT

With the move toward design and interface standards in data acquisition and processing hardware and software, the development of open architecture telemetry processing systems has moved from a goal to a reality. The potential for a system to support hardware and software from a variety of vendors, allow inclusion of user-written software and user-provided interfaces, and provide a scalable, growth oriented processing capability can now be realized. This paper discusses the open architecture concept throughout the hardware and software components of the typical telemetry processing system. Utilizing such a system ensures flexibility to support different configurations, better and faster analysis through greater user programmability, and overall reduced costs by providing a system that can grow as future hardware and software components are brought to market.

INTRODUCTION

Historical telemetry acquisition and processing systems have been proprietary closed systems. This follows the compute platform trend of proprietary system architectures so prevalent until recently. The real time nature of the processing requirement and the magnitude of the processing and I/O required has helped to keep telemetry acquisition and processing systems from following the open systems movement until very recently.

The typical telemetry system requirement is very I/O and processing intensive. Historically this has mandated multiple subsystems to perform the various tasks (bit synchronization, decommutation, data processing), multiple processors to perform the various processing functions (compression, engineering units conversion, derived

parameter processing, etc.) and multiple data buses to handle the I/O traffic. Often a proprietary tag and data bus is used to communicate between the various processing elements. Figure 1 shows a typical closed telemetry processing system architecture.

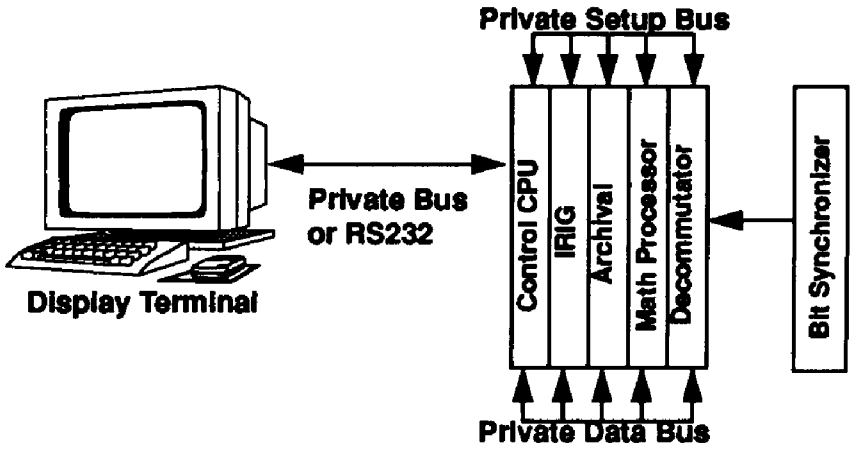

Figure 1 - Typical Closed System Architecture

Architecturally, early systems were entirely closed architecture with very little user customization or programmability possible. All hardware was vendor-specific and no user customization was possible without extensive vendor support. Internal data bus and processor architecture was proprietary in order to support the I/O and processing requirements. Similarly, system software was specific to one platform and built specifically for the system at hand.

Upgrading a system based upon new requirements historically has meant relegating the old system to the scrap heap and procuring a new system. Differing database formats required re-entering old data in a new format or writing a translation program.

With the advent of the workstation and PC compute platforms and the standardization/ publication of a number of data bus architectures, the move towards open systems architectures for telemetry systems began. Recent systems have taken steps towards providing an open architecture solution by utilizing third party workstations as part of the system or providing an Ethernet or PC link for display stations. Unfortunately, a majority of the underlying hardware and software solutions are still been built upon proprietary tag and data bus architectures. This still prevents extensive upgrade and user customization. Figure 2 presents a typical recent system with partially open architecture.

Before discussing why an open architecture is advantageous the term must be defined. The definition is broken into three components: hardware, software and database information. Open architecture hardware is defined as that which is based upon an

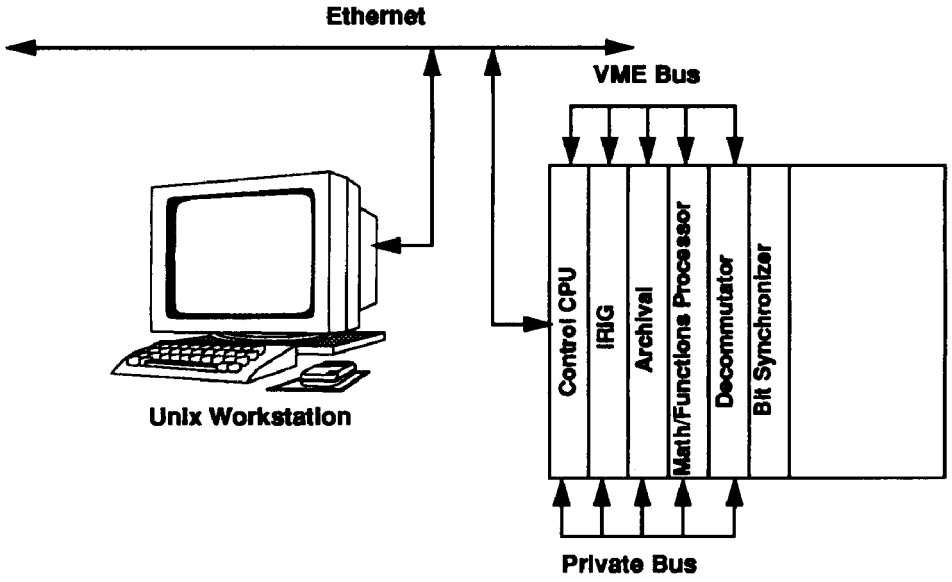

Figure 2 - Typical Recent System Architecture

industry accepted backplane interface for which numerous third party products exist. Examples are the PC and VMEbus platforms. Interface specifications are published (even IEEE-accepted for VMEbus) and there are numerous vendors supporting all manner of I/O and processing solutions. Open architecture software, again relies on standards and is not tied to any one hardware platform. TCP/IP networking protocol software, POSIX-compliant operating systems and windows/Motif graphical user interface software are good examples. Open architecture database information for the telemetry world was given a real boost by IRIG Standard 106-93. The telemetry attributes transfer standard (TMATS) definition provided in this document presents a format which clearly and fully defines a telemetry acquisition and processing configuration. Once TMATS is accepted and implemented on various test ranges, the effort required to pass database information between systems and sites will be minimized.

ADVANTAGES OF AN OPEN SYSTEM

There are numerous advantages to a totally open system architecture. Consider a telemetry acquisition system without the typical proprietary tag and data bus architecture. Removing this constraint allows use of CPUs and other I/O interfaces from a number of vendors that specialize in making fast, reliable, inexpensive CPU s with relatively short product cycles. Having the ability to plug any of a number of different CPUs into a system allows the ability to upgrade processing horsepower based upon requirements without buying a whole new system.

Specialized board level vendors are typically very competitive and sell large numbers of boards. There is significant price savings when compared against a proprietary design which has a much lower number of sales to amortize board development over.

A similar argument may be made for technical upgrades. Board level vendors are forced to bring new products to market frequently. Vendors are continually leapfrogging each other in price-performance. It is much more difficult for a proprietary systems vendor to fuel the R&D fire for the total number of interfaces provided in his system.

From a software standpoint, an open architecture provides interoperability across hardware platforms. The XWindows/Motif graphical user interface standard is a good example, where user workstations from many different vendors may be used as display stations for a single acquisition front end.

The ability to migrate the real time software between CPUs is also of great importance. This is provided by working with a widely ported real time operating system and programming in a standard higher level language.

Another advantage that can be provided on the software end is source code. Users often need to customize processing, add special displays and integrate custom I/O interfaces. For these cases, the only option has long been to develop an entire system from scratch. This is typically expensive, time consuming and the end product often lives and dies with the developing engineers. Providing the ability for users to customize their systems without creating them from scratch, while still allowing a growth path from the hardware standpoint, is an ideal solution.

The TMATS format definition provided in the IRIG 106-93 Telemetry Standards document provides much needed common interface definition for telemetry databases. It achieves the goal of providing a common format for transfer of configuration and processing information between systems and ranges. Once a telemetry configuration is entered in TMATS format it is complete and may be processed on any system that can import TMATS. This will require telemetry processing systems manufacturers to provide programs to import and export TMATS format databases and provide the translation to the local database format, but it is likely that all competitive systems manufacturers will embrace TMATS.

In summary, the advantage of an open architecture telemetry system is the adherence to standards and the growth and flexibility to incorporate boards from multiple vendors provided by the entire industry adhering to standards.

OPEN ARCHITECTURE SYSTEM EXAMPLE

AP Labs' VMEstation Telemetry System (VTS) is an example of a totally open architecture telemetry processing system based on the VMEbus. It may be configured with a wide range of third party off-the-shelf interfaces but the most basic system is presented here for discussion. Figure 3 shows a VTS system with one telemetry input stream, a real time CPU, and a real time archival capability. There is a real time subsystem and a Unix host, in this case co-resident on the same VME backplane. The system may be configured into a network of Unix host workstations. The tightly coupled workstation is the only station which is allowed to control it. Other workstations on the network serve only as passive display devices.

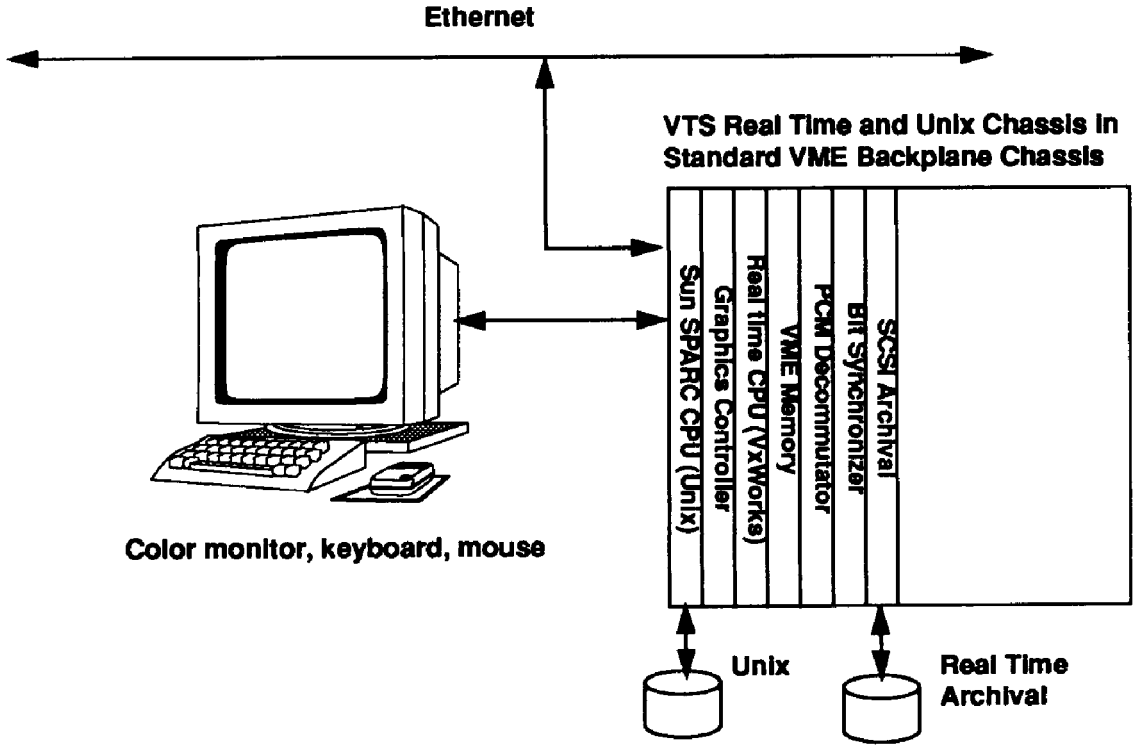

Figure 3 - Open Architecture System Example

In the configuration shown the real time CPU is running the VxWorks real time operating system and the host CPU is a Sun SPARC running SunOS. The real time CPU manages the telemetry input and performs the telemetry processing. Processed data packets are moved into local memory by the Unix CPU and displayed as directed by the operator. It is not necessary to have the Unix CPU embedded in the real time system - alternate configurations provide memory mapped interfaces between host and target or control over Ethernet. The display workstations can be any XWindows workstations. By virtue of choosing a real time operating system which is supported across many platforms, any of a number of host systems (Sun, Silicon Graphics, HP, IBM) and target processors (680x0, SPARC, MIPS, i960) can be configured. The

processing horsepower on both the real time and host configurations is matched to the data rates of the input stream(s) and the amount of computation anticipated. Additional CPUs may be added to provide greater processing capacity. For instance, each input stream may be slaved to its own processor.

A discussion of the data flow through the system is valuable. The system architecture is much like a classical data acquisition system, rather than the historical telemetry processing system tag and data bus approach. The decommutator in this example is a Berg Systems 4411VX which has a pair of onboard data buffers and works as a double buffered input device. The board collects frames of data as directed (any number of frames up to a total of 64KB of data) and generates an interrupt on the VMEbus. The CPU responsible for this interrupt responds to the interrupt and moves the block/frame of data to its own local memory for processing. No tags need be transferred because the CPU has the entire frame of tags resident in its memory and knows the appropriate tag by virtue of the knowing the position of the data word within the frame. This halves the I/O bus requirement immediately. Assuming that the CPU can perform all necessary processing for a given stream (typically a good assumption), no additional bus traffic is required until results are moved to the Unix processor for display. Multiple CPUs may be used to implement extensive processing if necessary.

Very high rate data and/or large numbers of streams may present a situation where the VMEbus becomes saturated. In this case the I/O load may be balanced by utilizing another industry standard I/O bus, the VSB or VME Subsystem Bus, for the interface between the decommutator and the CPU. The VSB works in parallel with the VMEbus and does not impact its performance at all. Multiple VSB buses (from 2 to 6 slots each) may be implemented on a single VMEbus backplane. Each VSB bus has an achievable I/O bandwidth of >20MB/second. The VSB bus acts to balance real time I/O with the VME bus and effectively takes the place of the historical proprietary tag and data bus for higher rate systems. Utilizing VSB decommutators, CPUs and other I/O interfaces allows assembly of very high performance non-proprietary systems.

Configuring a new/custom I/O interface into the system involves writing a software driver for the interface and a real time control daemon (task) to handle the interface. All other VME I/O is handled in the same manner, with a separate task dedicated to each I/O interface.

Software consists of a Motif/XWindows-based graphical setup and display capability for the Unix workstation, and real time device drivers, asynchronous I/O capability and processing daemons for the real time CPU(s). All software is written in C and is licensed for customers who wish perform extensive customizations.

THE FUTURE OF OPEN SYSTEMS IN TELEMETRY PROCESSING

With the move towards real time video transmitted via telemetry, faster and greater numbers of transducers and faster CPUs, it is obvious that telemetry data rates will continue to increase. The need for a higher speed I/O interface will eventually drive vendors to adopt new standards. The best open system approach available to fill this emerging need is Futurebus+. This extensible interface standard is in its early stages but is designed to grow over the next 15 years from its current 100MB/second to support an I/O bandwidth of >3GB/second. Bridges to other architectures (VMEbus, Multibus II) are already available as well as a minimum offering of other hardware but it will not be a viable solution for high speed data acquisition and analysis systems until a greater number of I/O interfaces are supported.

The goal should be an easy migration of the higher level software from existing open systems to support the newly developed hardware. Keeping a similar interface to the display workstations and preserving the TMATS interface should be simple and provide continuity as well.

CONCLUSIONS

Open architecture telemetry acquisition and processing systems are available today and becoming widely accepted. The benefits for the user community are a greatly extended system life cycle, lower system costs, greater user flexibility/customization and a common telemetry database information format.

REFERENCES

1. IRIG Standard 106-93 Telemetry Standards, Telemetry Group Range Commanders Council
2. Futurebus+ Logical Layer Specifications, Multiprocessor Standards Committee of the IEEE