

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

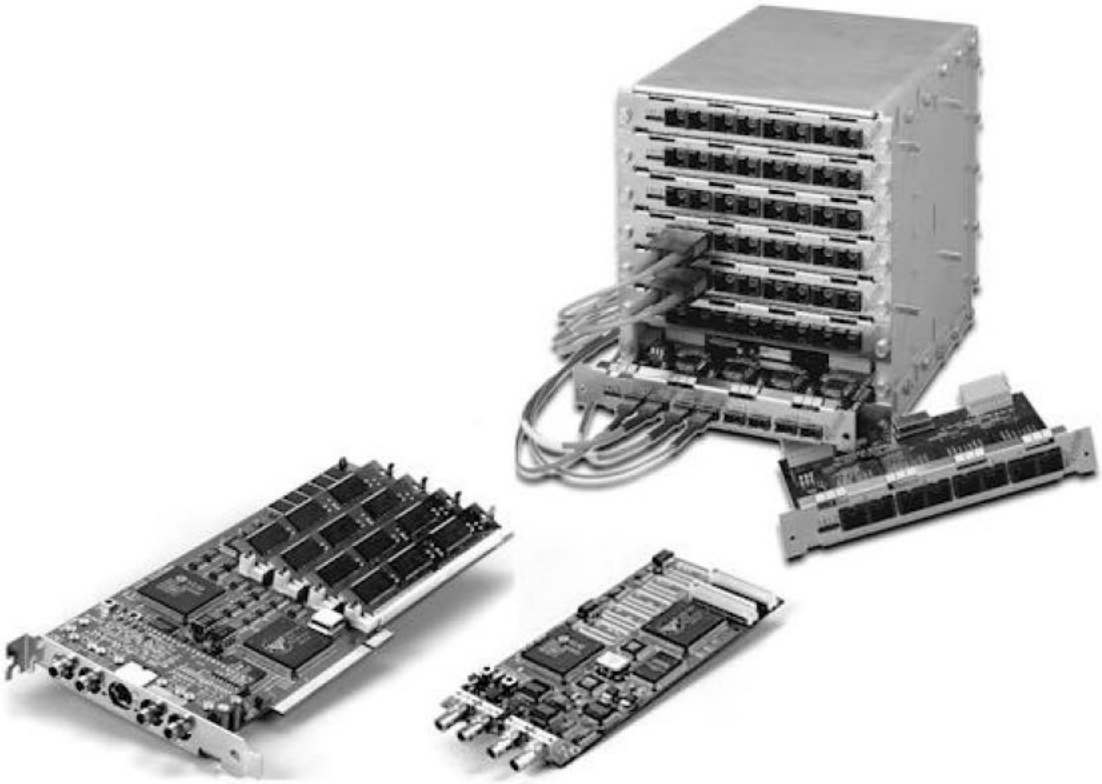
Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Curtiss-Wright / Systran SCRAMNet GT200* at our website: ***Click HERE***

Technical Description Document

SCRAMNet+ Network



Available From

Systran™

Corporate Headquarters
4126 Linden Avenue
Dayton, OH 45432-3068 USA

Phone: 937-252-5601
Sales: 800-252-5601 (USA only)
Fax: 937-258-2729

Product Specifications are subject to change without notice SCRAMNet and FibreXpress are registered trademarks of Systran Corporation Systran, the Systran logo, FibreXtreme, LinkXchange, BusXchange, 1553 Solutions, I/O Solutions, SCRAMalyzer, and EZ-SCRAMware are trademarks of Systran Corporation All other trademarks mentioned herein are the sole property of their respective owners © 2000, Systran Corporation All Rights Reserved
Doc F-T-SH-SNTDD-A-0-A1(05/05/00)

I. FOREWORD

A. Document Purpose

This document serves a twofold purpose for engineers and systems analysts who are evaluating the SCRAMNet+ Network for their real-time applications. First, it introduces the SCRAMNet+ Network product (the level of technical information about SCRAMNet hardware falls between a product specification sheet and a hardware/software systems reference manual). Second, it provides a tutorial covering issues related to real-time systems development. No publication of this type can completely answer all questions. Hopefully, this one adequately answers the investigative questions *Who? How?* and *Why?* as they relate to the SCRAMNet+ Network.

B. Additional Data

Need more information? Product pricing? A sales quote? Advice on a particular application? Call SYSTRAN in Dayton, Ohio (telephone 937-252-5601) and ask for Sales. SYSTRAN sales personnel will be happy to assist you.

This page intentionally left blank.

II. INTRODUCTION

A. The Unique Demands of Real-Time Computing Systems

Real-time computing is an industry-specific term uniquely defined by those in the real-time computing community and bound only by their own specialized application experience. In general, though, real-time computing refers to a demanding world of high algorithm computational loads and performance envelopes where system events must occur in milliseconds and microseconds.

Real-time computing usually involves a "complex" system design, namely, several interconnected computers working together to handle distributed computational tasks. Typical distributed real-time applications include man-in-the-loop, hardware-in-the-loop and virtual reality simulations for training and systems development/maintenance facilities. Other applications include instrumentation and control for telemetry, data acquisition wind tunnels, particle accelerators, engine test stands and any high-fidelity feedback control system. Real-time applications have unique and demanding data communication requirements that cannot be serviced by most commercial data processing networks.

Minimal Transport Delay (Low Data Latency)

The first unique real-time requirement is for minimal data transport delay. *Transport delay* measures how long it takes a network to pick up a data payload from one CPU and deliver it to another (transport delay may also be called A-T-A, or "application-to-application" time). The need for minimal transport delay results from applications running modules and algorithms repetitively at rates from 1 to 2,000 hertz. The data outputs from these modules serve as data inputs for software application modules running on other distributed CPUs. In the real-time world, data transfers between CPUs must take place within microseconds. Contrast this with data processing network applications that take several milliseconds just to establish network media access.

The time it takes for an application on one CPU to have its data outputs seen as data inputs by an application on another CPU is the measure of merit for real-time data transfer. Unfortunately, for real-time system architects, most networks on the market today have been targeted for the data processing world. They are designed to maximize bandwidth efficiency (the number of bits of data divided by the total number of bits transmitted) with very little attention given to minimizing transport delay.

Deterministic Performance

The second unique real-time requirement stems from the need to minimize data transport delay. Simply achieving a low A-T-A time is not good enough for most real-time system architects. Data communication must also be highly predictable, or "deterministic". Transferred data must readily be available for the other applications to use, within a specified and predictable time period, *on time, every time.*

For example, an avionics support simulator must exercise mission hardware and software in a real-world fashion for successful functional verification and validation. This may mean flying demanding, man-in-the-loop, simulated missions to fully exercise all possible flight operation modes. If the simulated environment supporting the mission hardware/software cannot keep up for even one frame time, the simulation may fail and the mission hardware/software may not be fully validated. Also consider a special operation force preparing for a mission on a complex high-fidelity mission rehearsal simulator composed of several independent but linked air- and ground-vehicle simulators. The usefulness of the rehearsal is diminished if the simulators cannot interact in a real-world, real-time manner due to excessive data latencies. Finally, imagine the danger posed by a launch control system in which the telemetry data monitoring the launch vehicle parameters occasionally arrive late to the control station.

The demand for predictable data flow places a rigid need for determinism on network protocol - a determinism simply not possible with most OSI-compliant networks. Layered OSI protocols, built on the physical and data-link layers, are most generally implemented in software. While meant to increase the capabilities of the networked system for both message routing-handling and data encoding-decoding, this software brings such variance to network operations that even simple network procedures like network media access adversely effect the predictable performance envelope which defines network data transport delay.

Transparent I/O

"Transparent" network I/O is a third unique real-time requirement. Unlike most data processing processing applications, real-time systems are CPU intensive. Since CPU time is at a premium in these systems, the CPU is better used for the needs of application modules and algorithms than for executing I/O. This means I/O-related communication needs to take place outside and apart from the host CPU.

B. Traditional Communication Approaches

Traditionally, only two approaches have been offered as solutions for the data communication needs of distributed real-time computing applications: *physical shared-memory architectures* and *message-passing networks*. (The SCRAMNet Network, a third alternative that combines the strengths of the traditional approaches while eliminating their weaknesses, is discussed later.) Physical shared-memory architecture (Figure 1) uses a high-speed parallel memory bus through which all com-puters can access a common-memory module (a variation of this structure uses multi-ported memory in lieu of the common bus). Physical shared memory offers the required data communication speed along with other real-time benefits:

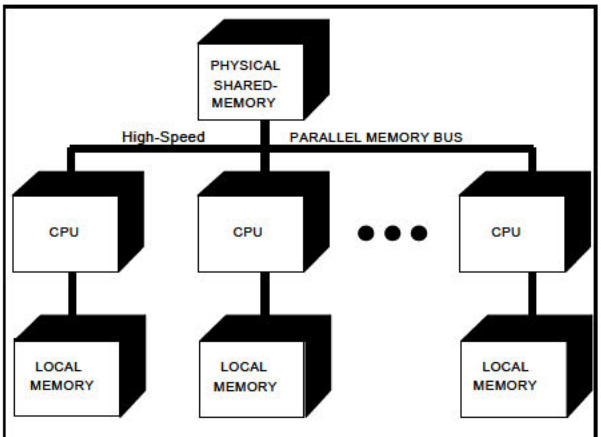


Figure 1
Physical Shared-Memory Architecture

- Minimal transport delay
- Efficient communication of small data packets
- Transparency (virtually no software overhead is needed for data communication
- Fast system reconfiguration
- Fast error recovery
- Determinism

The second traditional communication approach (Figure 2), the message-passing local area network (LAN), uses a serial communication link through which all computers pass message packets. The LAN approach *has* helped overcome many physical shared-memory limitations. Specifically, LANs:

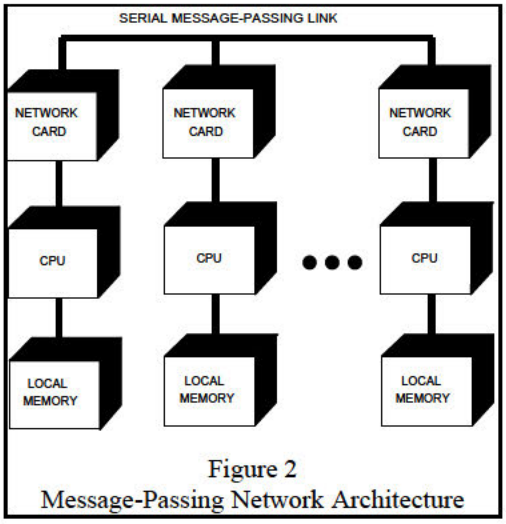


Figure 2
Message-Passing Network Architecture

- Allow the integration of different vendors' CPU hardware,
- Allow for a much larger number of CPUs,
- Permit the physical separation of CPUs to distances of hundreds or even thousands of feet,
- Provide pricing and support advantages (targeted for data processing applications, LANs enjoy large sales volumes resulting in excellent pricing and support).

These important advantages provide growth potential for networked systems and foster competitive procurement of system components. However, LANs have reintroduced many of the original problems facing real-time data communication.

The major real-time drawbacks of LANs are:

- Transport delay (data communication is orders of magnitude slower than that of physical shared-memory),
- Extreme inefficiency for small communication packets,
- Lack of I/O transparency (large software overheads "eat up" CPU time),
- Lack of determinism in performance (large performance envelopes make it difficult for message-passing networks to provide or maintain synchronization over the
- distributed computer system,
- Difficult and time-consuming error recovery and system reconfiguration.

C. The Replicated Shared-Memory Approach

Table 1 reveals that the two traditional distributed real-time communication approaches are actually complementary: where one is weak, the other is strong. What the real-time system architect needs, however, is a third networking alternative that combines the strengths of the traditional approaches while eliminating their weaknesses. Such an alternative does exist in the SCRAMNet replicated shared-memory network by SYSTRAN Corp.

The third column in Table 1 shows the many advantages of replicated shared-memory technology. First, replicated shared memory combines the most important strengths of the traditional approaches to provide minimal transport delay, determinism, low software overhead, and fast error recovery. However, these gains do not come at the expense of other critical advantages: having a large number of connected nodes, connecting nodes over long physical distances, using different vendors as sources for obtaining host CPUs.

Table 1
All Three Alternatives Compared

Feature Required	Physical Shared Memory	Message-Passing LAN	Replicated Shared Memory
Low Latency/High Speed Inter-Processor Communications	X		X
Tight System Synchronization	X		X
Use of Multiple Vendors' CPUs		X	X
Allows Large Numbers of CPUs		X	X
Allows Physical Separation		X	X
Low Software Overhead	X		X
Fast/Automatic Error Correction	X		X
Optimum Small Data Packets Handling	X		X
Deterministic Performance	X		X

Second, there is a simplicity to the replicated shared-memory architecture that helps lower system development and maintenance costs. Except for initialization, a replicated shared-memory system has no communication-related software. Because of this, there are fewer lines of code to write, debug, and maintain. Also, the logical architecture of the replicated shared-memory system is very similar to that of one virtual computer executing multiple tasks. This structure reduces the initial system learning curve for developers and programmers, further lowering implementation costs.

Even though physical shared memory offers true real-time data communication, it still has system-related limitations, specifically:

- Memory bus contentions,
- Limited physical distances between computers,
- The use of only a single vendor's hardware,
- The drawback of being a "special" (often very expensive),
- Connection to only a few computers,
- Limited number of CPUs (*e.g.*, two to eight).

III. SCRAMNet+ Conceptual Overview

A. What is a Replicated Shared-Memory Network

The SCRAMNet replicated shared-memory network is a rather simple concept, and much of elegance and power derive from this simplicity. This design operates by placing a SCRAMNet card in each computer to be connected -- just like a message-passing network. The cards then communicate over a serial ring architecture (Figure 3). However, this is where the similarity to message-passing networks stops.

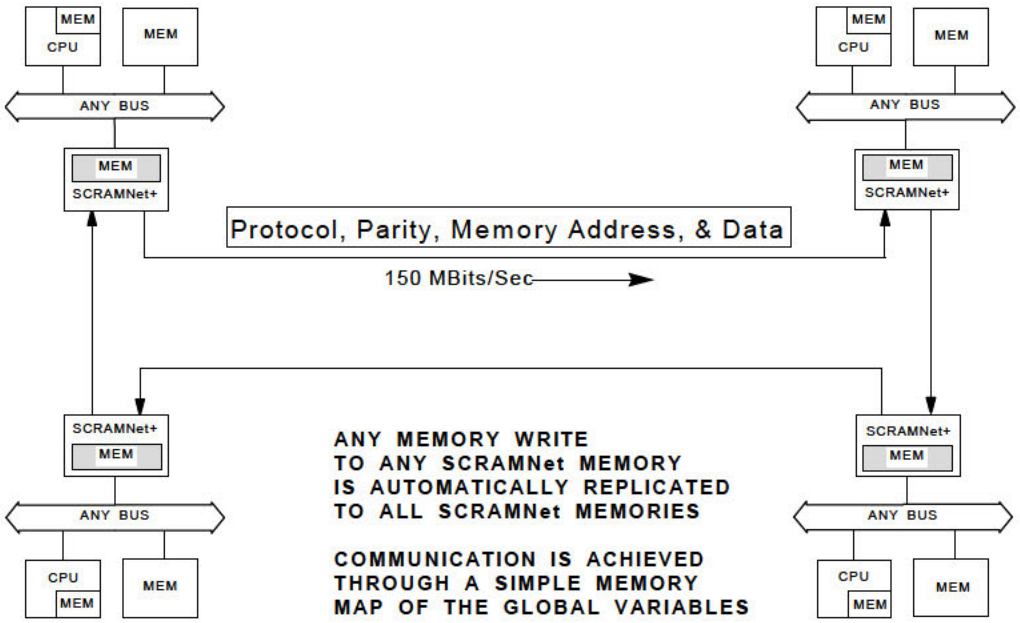


Figure 3
SCRAMNet Shared-Memory Architecture

Replicated shared-memory networks do not attach lengthy protocol information, or pass these complex messages with cumbersome hardware and software protocols. Replicated shared-memory networks do not use the seven layer ISO implementation. Replicated shared-memory networks obviate the need for these cumbersome, time-consuming, and non-deterministic protocols found in message-passing networks.

In a replicated shared-memory network, a datum is communicated very quickly and directly -- the transmitting CPU simply makes a high-level language statement: A = B, where the variable A is located in the shared-memory window of the SCRAMNet card. Within microseconds all other CPUs on the network have the same value of the variable A in their shared-memory areas. The SCRAMNet card does this immediately and automatically by passing both the datum and the datum address over the network. The variable is written to the same relative shared-memory address in each computer's shared-memory area by the receiving SCRAMNet cards.

Control -- another critical performance requirement of simulators -- can be passed over a replicated shared-memory network by the same mechanisms as data are. Interrupts can be transmitted to all (or selected) CPUs within microseconds. For example, on a ten node network, with each node separated by 50 meters, all ten CPUs can be interrupted in a maximum of 9.5 microseconds. Likewise, a new datum value can be transmitted to all ten CPUs in the same maximum of 9.5 microseconds. Moreover, the minimum time for a data or interrupt message to traverse a ten node ring is 4.5 microseconds.

This replicated shared-memory network approach also provides a very deterministic performance envelope for the system designer. Control and data communications take place in a few microseconds for an upper-bound, and there are no non-deterministic software routines which must be invoked to complicate this process. Also, using the ring network with known, fixed-length transmissions, there are no “unknowns” in the transmission link transport times.

B. Topology

Specifically designed for the real-time market, the SCRAMNet Network blends the benefits of the replicated shared-memory design with the ring network topology. Ring topology was chosen for SCRAMNet over bus or star topologies because it offers an excellent combination of networking characteristics that uniquely enhance SCRAMNet's real-time networking mission.

For instance, many real-time systems require absolute data integrity. No data errors can be tolerated; if an error occurs, it must be corrected immediately. The ring topology can implement an error detection-correction scheme that is virtually unbeatable. With ring topology, a message packet placed on the network circulates around the ring until it is removed by the originating node. When the node originating the message removes the packet (called *source removal*), the node can do a bit-by-bit comparison of the returning message with the original. Any difference causes the original message to be immediately retransmitted. Many error detection-correction schemes can be implemented on each of the network topologies, but none is as robust, yet easy-to-implement as the source removal and compare scheme for the ring topology.

Furthermore, the physical structure of the ring topology makes it highly compatible with fiber optic technology. State-of-the-art fiber optics have become a requirement for many real-time systems because of their many inherently strong real-time capabilities. For example, fiber optic transmitters can clock data onto the optical media at ultra-high rates over extremely long distances. Fiber optic cables are immune to noise interference. With fiber optics, ground loop problems are eliminated because the nodes are electrically isolated. And optical cables are naturally immune to electromagnetic interference and electromagnetic compatibility (EMI and EMC) problems. For users who can forego the requirements that dictate the usage of fiber cables, a standard coaxial cable can be used.

Other noteworthy strengths of the ring topology include:

- Ease of implementatio
- Connectionless service to a large number of possiblenode connections
- Simple and efficient media access

SCRAMNet's ring topology can be implemented using three media types. They are:

- Standard Link Multi-mode Paired Fibre Cable
- Long Link Multi-mode Paired Fibe Cable
- Coaxial Paired Conductor Cable

The standard link fiber can be used for distances up to 300 meters and the long link fiber can be used for distances up to 3500 meters. The cable has a core/cladding diameter of 62.5/125 microns for both standard and long link. The connectors are ST type for both fibers in the cable. The coaxial cable is a paired conductor RG-58U 50 ohm cable utilizing an SMA type connector.

C. Direct Backplane Support

Since today's real-time environment is composed of many different computer systems architectures SCRAMNet supports most of the popular bus architectures. These are:

- VME3, VME6U & VME9U
- ISA
- EISA
- GIO
- SBus
- PCI
- PMC

The SCRAMNet card plugs directly onto the systems backplane and appears to the system as a memory card. The programmer then makes an assignment statement in the form the system requires to map the memory of the SCRAMNet card into the program's address space.

In order to provide support for those backplanes we do not currently (or plan to) support with a standard product, SYSTRAN has developed a Rehostable card that provides all the core logic of the SCRAMNet network card such that a user could use this product in conjunction with his own host interface design to allow SCRAMNet access for a system we do not support directly.

D. Modular Top-Level Architecture

As discussed previously, to ensure that SCRAMNet can successfully serve the diverse and demanding requirements of the real-time computing market, SYSTRAN Corp., SCRAMNet's developer and manufacturer, has combined its unique replicated shared-memory concept with a network topology and protocol that enhances all supported real-time performance requirements.

However, performance is not the only measure of success in the real-time community. Platform versatility is another key factor. Therefore, SCRAMNet has been designed so that it can be hosted by most CPU's currently used by real-time applications. Modularity is the key to this platform versatility. SCRAMNet's top-level functional breakdown is shown below in Figure 4.

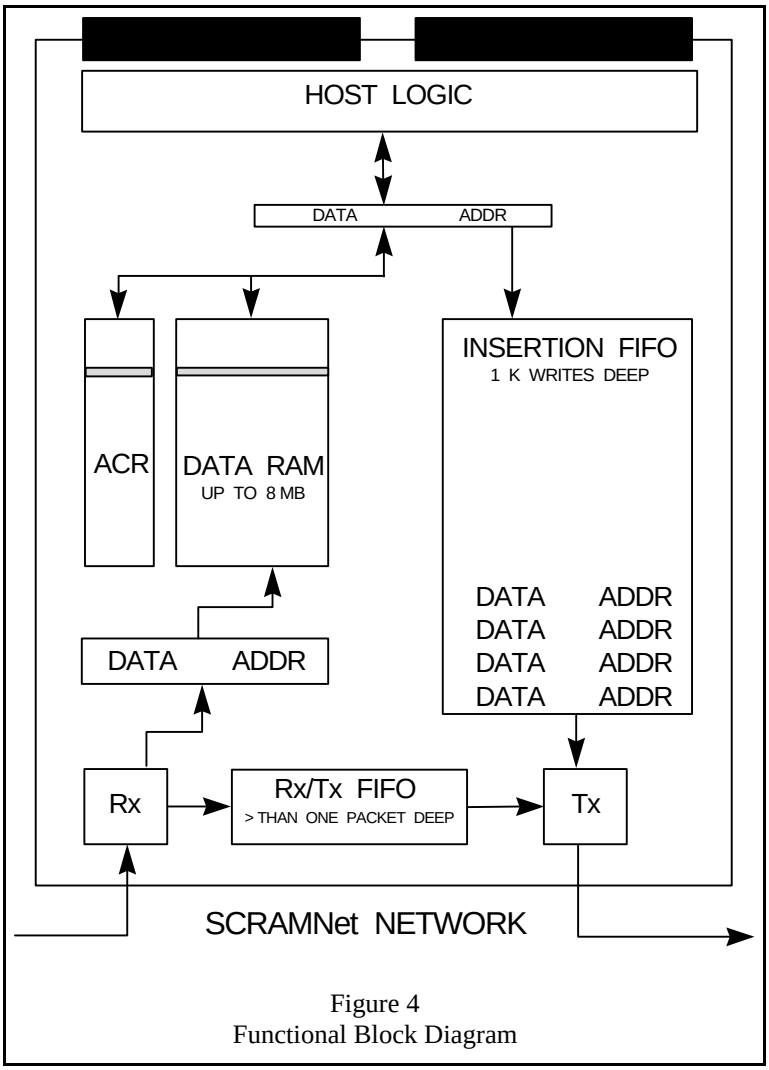


Figure 4
Functional Block Diagram

Host Board

SCRAMNet employs an ASIC based core logic that is reused as the basis for every bus or host implementation. This core logic provides functions such as message reception and transmission arbitration, network control, memory management, interrupt handling, error detection and correction and node configuration to name a few. This core logic is then integrated with host or bus specific logic into a form-factor that allows it to be used in multiple computer systems. It is this modular architecture that allows easy porting to the host CPU and enhances system-level network troubleshooting and maintenance.

Media Interface

Modularity in the media interface is provided by the Media Access Card (MAC). This plug-on module provides all of the interface to the media for all of the receive and transmit functions. As stated before, the media can be coaxial, standard fiber or long-link fiber. The MAC can be easily changed in the field by the user and therefore allows quick media type transition and field repairability.

SCRAMNet Memory Options

Most of the SCRAMNet cards come with an ASIC based 4KBytes of memory. In addition to this built-in memory, most host cards can also be populated with 128KBytes of on-board memory. The Rehostable can hold up to 2MBytes of on-board memory. For the 6U VME, EISA, ISA and PCI cards, growth up to 8MBytes is accomplished via high-speed SRAM SIMM modules. For the 3U VME, SBus and GIO cards, growth up to 8MBytes is accomplished via a plug-on memory card.

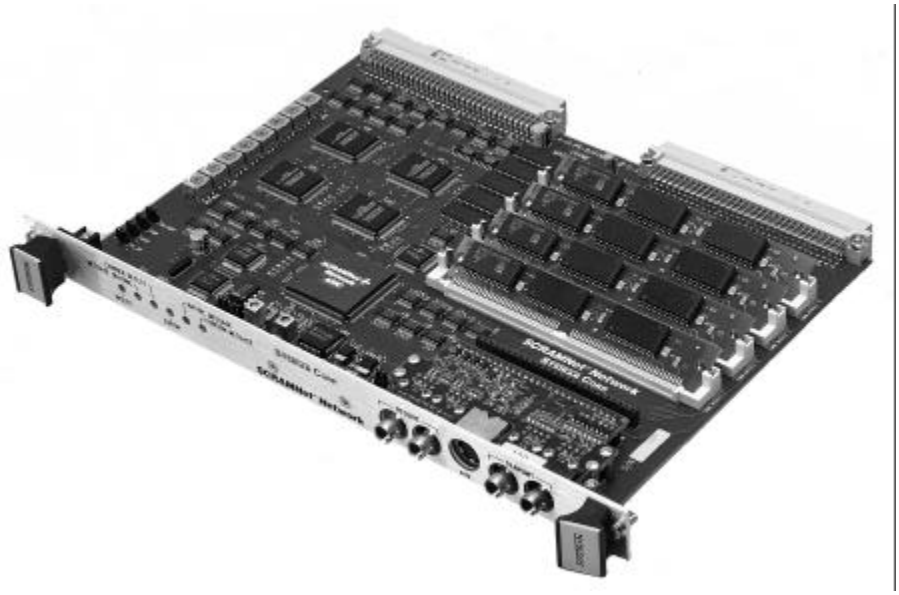


Figure 5
6U VME SCRAMNet+ Card

This page intentionally left blank.

IV. SCRAMNet+ Fundamentals

A true real-time networking solution, the SCRAMNet Network offers advantages unique to replicated shared-memory communication:

- As easy to use as memory, SCRAMNet equips each network computer with its own local copy of shared memory. The shared memories of all network computers are updated via a high-speed, serial-ring fiber optic network link.
- Like memory, SCRAMNet usually requires no separate software driver to pass information.
- SCRAMNet can make more efficient use of its bandwidth by automatically filtering out unchanged data values.
- SCRAMNet's deterministic performance and Platinum protocol, an optimized ring network protocol, provide quick network access and low application-to-application message latency.

Earlier versions of SCRAMNet were at a minimum a two-card set (mother-board and daughter card) and at a maximum a separately enclosed table top or rack-mount unit with enclosure adapters to provide bus and network cabling.

SCRAMNet+ is now a single board device that is intended to mount within the enclosure of the host computer. Cabinet kits are still an available option for those computer systems with embedded chassis where physical restrictions make alternate routing for the network cabling and connections desirable.

SCRAMNet+ also supports a wide variety of hardware platforms and is a more efficient user of system power resources due to its integrated logic circuit design. Overall SCRAMNet+ is a major step forward in board-level design and uses state-of-the-art technology to the best advantage of the using organization.

These features plus its elegance and ease of use in the operational environment, described below, make SCRAMNet+ the ideal systems interconnect for demanding distributed real-time applications.

Because SCRAMNet+ is part of a family of products that includes the original SCRAMNet product, the SCRAMNet Classic, SYSTRAN has maintained where possible operational compatibility. This allows an organization with a significant investment in Classic SCRAMNet to migrate to SCRAMNet+ without losing functionality and also protect their investment.

A. Interconnecting the Network

The ring topology (as previously discussed) is implemented by running a paired coaxial or fiber cable from the SCRAMNet transmitter connector pair to the receiver connector pair of the next SCRAMNet node and then repeating this for every node until you have connected all of the nodes as in Figure 6. The ring is unidirectional, hence, the reason you connect the transmitter of one node to the receiver of another node. Data flows in only one direction and is received, stored and retransmitted to the next node until the data returns to the originating node. A paired conductor cable is used due to the nature of the SCRAMNet protocol and does not provide redundancy.

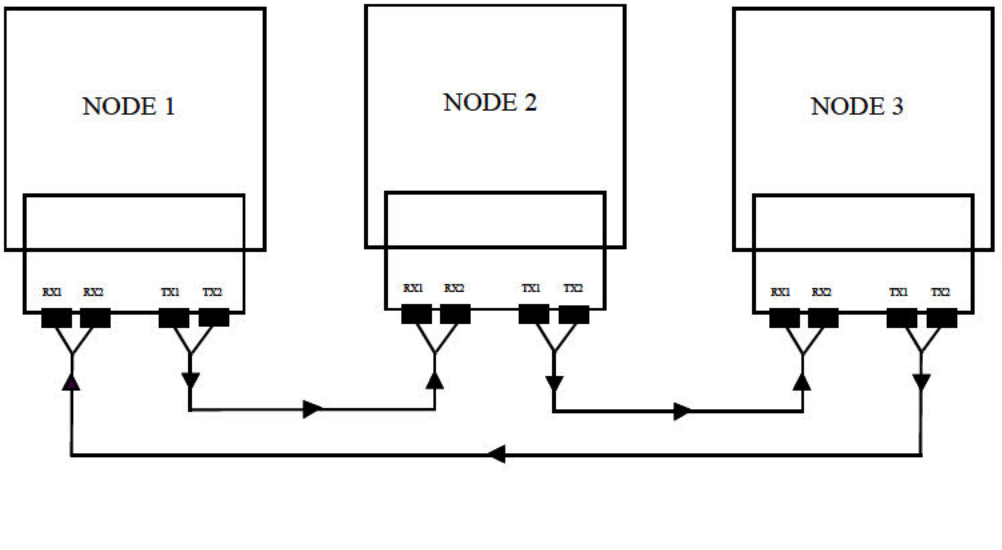


Figure 6
Node Interconnection Methodology

B. Protocol

Providing real-time response *and* ease of use, the SCRAMNet Network is the first network with communication protocols specifically selected and developed for the real-time community. Other networks are modeled after standards developed around the International Standards Organization's (ISO) seven-layer model of the communication process.

This model, called the Open Systems Interconnection (OSI) reference model, (Figure 7) provides a hierarchical definition of all the functions necessary for communicating data between CPUs. Each layer has its own functional description and purpose. Also, each layer depends on services provided by a lower hierarchical layer and, in turn, provides services to the next higher layer.

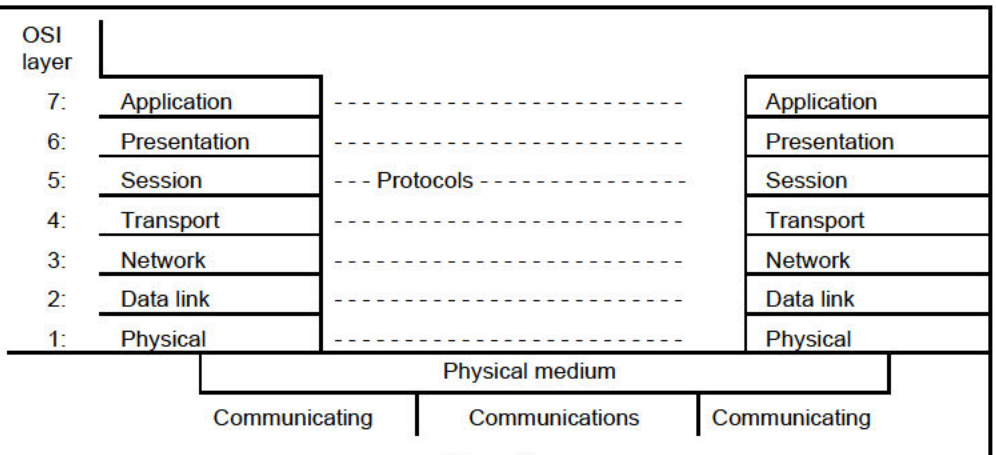


Figure 7
OSI Model

The major drawback of OSI for the real-time community is that its protocols were developed to provide an interface standard capable of supporting a variety of end-user applications. Performance and ease of use have been sacrificed in favor of serving a broad customer base. SCRAMNet's protocols, on the other hand, have been optimized for real-time applications.

The Network Physical Layer

The SCRAMNet Network's physical layer is the basis of its real-time protocols. State-of-the-art technology enables SCRAMNet to offer real-time power and performance in one highly usable package. Consider SCRAMNet's features:

1. Clock rate of 150 million bits/second
2. Transmission over paired fiber optic or coaxial cables
3. "Ring" configuration
4. Missing Transition Code data encoding scheme with extremely low bit error rate of 10^{-15}
5. Network throughput ranging from 6.5 Mbytes/sec to 16.7 MBytes/sec,
6. Internal node latency of only 250ns (min. for a 4-byte data packet) to 61.8us (max. for a 1024 byte data packet) at each node
7. Accommodation of up to 256 nodes per network ring
8. Data packet size from 4 bytes to 1024 bytes

Network Media Access Control

The SCRAMNet Network uses a register insertion technique for media access control. Technically, this means that each SCRAMNet node on the ring contains a register of sufficient size to temporarily buffer circulating message packets. In terms of performance, it means that SCRAMNet possesses three key real-time capabilities:

1. *Deterministic performance with no data collisions*: The register insertion media access protocol guarantees all network nodes can have messages outstanding on the ring at the same time. Data collisions are avoided even when all nodes transmit simultaneously.

2. *Very fast media access time:* The register insertion media access protocol guarantees that a node transmitting its own data onto the ring can do so in the minimum time possible. Only point-to-point links are able to achieve faster media access.

3. *Maximum network bandwidth usage efficiency:* Combining the register insertion media access protocol with known message packet granularity and low internal node latency provides automatic load balancing of total allocatable network bandwidth among all nodes offering network traffic. Equal bandwidth portions are allocated to nodes as long as they are using it to offer new data onto the ring. Automatic load balancing extends to allowing *one* node to use *all* available network bandwidth if it is the only node offering data.

With this protocol, a node can transmit its own data onto the ring during any inter-message gap. If a circulating message packet is received while the node is transmitting its message packet, the incoming packet is buffered in the Rx/Tx register. Once the node has completed its transmission, it empties the Rx/Tx register and begins passing the circulating message packet.

Four Real-Time Network Layer Protocols

The SCRAMNet Network satisfies most real-time application requirements with four user-selectable network layer message protocols. These protocols are described below.

Platinum Network Protocol

Platinum protocol guarantees delivery of error-free data within minimum and predictable time periods. Based on the register insertion media access protocol, Platinum protocol allows multiple messages from each node onto the network ring at any given time and uses a fixed-length packet that is compatible with the operational mode of SCRAMNet Classic. When the Platinum mode error correction facility detects an error in the returning message packet, or a message packet does not return to its originating node within a specified time, the original message packet is automatically retransmitted. Appropriate error reporting status is then indicated in the initiating node's SCRAMNet Control and Status Register (CSR). Platinum mode maintains network fairness while providing the capability for one node to use the entire bandwidth. The maximum bandwidth of the SCRAMNet Network under the Platinum protocol is 6.5 MB/sec.

Platinum+ Network Protocol

Similar to the Platinum protocol the *Platinum+* protocol adds an additional dimension to increase the overall throughput of the network. This feature allows the user to select the option of having a variable length data packet. Under the Platinum+ protocol the data packet maximum length can be set to be either 256 bytes or 1024 bytes. This user selectable option requires that the data (being formed into the longer than 4-byte packets) be on sequential word addresses within the shared memory. During transmission the transmitter logic checks the address of the next word stored in the Transmitter FIFO and if it is a sequential address then that word is appended to the data packet without an address location being added. This process continues until either the maximum packet length is achieved, a non-sequential address is found or the FIFO becomes empty. Therefore, a packet less than the maximum length can be sent.

Due to the variable length data packet, network throughput is increased and does so based on the length of the data packet being transmitted. Figure 8 shows the throughputs possible for various lengths of data packets. Note that a sequentially written data structure of just 32 bytes increases the throughput to approximately 13 MBytes/Sec.

Burst Mode Network Protocol

The *Burst mode* protocol also maintains network bandwidth fairness while providing the capability for one node to use the entire bandwidth under certain circumstances. With burst mode, all nodes offering network traffic are able to transmit simultaneously, equally sharing available network bandwidth. However, if only one node is offering data while the other nodes are "listening", burst mode allows this node to use the entire 6.5 MB/sec. bandwidth. When more than one node is transmitting in burst mode, the effective per-node output is the 6.5 MB/sec. maximum bandwidth divided by the number of transmitting nodes.

Based on the register insertion media access protocol, burst mode allows nodes to transmit messages onto the network ring continuously without waiting for messages to return. Parity error detection takes place at each node and corrupted messages are removed, but the originating node does not retransmit the message if an error is detected. Messages that return to the originating node are always removed by that node without the source-compare process occurring.

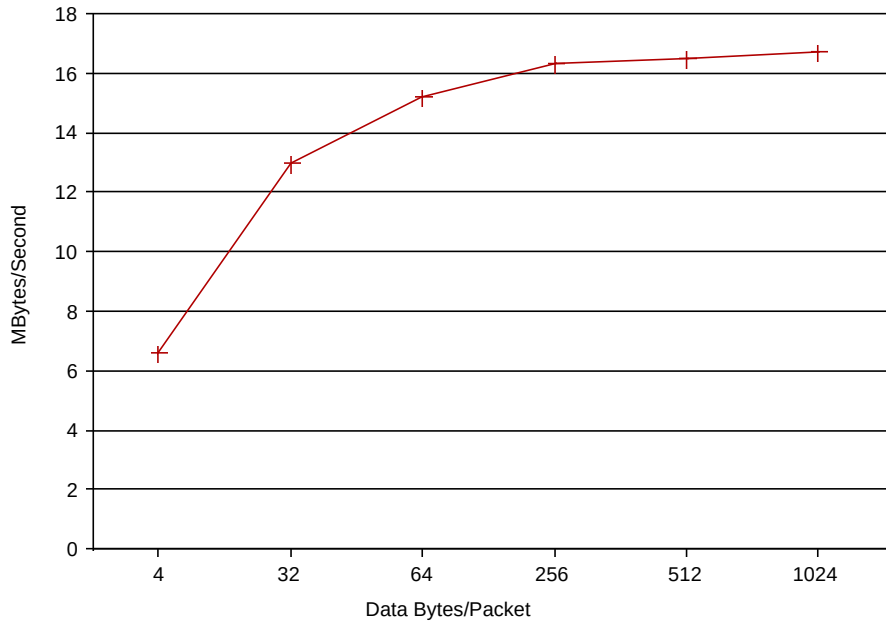


Figure 8
Network Data Throughput

Burst+ Mode Network Protocol

Similar to Burst, the *Burst+* protocol allows for variable length records in the same fashion as the Platinum+ protocol. As with Platinum+, this increases the maximum throughput to 16.7 MBytes/sec.

Automatic Error Detection and Correction

Totally transparent to the real-time application program, SCRAMNet's error detection and correction mechanisms are the result of sound engineering based on a thorough understanding of both the technical issues surrounding bit error rates and the data integrity requirements of real-time systems.

To begin with, the rate at which SCRAMNet generates bit errors is a function of the signal-to-noise ratio at the input to the fiber optic receivers. Therefore, the bit error rate increases if noise levels rise or if signal strength decreases. Historically, noise induced from electromagnetic interference (EMI) on the connecting wire cables has been a major concern to real-time network designers. However, SCRAMNet's use of fiber optic cables reduces and confines signal noise concerns to only the electronics directly connected to the receiver or to other external noise transmitters close enough to SCRAMNet receivers to impact the SCRAMNet environment. For this reason, SCRAMNet receivers have built-in design features and performance margins that reduce the possibility of internal circuit noise and shield against external environmental noise. This means that the average SCRAMNet user should have an actual bit error rate of approximately 10^{-15} - meaning an error could occur roughly once every 76 days of continuous, 24-hour, 100% bandwidth-saturated network use.

Even with SCRAMNet's 10^{-15} error rate, some customers have applications that need the comfort zone provided by an error detection and correction methodology. For example, any application using the SCRAMNet network as part of a deployed weapon system requires the extra data integrity insurance provided by error detection and correction. However, because errors on the SCRAMNet Network are so few and far between, the error detection and correction methodology must be such that it does not degrade normal network system performance in any way.

Under the Platinum protocol, SCRAMNet's error detection and correction is exactly what real-time applications require for data integrity. It is fast, efficient and, best of all, user transparent. Platinum protocol guarantees accurate data delivery.

Platinum protocol uses a "source message packet removal and compare" error detection and correction scheme. Each message packet has a source ID field that is unique to the originating node. Source removal takes place when a SCRAMNet node receives message packets *with a source ID equal to itself* (*i.e.*, its own message packet) from the network. After receiving its own message, the node performs a bit-by-bit comparison of the *received* message with the *transmitted* message (the transmitted message is saved in the Transmit FIFO for the purpose of comparison). During message comparison, every possible combination and number of errors is detected. The presence of any errors in the received message causes data from the point of the error to be retransmitted to the network automatically.

There is no real-time system performance degradation if the received message is error free. But even if an error has been detected, the amount of time spent correcting it is minimal. This occurs for two reasons. First, unlike data processing networks that transmit large data streams and depend upon the receiving node to request retransmission; SCRAMNet detects and automatically corrects its own data transmissions. Second, because SCRAMNet has a much lower transport delay than other networks, it detects errors very quickly and need only retransmit a minimal amount of data. SCRAMNet's time to correct errors is measured in microseconds not seconds like typical data processing networks.

In addition to Platinum mode source-removal-and-compare error-correction by the sending node, every SCRAMNet node performs error checking on all messages received under both the SCRAMNet+ Platinum and Burst protocols. In Platinum mode, error detection takes place so that correction can ensue. In Burst mode, error detection takes place so that corrupted data can be removed from the network prior to corrupting SCRAMNet memory. Each receiving node detects errors by performing a parity bit check on every transmitted byte in a received packet. If a parity error is detected, the entire message packet is removed from the network. If a sending node is operating in Platinum mode and one of its messages is removed because of parity errors then the node will not receive it back within the Network Time-out period which will cause the message to be retransmitted automatically.

C. Message Packets

The SCRAMNet Network message packet has been fine tuned to provide optimum real-time network performance. This standard has been achieved by reducing message transport delay and by designing only network control bits deemed "absolutely necessary" into the message packet. For example, a destination node ID is not found in the SCRAMNet message packet for these design reasons:

- SCRAMNet is a “ring” network, meaning every message is received, resynchronized, and retransmitted. Since SCRAMNet is fast enough to place all received data into memory at full network speed, and since customers usually desire broadcast capability, there is no performance penalty for maintaining memory consistency around the ring. A node that does not need all data is not required to read all data.

- Flexible message routing is “expensive” for the network. The added message packet bits cause a pure network delay from clocking the extra bits on and off the network for every message. Also, each node requires more complex (and costly) hardware to make the routing decision.

SCRAMNet’s design balances the cost and performance of the network implementation against the actual needs of the real-time system integrator. Achieving this balance has resulted in a message packet consisting of 81 bits plus one start bit for a total length of 82 bits. The message packet itself is comprised of five fields - Source ID, Age, Control, Data Address and Data Value - described and shown below. Note that for every 8-bit byte of data in the message, there is a parity bit attached.

ID	AGE	CONTROL	ADDRESS	DATA VALUE
8+P	8+P	3	5+P 8+P 8+P	8+P 8+P 8+P 8+P

1. *Source ID*: This 8-bit field contains the ID of the originating node. Values range from zero to 255. Thus, there can be 256 nodes on the network ring. Node IDs do not have to be selected in sequential order. However, it is important to choose a unique ID for each node. A match between a node's ID and the ID field of a received message packet results in the removal of the message packet from the network.

2. *Age:* This 8-bit field incrementsby one as a message passes through each network node. If the age ever exceeds 255 (the maximum number of nodes possible on the network), the message is removed from the network. It also provides a count of the actual number of nodes on the ring.

3. *Control Bits:* This field contains three control bits. The first bit is reserved and the second bit is *INT* (signals an interupt message), and the third is *RTY* (signals a retry message).

4. *Data Address:* This 21-bit (A22-A2) field contains the relative SCRAMNet memory address. Bits A0 and A1 are always zero for a long-word boundary and are not included in the packet. The resulting 23-bit address provides up to 8MBytes of memory address space.

5. *Data Value:* For a 4-byte transfer the 32-bit field contains the data value of the word in SCRAMNet memory that is currently being updated around the ring. As previously discussed this packet can contain a “data value” portion that could vary in size up to 1024 bytes when using the Platumnum+ or Burst+ modes.

D. Shared Memory

The SCRAMNet Network’s shared-memory controller channels network writes to shared memory without incurring any local host bus overhead. Furthermore, read-write accesses from the host CPU are arbitrated, eliminating interference to ongoing network traffic. This approach enables network data communication to be totally transparent. In fact, not only does global network communication occur without direct CPU involvement, it occurs without impeding CPU performance in any way.

The power of the shared-memory controller is evident in the simplicity with which global communication is implemented. SCRAMNet shared memory is first installed into the host CPU's memory space. The system software engineer then configures system-wide global data as a contiguous data set, exactly as it is configured with a FORTRAN common or C data structure. Application program global data structures can then be mapped directly onto shared memory by linking the starting address of the global data set to the first address of SCRAMNet memory. For the application programmer, data structures mapped to shared memory are handled exactly like those mapped to local CPU memory. Then, whenever an assignment occurs to a global data set variable, SCRAMNet hardware automatically and transparently sends the updated variable out to the network.

SCRAMNet's shared-memory controller makes interprocessor communication both software *and* hardware transparent. In terms of software, any data written into SCRAMNet memory are automatically sent to the same SCRAMNet memory location of all network nodes. No specific network software is needed, which means software development and maintenance costs normally associated with specific network I/O code are drastically reduced! In terms of hardware, the shared-memory interface causes global network I/O to occur with normal CPU memory transactions adding no I/O-specific overhead to the host CPU. This frees the CPU for other processing.

E. Auxiliary Control RAM

Auxiliary Control RAM (ACR) is an important SCRAMNet Network control structure, offering a simplified approach to initializing software-transparent, real-time system interrupts and external triggers. The ACR accomplishes this by making five independent, user-selectable actions available during a write to a SCRAMNet shared-memory address. Because every 4-byte shared-memory word (on even 4-byte boundaries) has its own associated ACR bits, each shared-memory location, therefore, has its own associated action(s) that can occur when accessed by either the host or the network.

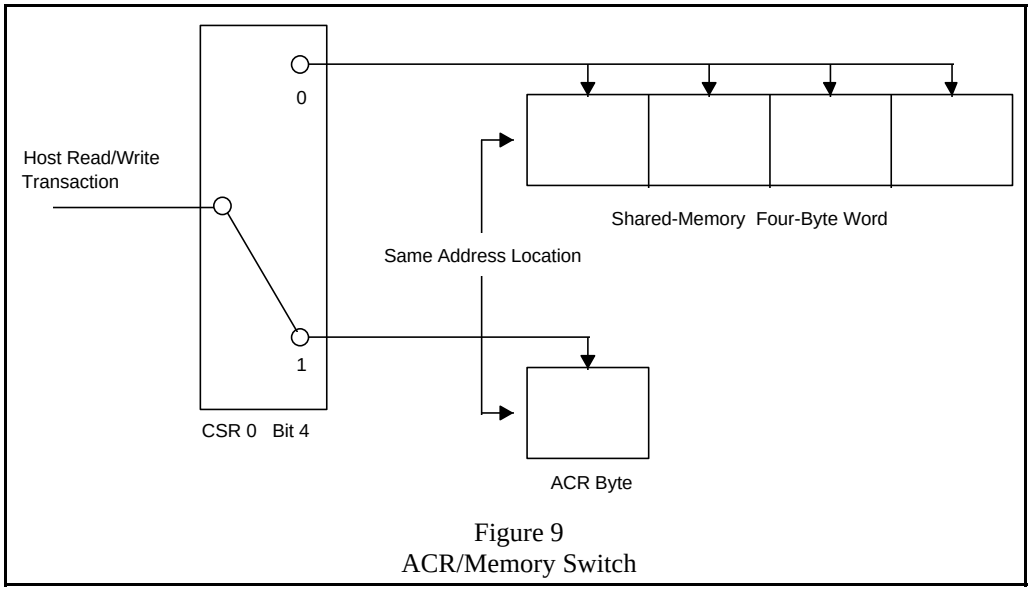


Figure 9
ACR/Memory Switch

In Figure 9, it is shown how the ACR/Memory Switch selects a host read/write to SCRAMNet memory or to ACR. Channeling is user controlled and may be toggled to the desired position by writing a bit into the appropriate SCRAMNet CSR. When ACR access is enabled, the ACR byte is available to be read or written and is located by referencing the address corresponding to the shared memory word to which you want the trigger or interrupt to apply. At this point shared memory is not accessible. The ACR byte defines what external trigger and/ or interrupt action(s) are to be taken whenever any byte of the SCRAMNet shared memory, 4-byte word is written or read. Currently, only the lower 5-bits of the ACR byte are defined.

ACR interrupt actions are bit 0, Receive Interrupt Enable (RIE), and bit 1, Transmit Interrupt Enable (TIE). TIE allows the network message packet to be transmitted as "interrupt enabled". RIE, on the other hand, causes an interrupt request to be generated on the host CPU backplane when the network receives an interrupt enabled packet. If these actions are disabled in ACR, no hardware interrupt action is taken, with one exception (see Section F.).

The SCRAMNet Network board can generate two types of output signals that are based on the ACR. Both of these consist of a 26.64 ns pulse that is a TTL-level compatible, non-terminated output. The external triggers defined by ACR are:

•Bit 2, external trigger 1 (the host has accessed this SCRAMNet memory location),
•Bit 3, external trigger 2 (the network has written to this SCRAMNet memory location).

Another important feature of the ACR is to enable the HIPRO mode of operation. Setting Bit 4 enables HIPRO which performs certain actions designed to improve SCRAMNet and Host performance as described below.

HIPRO WRITE

The SRAMNet+ network message is based on 32-bit longword data. If a host processor is only capable of 8-bit or 16-bit transactions, then SCRAMNet+ would issue a 32-bit network write even though the actual data content would only be 8-bits or 16-bits. This would effectively quarter or halve the network bandwidth.

The HIPRO mode was created to provide an efficient means to transmit two 16-bit data transactions or four 8-bit transactions as one 32-bit network WRITE. For a 16-bit write, the first of the two 16-bit WRITES will be to SCRAMNet memory but will be prevented from going on the network. The second 16-bit WRITE to memory will trigger the WRITE of a 32-bit data packet to the network. For an 8-bit write, the first three of four WRITES will be to memory but will be prevented from going onto the network. The fourth WRITE to memory will trigger the WRITE of the 32-bit packet to the network. The two 16-bit transactions or four 8-bit transactions must be within the longword boundary.

Once the ACR has been initialized, the ACR-enable bit in the CSR should be set to zero so that shared-memory returns to its normal configuration. ACR actions are still in effect, but the ACR bytes can no longer be accessed by the host while the ACR-enable bit is zero.

F. Flexible Interrupt Processing

The SCRAMNet Network's ability to generate interrupts to its host processor is the kind of feature system integrators need to efficiently manage devices, synchronize events, and schedule tasks on distributed real-time computing systems. This flexible interrupt capability can control a camera, an image projector, or a simulator motion base. It can also synchronize imagery with audio and motion. With usefulness that extends to many other practical control and

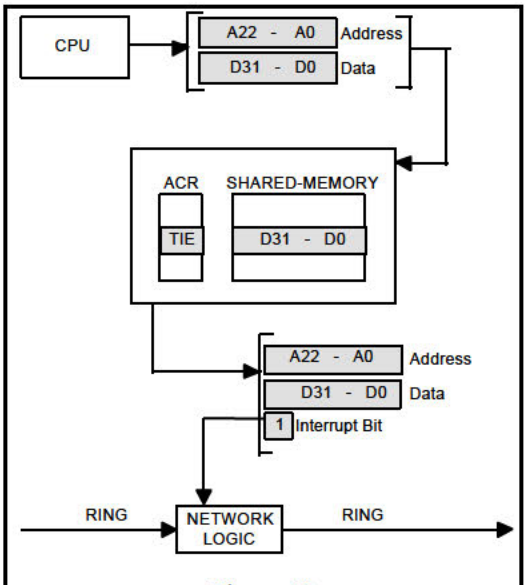


Figure 10
Transmit Interrupt Processing

synchronization needs, SCRAMNet's interrupt processing scheme is both powerful and easy to implement.

Much of the power in SCRAMNet's interrupt structure stems from the fact that interrupt words are actually normal data words, located in memory, which are designated at initialization to cause interrupts. Tagging these words for interrupt is simply a matter of setting the appropriate ACR at the corresponding SCRAMNet addresses. Once this is done, SCRAMNet handles interrupts like any other data on the network (Figure 10).

Whenever a write occurs to a SCRAMNet memory address, network control logic automatically checks ACR memory at the same SCRAMNet offset address to see if the Transmit Interrupt Enable bit is set. If it is, the word is transmitted onto the network like any other data word. Except that the interrupt bit is set in the network protocol message, indicating to all nodes receiving the word of the intent to enable interrupt for this data word.

In general, on the receiving side (Figure 11), network control logic checks for the presence of the interrupt bit in the message. If the bit is set, network control logic checks for the presence of the Receive Interrupt Bit at the appropriate SCRAMNet address in ACR memory. This check is performed to see if the host processor wants to receive an interrupt by a network write to that location. If so, the SCRAMNet address for the word is deposited into the interrupt FIFO, the word is written into memory per the normal fashion, and an interrupt is issued to the host processor with a predetermined interrupt vector and level. The exception to this method is that a CSR bit can be set to make all shared-memory locations interruptable locations regardless of the ACR bit, so that all incoming network writes, with TIE, generate a host interrupt.

Because interrupts are such a key facility for synchronization and control within real-time systems, SCRAMNet offers a very flexible feature for selective interrupt processing. By setting a receive interrupt override bit in the CSR's a node can be instructed to always process each incoming message as though its interrupt bit was set. This allows the application on the receiving node to determine which memory locations to interrupt based on the status of the RIE bit for each memory location without depending on the transmitting node to set the interrupt bit in the message packet. This feature could be useful when a receiving host wants to insure a write to a specific memory word always causes an interrupt. If the receive interrupt override bit is not set, then the interrupt bit in the packet must be received before an interrupt can be generated. This could allow only certain processors to actually interrupt other processors on a single memory location.

The SCRAMNet Network's interrupt system does not lose data even though it is ultimately the user's responsibility to read the interrupt FIFO (the user does this to determine which memory word causes an interrupt in preparation for taking the appropriate action). Reading the interrupt FIFO causes it to "pop". However, interrupt latency and host processing speed may result in the generation of multiple interrupts while the host is processing an interrupt. Therefore, SCRAMNet is equipped with an interrupt FIFO capable of queuing up to 1,024 interrupts. Each interrupt is queued in the interrupt FIFO with the SCRAMNet relative address of the memory

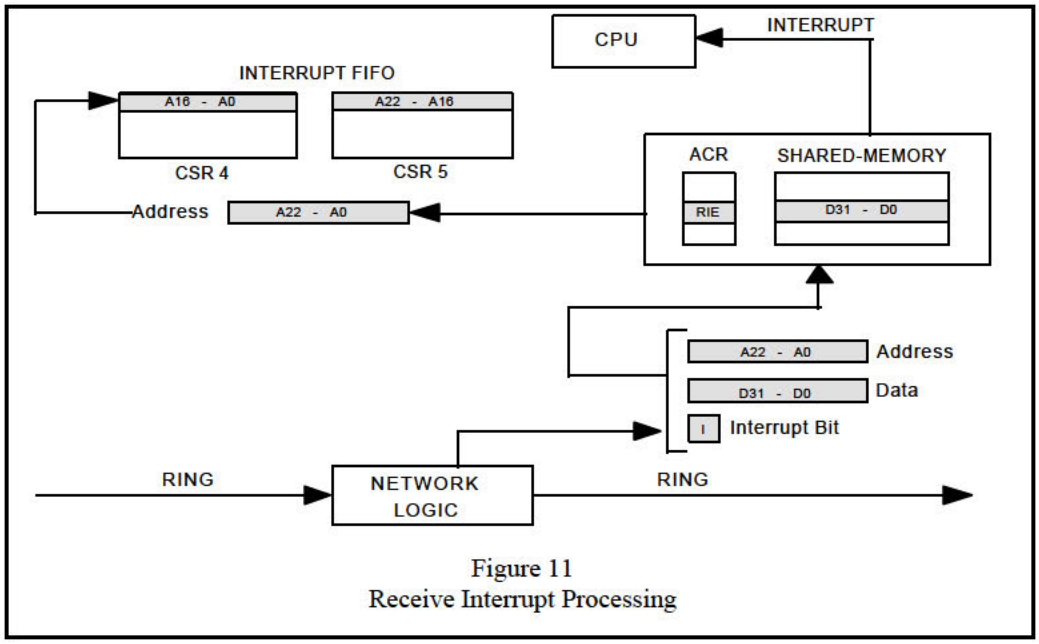


Figure 11
Receive Interrupt Processing

word causing the interrupt.

Entry into the interrupt FIFO by the Host disables the interrupt hardware until the host CPU has emptied the FIFO and rearmed the interrupt hardware. This allows the host CPU to batch process a burst of interrupt events very efficiently. Interrupt addresses can still enter the FIFO as they are received during this time.

G. Real-Time Data Filter

The *data filter* is a powerful real-time capability unique to the SCRAMNet Network with a usefulness that is best illustrated using an F-16 avionics support facility as an example. Like many other cyclical real-time systems, this F-16 facility schedules algorithms and tasks at fixed, clock-driven frame rates. For example, the facility schedules the same series of tasks every 20 milliseconds. This produces a simple, efficient distributed executive with all CPUs synchronized by a common clock.

However, system inefficiencies occur in the data flow between the system's distributed CPUs. The F-16 facility executes models and tasks based on a clock tick, not on need. In fact, few of its algorithms actually produce new data from frame to frame. Performance tests show that approximately 75 percent of the data is unchanged between adjacent timing frames. If a tool were available for filtering out redundant data transmissions, this system could realize an effective network bandwidth *more than 4 times* its actual bandwidth. Fortunately, just such a tool - the data filter - is available in the SCRAMNet Network.

The operational theory behind SCRAMNet's data filter is simple. When the host CPU performs a write cycle to any specific SCRAMNet memory location (Figure 12), SCRAMNet's data filter immediately performs a fast read on that location and stores the contents in a comparator. If the new value does not equal the old value, then the new value is written into SCRAMNet memory and a network transaction occurs. However, if the new value is exactly the same as the old value, *only a write to SCRAMNet memory takes place*; no network transaction is generated. By eliminating the transmission of outputs that do not change from one computational cycle to another, SCRAMNet's data filter can eliminate a great deal of network traffic for a much broader effective network bandwidth.

Following are six points to keep in mind about the SCRAMNet data filter feature:

1. The data filter eliminates unnecessary data transmissions on the network
2. The data filter can be selected for all of SCRAMNet memory or only memory beyond the first 4K bytes.
3. Data filter operations are transparent to the host CPU and do not degrade host or network performance
4. If even one bit changes in a 32-bit longword memory location, that value is not filtered
5. The data filter is automatically disabled for any memory location targeted to interrupt another node (even if the value is redundant, it is still transmitted onto the network to interrupt other nodes)
6. Data filter usage is optional, dynamically user selectable through the CSRs

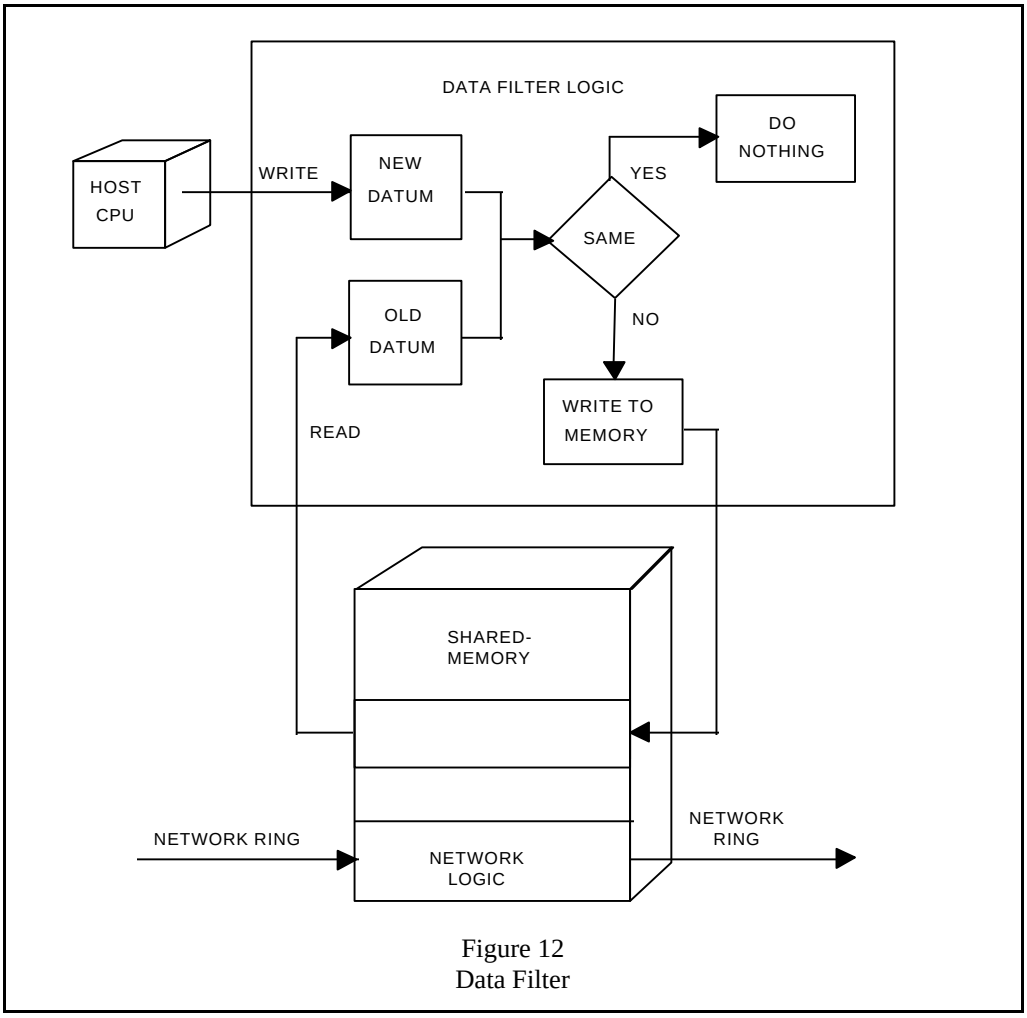


Figure 12
Data Filter

H. Real-Time Byte and Shortword Swapping

A strength of the SCRAMNet Network is its ability to successfully link together heterogeneous computers into one "virtual" computing system. Incompatible binary representations of integer data between different manufacturers' CPUs is a major concern in such systems and typically why system designers avoid developing them. This concern is addressed with SCRAMNet's byte swapper, an optionally-implemented feature that helps resolve data format incompatibilities so that different manufacturers' computers can work together efficiently.

The two commonly used binary representations of integer data are:

1. *Big Endian* – the *most* significant byte of data is in the lowest address in memory
2. *Little Endian* – the *least* significant byte of data is in the lowest address in memory

The fact that most real-time applications have a database comprised of a mix of integer sets – arranged and accessed in memory in units of bytes (8-bit units), shortwords (16-bit units), and longwords (32-bit units) – further complicates Big Endian/Little Endian incompatibility. Each type of memory access, byte or shortword, requires a different and unique swapping transaction to convert from one representation to the other.

The SCRAMNet Network is designed as a Big Endian network. When working with Big Endian platforms such as VME, all data is transferred to the network in the proper swap mode and no extra consideration need to given to byte swapping. Similarly, if all of the nodes on the network are Little Endian machines, i.e. PCI or PMC, the byte swapping is automatically handled by each node, which swaps the data transferred to and from the network accordingly.

In a heterogeneous computing environment, all Big Endian platforms simply connect as usual, since SCRAMNet is designed as a Big Endian network. The PCI/PMC platforms are required to handle any necessary byte swapping for network compliance. Byte swapping on SCRAMNet PCI is handled by the PCI bus interface chip. This chip provides three modes of operation known as Long, Word and Byte modes. The SCRAMNet Application Programming Interface (API) provides access to these transaction modes. When writing 32-bit data to SCRAMNet, setting the transaction mode to Long_mode writes the data correctly to the Big Endian network. Similarly, Word_mode and Byte_mode set the appropriate transaction mode for 16-bit and 8-bit data.

Because changing transaction modes on the fly can contribute to performance degradation and in some remote instances, data corruption, we recommend the following three byte-swapping solutions in order of preference.

1. *Program using 32-bit variables only* – When defining variables in 'C' programs, the use of 32-bit variables exclusively is the most desirable solution. The transaction mode can be set to Long_mode at program initialization and all variables will be transparently put into Big Endian format. This approach is by far the simplest of the three and does not impact system performance in any way while maintaining proper byte ordering.
2. *Use software byte swapping* – Many applications are already written using variable sized data types. If variable size byte swapping is required, it is recommended to set the transaction mode to Long_mode and to implement simple byte swap macros in software for 16 and 8-bit addressing. Using this method, the hardware handles all 32-bit transactions and gives the application programmer swapping control of the smaller data sizes. When compared to changing transaction modes of the PCI bus interface chip, this method requires very little processor overhead.

3. *Use SCRAMNet transaction modes* – If neither of the first two options are possible, the application programmer can change transaction modes as required by the data size. It is recommended that the proper transaction mode be set and then like sizes of reads and writes, i.e. all 8-bit, all 16-bit and all 32-bit, be performed together. This will minimize the number of accesses to the PCI bus and have the least impact on system performance.

Byte 0	Byte 1	Byte 2	Byte 3
Byte 3	Byte 2	Byte 1	Byte 0

Big Endian Number 01234567
Stored as:

Byte	**Data**
0	01
1	23
2	45
3	67

Little Endian Number 01234567
Stored as:

Byte	**Data**
3	67
2	45
1	23
0	01

SCRAMNet+ is by design a Big Endian network, so the conversion for Little Endian nodes is as follows:

1. SWAP the 16-bit shortwords

Byte	**Data**		**Byte**	**Data**
3	67	1	23	
2	45	becomes	0	01
1	23	3	67	
0	01	2	45	

2. SWAP the bytes within each shortword

Byte	**Data**		**Byte**	**Data**
1	23	0	01	
0	01	becomes	1	23
3	67	2	45	
2	45	3	67	

Figure 13
Integer Byte Swapper

I. General Purpose Counter/Timer

SCRAMNet+ provides a 16-bit counter/timer, that can be programmed by Control and Status registers, to perform specific counting and timing functions. Table 2 shows the modes of operation of the counter/timer and the description of each mode of operation.

Table 2
General Purpose Counter/Timer Modes

Mode	Description
Count Errors	Each error detected in CSR1 will increment the counter by 1.
Count Trigger 1 and 2	Each time a trigger event occurs the counter will increment.
Ring Time	Set this mode and clear the counter. The counter will begin when the next message is transmitted and stop counting when any message generated by this node is received.
Network Packets	Count incoming network packets.
Free Run @ 26.66 ns	Increment counter using internal 37.5 MHz clock. Counter will roll over every 1.78 ms.
Free Run @ 1.706 us with trigger 2 to CLEAR	Increment counter using the 585.9 KHz clock. Counter will roll over every 111.8 ms. Assertion of trigger 2 will clear the counter.

The counter/timer has a rollover/reset function which can generate an interrupt to the host system based upon the occurrence of specific conditions within the counter/timer. The free running counter function can be particularly useful to the applications programmer when used with the presetting capability of the counter/timer output register. Lets say that an interrupt needs to be generated every 100 ms. One of the free running counters increments at 1.706 us. So by dividing 100 ms by 1.706 us you get 58,616. When you subtract this from 65,536 (the counter rollover limit) you get a value of 6919. So if you preset the counter output register with a value of 6919, then 100 ms will have elapsed when you reach the rollover point of 65,536 and an interrupt will be generated to the host.

The global timer mode clocks with a resolution of 1.706 us and resets on an external trigger 2 event (See paragraph L. External Triggers). A common shared-memory location in each node on the network may be set via the ACR to generate an external trigger 2 when that memory location is written to. Therefore, a memory write from a single node to that memory location on the network can simultaneously reset all the global timers in the ring. Thereby providing a network-wide timer capability where all nodes are synchronized within the ring time of the network. This could be used to tag time-critical data or to measure and compare the completion time of various tasks within a distributed real-time system.

J. EEPROM Initialization of Control and Status Registers

Power-up initialization of the SCRAMNet+ CSR settings is accomplished via an EEPROM. The EEPROM initialization process is included to allow for more flexibility in the power-up settings of the CSR's. In addition, previous versions of the SCRAMNet cards used jumpers and hard switches, which required physical intervention and node power down to change the initialization settings. With the EEPROM, the SCRAMNet card can be software controlled by an user application program.

As part of the Core Software Package provided with each SCRAMNet card, the EEPROM initialization program is included to assist the user with the power-up setting of the Control and Status Registers. While the CSR's have been preset (just prior to shipping) with standard defaults the user will no doubt want to customize the CSR's initial state to their own unique application needs.

K. Network Memory Virtual Paging

The virtual paging feature offers increased flexibility in the use of SCRAMNet memory. All SCRAMNet+ nodes in the network use the same 8 MByte shared-memory address space. Virtual paging allows a node with less than 8 MBytes to move their memory "windows" throughout the SCRAMNet+ physical 8 MByte shared-memory space on the network. Virtual paging is not between the host and the SCRAMNet memory. When the host writes to address 0 in shared memory, it will always be stored at address 0 on shared memory. The same applies for a host read. Virtual paging only changes the network address.

If a node has 4 MBytes of SCRAMNet+ shared memory, it can be paged into the upper 4 MBytes or the lower 4 MBytes of the 8 MByte network shared-memory map. If it is paged into the lower 4 MBytes of the shared-memory map, it would appear as if virtual paging were disabled. The network address would be identical to the shared-memory address.

If the 4 MByte block were paged into the upper 4 MBytes of the shared-memory map, there will no longer be a one-to-one relationship between shared-memory address and network

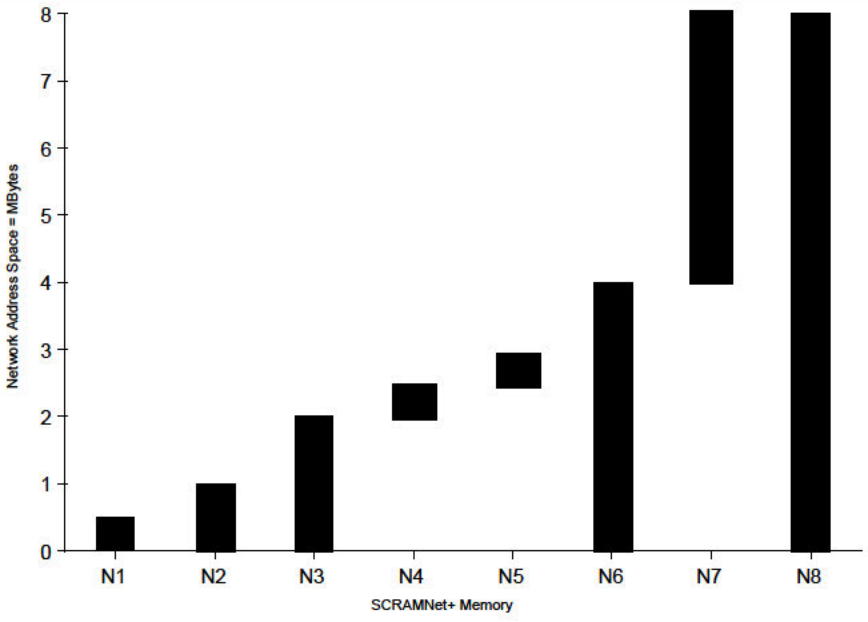


Figure 14
Network Memory Paging

memory address. That is, a host write to a SCRAMNet+ memory address of 2 MByte would result in a network address of 6 MByte. However, the data is stored in shared memory at address 2 MByte. Please note: The host always sees the base address of SCRAMNet memory as 0.

This translation is bi-directional as well. An incoming network message with a network address of 6 MByte, will be written to SCRAMNet+ shared memory at address 2 MByte, assuming that nodes memory is paged into the upper 4 MBytes. If a SCRAMNet+ nodes shared memory is virtually paged into the upper 4 MBytes WRITE accesses by the network to the lower 4 MBytes of shared memory will be ignored since there is no memory there.

This feature permits two different nodes with 4 MBytes of memory to be paged into different halves of the network shared memory map and not be able to see the other node's network traffic (See Figure 14). It also permits a 4 KByte board to be located anywhere in the network shared- memory map (on a 4 KByte boundary) and it also allows a 4 KByte node to page itself to talk to any shared-memory location on another node. There are many application based motives for this mapping flexibility.

L. External Triggers

As it was mentioned in the section on the ACR, two types of external triggers are provided by the SCRAMNet+ network. All of the external triggers will occur only if the ACR memory location has been configured to enable triggers. Activating the triggers for any memory location will cause an external trigger to be generated when the memory location is accessed in the prescribed manner. All triggers generate a 26.64ns TTL level compatible, non-terminated, output. Table 3 describes the external trigger definitions.

Trigers 1 and 2 are activated by SCRAMNet+ memory access. Trigger 1 will be generated for any host access to SCRAMNet+ memory. Trigger 2 will be generated by a network write to the SCRAMNet+ memory.

Table 3
External Trigger Definitions

Trigger 1	Host Read/Write (ACR bit 2 enables)
Trigger 2	Network Write (ACR bit 3 enables)

This page intentionally left blank.

V. SCRAMNet+ Software Tools

A. Core Software Package

Using the SCRAMNet network results in *major cost and time savings* for initial software development as well as for subsequent software maintenance and upgrade efforts. Normally, one of the more costly, difficult tasks associated with production of a distributed real-time system is the development of software that successfully coordinates the flow of global I/O data. SCRAMNet simplifies this task by making global I/O communications *software transparent*. This means that in real-time operation, global I/O occurs when no network-specific software has been executed. Four software products are available to assist the SCRAMNet+ customer in implementing and using the SCRAMNet+ hardware features. They are:

- SCRAMNet+ hardware diagnostic and maintenance program (DIAG),
- SCRAMNet+ EEPROM initialization software,
- SCRAMNet+ drivers for handling real-time interrupts,
- SCRAMNet+ register/memory mapping routines.

Hardware Diagnostic & Maintenance Software

SCRAMNet hardware diagnostic and maintenance (DIAG) software is normally the first SCRAMNet software tool used after SCRAMNet hardware has been installed. DIAG software is a collection of “C” or FORTRAN routines designed to functionally test every SCRAMNet feature in all of its operational modes. The menu-driven DIAG software is easy to install, even easier to execute and clearly documented. Following is an overview of how it works.

Once the DIAG software is activated, its main menu displays three submenu selections used to execute the programs that test the functionality of a SCRAMNet Network node from three different aspects. Also included at the main menu level are selections to allow for the running of all tests or selected tests, checking node status, saving test results for future analysis and recalling prior test results.

The first submenu lists *Memory tests* that evaluate SCRAMNet main memory and ACR. These memory tests view SCRAMNet as an add-on memory board to the host processor.

The second DIAG submenu lists tests that systematically check every functional feature supplied on the SCRAMNet network: data filtering, byte swapping, interrupts, etc. This submenu is called the *Loopback Test* submenu because using it invokes on-board diagnostic modes that allow network transmitters to be directly coupled, or "looped back," to the receivers on the same board. The concept behind this arrangement is to test each node in the network independently in order to isolate any problems found to a single source.

The third DIAG submenu lists tests that exercise each SCRAMNet node in concert with other nodes on the network. The submenu is called the *Network Test* submenu, and it is based on the concept that a SCRAMNet node does not normally function by itself but that it is always part of a larger network system (*i.e.*, two or more nodes).

Two important points to note about DIAG software*: It comes standard with the complete DIAG software source code, and it includes an operating system compatible device driver object*. The SCRAMNet Network is easy to implement - but by using the DIAG source code as an example of how to set up, initialize and control every feature of the SCRAMNet Network, a real-time architect can have a SCRAMNet network "up and running" *in a matter of hours rather than days*, as is typically required to install and program other networks. Standard SCRAMNet features such as these are evidence of SYSTRAN's commitment to helping customers successfully develop distributed real-time systems.

EEPROM Initialization Software

The EEPROM Initialization Program (EPI) is a SCRAMNet+ utility designed to simplify configuration of the network node initialization state upon power-up. The EPI is not necessary for the usage of the SCRAMNet node in the network. The EPI program will store the control registers configuration in the serial EEPROM, which then initializes the node on power-up. Node configuration in the SCRAMNet Classic product was a fixed initialization state arrived via the usage of jumpers and switches. The EEPROM initialization is only a start-up configuration, the user can still change the nodes configuration state dynamically during use with SCRAMNet+, but now the start-up state can be dynamically changed without powering down the board and changing physical switches and jumpers.

EPI can be run when the board is installed to set the desired start-up state or it can be run when the node is on-line with the effects of any modification to the power-up state occurring during the next power-up or system reset. EPI is completely menu driven and contains a context sensitive help feature to display useful information on the selected item.

Software Interface Library

The SCRAMNet+ Software Interface Library is a series of "C" or FORTRAN routines. These routines perform a number of general utility operations. The subroutines provided can be accessed from any user written high-level language program following proper link to the library module. The various subroutines are divided into four general categories: Setup, Data Flow, Interrupt and General.

Set-up Routines

The routines perform very primary functions necessary for the nodes' correct operation and use. They initially configure the node (based upon data derived from the configuration file) by mapping all the registers and memory and obtaining a pointer to the SCRAMNet memory. They also provide a routine that can be called to validate that the registers and memory have been properly setup.

Data Flow

These routines turn on the byte swapping feature, enable data filtering, insert the node into the ring, and put the node into a write-me-last mode.

Interrupt Setup Routines

These routines enable or disable access to the ACR, select and set a specific type of interrupt to be enabled, and select an address range within the shared memory that will be used as interruptable memory locations.

General Routines

These routines as denoted by their classification provide various non-specific functions regarding the majority of an applications' interface for controlling and using the SCRAMNet node. Some of the key routines are:

- Set Virtual Paging
- Read Interrupt FIFO CSR's
- Set Network Protocol
- Write CSR Values
- Set Network Timeout
- Obtain Node ID
- Set General Purpose Counter
- Read CSR Values

As with the DIAG software, the Software Interface Library comes standard with the source code included, providing useful examples of how to implement SCRAMNet's many powerful real-time network features, thus providing the application programmer a basis for extending and/or creating his/her own routines for functions Systran has not provided.

Operating Systems and Languages

Traditionally, software development environments have not been considered adequate run-time situations for real-time applications. This has mainly been because the internal processes of the typical operating system (OS) have been non-preemptable, and non-preemptable processes result in indeterminate program context switch times. To the real-time architect, this translates into an inability to guarantee the needed response times to real-time events. For this reason, real-time architects have learned to develop programs under one OS environment and execute them under another special real-time OS environment.

However, recent advances in hardware technology and software standards have put the development and operational environments of real-time systems in a state of flux. Operating systems that are unique to the real-time environment still exist, but more and more the movement is toward developing *and* running applications in real-time under the same OS environment. For this reason, SCRAMNet DIAG software and Software Interface Library have been ported to a wide variety of OS environments. These environments include but are not limited to:

- UNIX (most popular versions)
- Open VMS for DEC systems
- VxWorks
- LynxOS
- MS-DOS with DOS extenders
- Solaris
- IRIX
- Harris CX/UX
- P/RT
- Windows NT®/Windows® 95

If SCRAMNet Network software has not been ported to a particular OS environment, it can usually be rehosted to the environment using the source code provided.

B. Simplistic Approach to Node Initialization

SYSTRAN invests a great deal of effort into making SCRAMNet network nodes easy to initialize and use. SCRAMNet nodes are initialized through seventeen clearly defined and documented Control and Status Registers (CSRs). Following is a list of control features:

- Software setting of the SCRAMNet interrupt vector
- The reading of several message and node status indicators
- Network mode control of data filtering and node network linkage control
- Resetting the Rx, Tx, and the Rx/Tx FIFO's
- Turning on/off error correction
- Enabling variable length records (Plus mode of operation)
- Developing Interrupt & error masks to fine tune error & interrupt control
- Setting the memory address and address size

Typically, the application program memory maps SCRAMNet CSRs. This allows application programs to write SCRAMNet CSRs just like any other memory. Once set, the Control Registers are generally left unchanged during real-time operation. An alternative approach is to use the SCRAMNet Core Software Package. This package has well documented, easily implemented, user callable routines to memory map the CSRs as well as to initialize every SCRAMNet mode and function.

C. Coordinating Tasks and Data Updates

One of the most challenging tasks to confront a systems integrator is coordinating the activities of distributed processes that are sharing data. The flexibility of SCRAMNet's replicated shared-memory design provides several options for solving system task and data synchronization problems. These options, or synchronization models, are briefly discussed below.

Synchronized Frame Model

The *synchronized frame model* provides tight synchronization of tasks running on distributed processors. Implementing this model involves issuing a SCRAMNet interrupt at the beginning of every time frame to all nodes on the ring. This synchronizes the execution of distributed tasks so that they start simultaneously. In this model, one processor, with a programmable real-time clock, writes to the synch word based on a clock interrupt. This means that the clock interrupt service routine has to be written to a synch word that has been initialized in order to cause a SCRAMNet interrupt to all the pertinent distributed processors. A similar but even more robust solution is to have different synch words for different frame rates (*e.g.*, a synch50 interrupt word for the 50-hertz rate group of processes and a synch32 interrupt word for the 32-hertz rate group of processes).

Traditionally, this model handles the data flow between distributed processes by having every process do global inputs at the beginning of the time frame and outputs near the end of the frame. Of course, all global outputs must be seen as global inputs to the receiving processes before the next synchronizing interrupt marking the beginning of the next frame.

Data Driven Model

The *data driven model of synchronization* allows distributed tasks to be triggered for execution based on receipt of input data. (In this model, the last global output from every process should be initialized within SCRAMNet to cause an interrupt in the receiving process.) The synchronized frame model does not schedule any network traffic for most of the time frame and then inundates the network with traffic for all processes, simultaneously dumping data at the end of the time frame. The data driven model leverages network data traffic over the length of the frame. This makes the data driven model more robust than the synchronized frame model.

Asynchronous Model

The *asynchronous model* allows distributed tasks to run as they normally do on their own processors. Data flow between tasks can be handled with a simple counter system, saving the system overhead incurred when interrupts are used. In this model, the burden for receiving a contiguous set of data is on the receiving process. The *transmitting* process increments a counter at the beginning of the dataset, which is, itself, the first word of the dataset. The transmitting process then proceeds to write all the output data in a sequential manner. The last word of the dataset is another counter which the transmitting process also increments to reflect the same value as the counter at the beginning of the dataset. The receiving process checks the last counter first to see if it has the same value as the first counter. If the value is the same, the receiving process starts to read all input values. At the end, the process checks the first dataset counter again. If the value is still the same, the transmitting process has not re-entered the dataset. If the value is different, the receiving process tries again.

This page intentionally left blank.

VI. APPENDICES

A. Host Access Timing

Throughout the document we have referred to the throughput of the network and the node latency times which gives the user a realistic estimate of what they could expect for overall system performance. The last key performance factor is the timing consideration for reads and writes from the host computer to the SCRAMNet shared memory. Each host interface (VME, PCI, ISA, etc.) has specific procedures for acquiring the bus and for issuing a read or write and has its own specific timing consideration for these actions. The time it takes for each of these actions can be obtained from the reference documents describing these buses. Since the SCRAMNet actions for getting data to and from memory are consistent for all bus types we have provided this timing below in Figure 15.

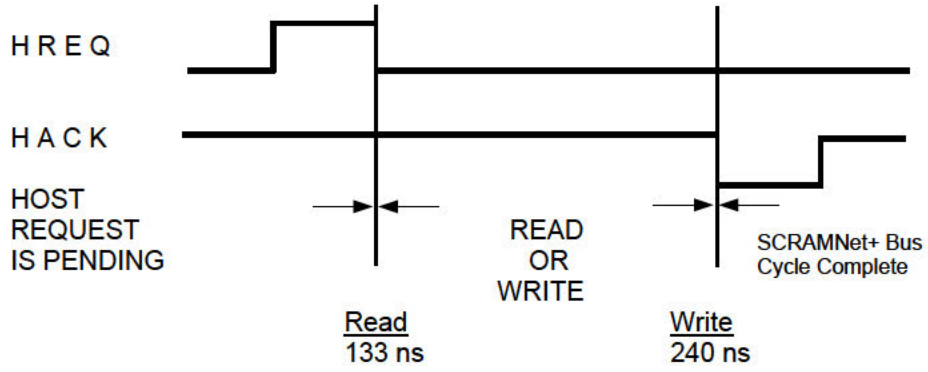


Figure 15
Memory Access Timing

Since the shared memory is controlled by a dual-port memory controller there are times when the memory may not be accessible because the memory controller is "busy" handling a "network write" to the shared memory. This would only effect the host transactions to memory when the host is attempting back-to-back reads or writes to shared memory. As an example of what the very worst case timing could be for back-to-back read or write transactions see Table 4. It should be noted that these times are theoretical since we have yet to discover a network in full saturation that is coupled with a host fast enough to create this situation.

Table 4
Worst Case Host Access Timing

Access Time	Cycle Time	Access Type	Cycle Time
Host Read #1	133 ns	Host Write #1	240 ns
2 Network Writes	+266 ns	3 Network Writes	+399 ns
Host Read #2	+133 ns	Host Write #2	+240 ns
Total Back-to-Back Reads	532 ns	Total Back-to-Back Writes	879 ns

B. Fiber Optic Specifications

All powers given in dB referenced to 1 micro watt and are average, not peak-to-peak.

Standard Link Fiber Optics Specifications (Cable length less than 300m)

Fiber optic transmitter:	Hewlett-Packard HFBR1414TB (12 dB to 19 dB into 62.5/125)
Fiber optic receiver:	Hewlett-Packard HFBR2416TB
* Power budget 12 dB (with 62.5/125mm fiber)	
Connectors:	Receiver: ST Transmitter: ST
Cable:	The cable must use a multimode fiber with a core/cladding diameter of 62.5/125 mm. Other sizes may be used but must be done with a working knowledge of fiber optics.

Note: Fibre Skew maximum allowable delay shown between fiber pairs = 1ns.

The numbers 62.5 and 125 of the 62.5/125 cable specification refer to the diameter of the cable in microns. 62.5 microns is the diameter of the optical fiber while 125 microns is the diameter of the cladding around the core. The core/cladding are extruded as one.

Long Link Fiber Optics Specifications (Cable length less than 2Km)

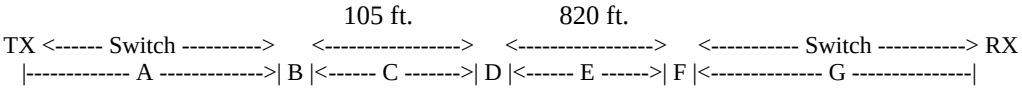
Fiber optic transmitter:	Hewlett-Packard HFBR1312T (10.5 dB to 15.5 dB into 62.5/125)
Fiber optic receiver: Hewlett-Packard HFBR2316T	
* Power budget 10.5 dB (with 62.5/125 mm fiber)	
Connectors:	Receiver: ST Transmitter: ST
Cable:	The cable must use a multimode fiber with a core/cladding diameter of 62.5/125 mm or a 50.0/125 mm cable can be used if greater distance (> 1000m) is required or many nodes are in the ring.

Note: Fibre Skew maximum allowable delay shown between fiber pairs = 1 ns.

Design Example:

Two nodes are connected 925 feet apart by a two-piece connected fiber optic cable and fiber optic bypass switches installed at each end (two pieces of cable are used for example purposes only). Using one 925-foot cable eliminates the 1.0 dB loss for the connector.

1. Check light-power budget:



CodeLink Element	Attenuation (Power Loss)
ABypass Switch	2.00
B Connector	1.00
C105 ft. (0.032 Km)	0.13
D Connector	1.00
E820 ft. (0.25 Km)	1.00
F Connector	1.00
GBypass Switch	2.00
	8.13 dB Total Attenuation

8.13 dB power loss is acceptable since it is less than the 12.0 dB light-power budget.

2. Check bandwidth-length budget:

0.28 km x 150 MHz = 42 MHz-Km required
(acceptable since 42 MHz is less than the 50 MHz-km maximum)

The design example is based on average data and is useful as a guideline. It should be noted that the only sure method of determining these values is to assemble the system and then measure the losses with an optical power meter.

C. Bypass Options

Electrical and optical node bypass options - the ability to have data circulating the ring pass around, or “bypass”, a node - give the SCRAMNet Network added versatility in the form of robust fault tolerance. It also provides the ability to operate a network node in standalone mode, looped-back on itself. A standalone node looped-back on itself allows software and hardware development to occur at the node without interfering with the rest of the ring network operations.

Figure 16 illustrates the operation of a wire loopback. Not only is data from the network routed around the node, with the network node’s transmitter and receiver acting as a signal repeater, but data from the node is routed back into itself. The major difference between a wire and an optical loopback is that the optical loopback is outside the node and involves the node’s optical transmitter and receiver. The wire loopback, on the other hand, uses the node's own electronics, but not the transmitter and receiver.

SCRAMNet’s optical loopback capability is accomplished through the use of the Fiber Optic Bypass Switch. The switch is connected to the auxiliary connector on the SCRAMNet node. An application program can send control information to dynamically determine the actions of the switch or the switch can ascertain the power state of the node and switch in or out of bypass or include state. The power sensing capability allows the switch to bypass a failed or powered-down node thereby maintaining the integrity of the ring. Though the switching time is fast (20ms), messages could be corrupted but the error correction capability insures that no messages are lost.

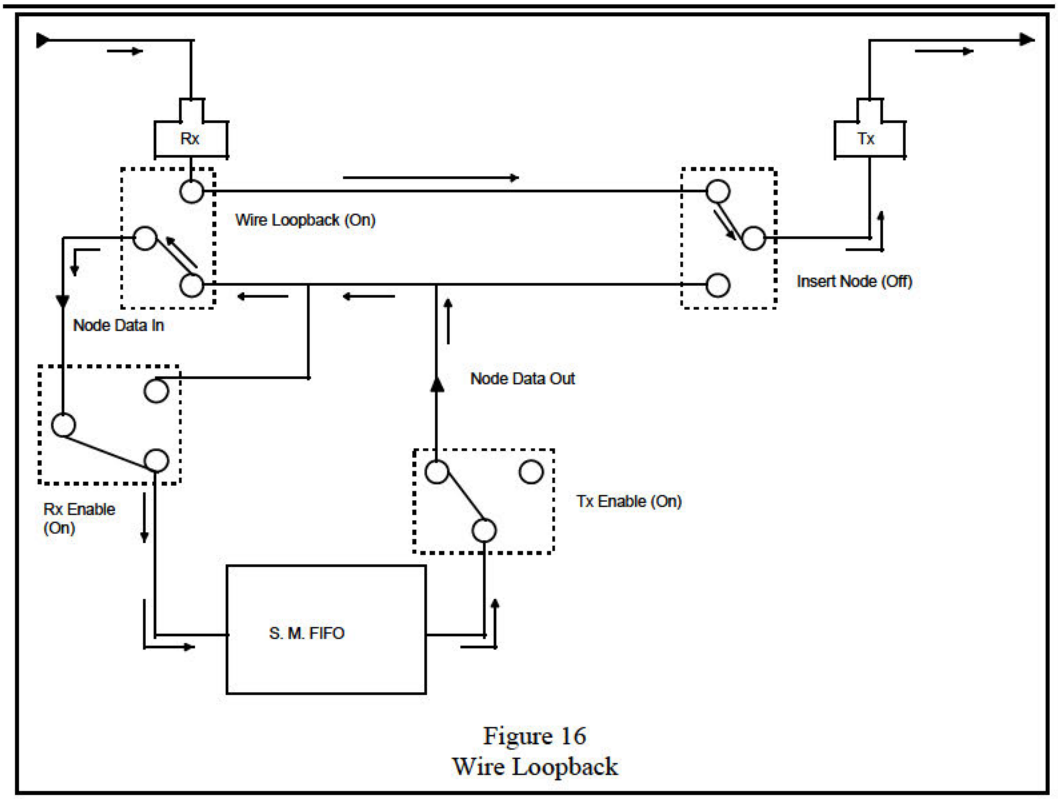


Figure 16
Wire Loopback

D. Form-Factor Adapters

SCRAMNet Network nodes are designed and packaged to fit directly into the backplane of most popular host processors. However, a great many physical footprints for backplanes are used by CPU vendors - a variety that could create difficulties installing SCRAMNet in a specific backplane. For example, a SCRAMNet VME node might have to fit into a VME6U backplane slot on one machine and a VME9U backplane slot on another.

SYSTRAN Corp. has solved such installation difficulties using form-factor adapters. Under the form-factor adapter approach, SCRAMNet is designed to fit into the backplanes with the smallest physical footprint (VME3U and VME6U for VME backplanes). Form-factor adapters are then used to conform SCRAMNet to the larger formats such as the VME9U.

SYSTRAN Corp. constantly maintains a variety of the most commonly used form-factor adaptors to facilitate SCRAMNet installations. Form-factor adaptors are an easy, elegant and cost-effective way of solving potential SCRAMNet Network installation problems.

E. Combining, Repeating and Isolating Networks

Often the real-time network designer needs configuration control over the network topology and computing resources to meet the changing needs of their real-time operation. The SCRAMNet Quad Switch is designed to provide the network administrator the configuration flexibility they need. The Quad Switch was originally conceived to allow local clusters of up to four SCRAMNet nodes to be switched in or out of a primary SCRAMNet ring independently and dynamically (Figure 17). It is equally useful when a critical real-time resource must be shared and easily re-allocated between many systems (Figure 18).

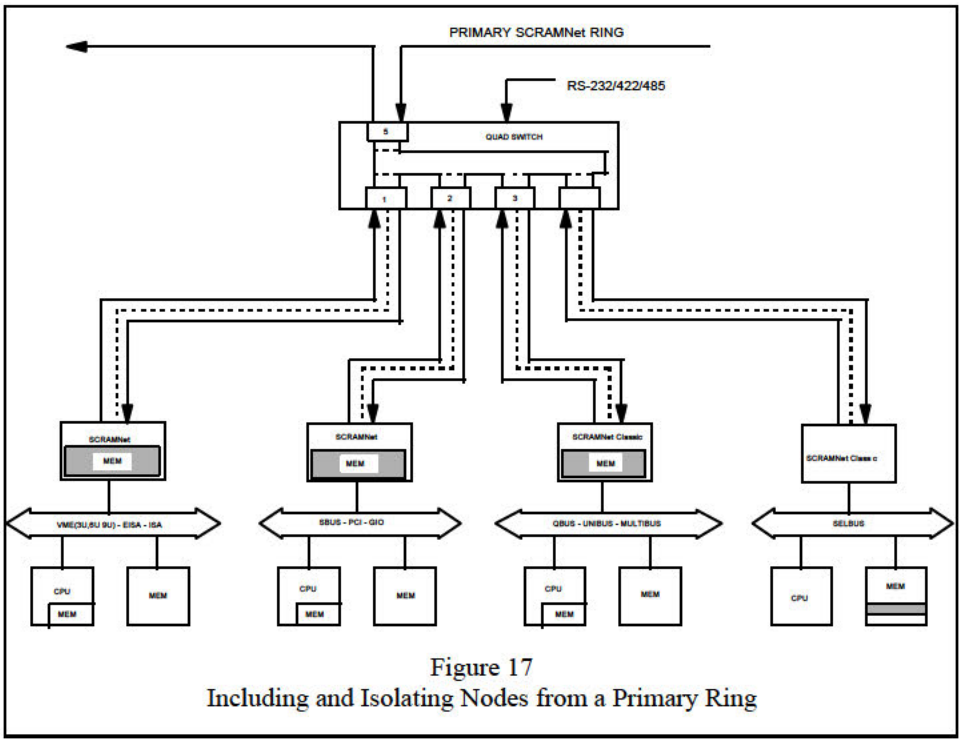


Figure 17
Including and Isolating Nodes from a Primary Ring

When used as a fiber optic bypass the electronic switching action is very fast, introducing a total network disruption of about one microsecond. This is over 10,000 times faster than mechanical, optical bypass switches and allows ring re-configuration to be performed in real-time with minimal impact on the system.

As a repeater, each Quad Switch port converts optical signals to electrical signals. The PLL (Phase Lock Loop) of each port resynchronizes and then re-transmits the pair of transmission signals. This allows each connection to the Quad Switch to be the maximum length for the type of media selected.

The Quad Switch can also perform media conversion. Since each port has a Media Access Card just like a SCRAMNet node each port can be configured to handle coax, standard link or long link fiber. This allows a signal to arrive on one media type and go out on another (Figure 19).

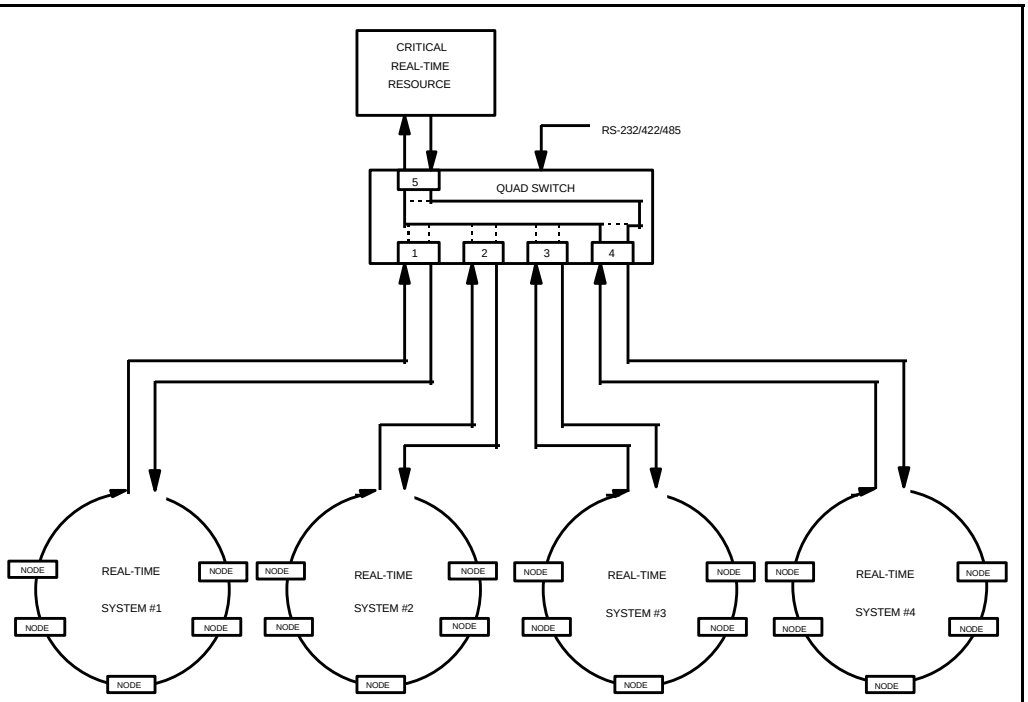


Figure 18
Reallocating Critical Real-time Resources

The Quad Switch is very user-friendly and has many features to allow dynamic control, configuration and statusing on the state of the switch. Visually LED's signify the state of node inclusion in the ring and if carrier is detected. If carrier is not detected then the port is put into Isolate state and the port is bypassed thereby maintaining ring integrity. The auxiliary connector and the associated control cable links the port to the node to allow the application running at this node to switch the Quad Switch in and out of Include or Isolate state. A manual Include/Isolate switch is used to guarantee that a node is isolated or that it can be included. A serial interface port is used to send message packets to control any Quad Switch function or obtain Quad Switch status remotely via an RS-232 or RS-422/485 connection.

F. SCRAMNet Applications

SCRAMNet's general applicability in the real-time communications marketplace is evidenced by it's broad acceptance in diverse applications in many different industries. Much of the general acceptance of the SCRAMNet products are due to the simplicity of operation, ease of use, flexibility of configuration, throughput and deterministic ability. Many industries think of SCRAMNet as being synonymous with real-time data communications. The key industries in which SCRAMNet has enjoyed considerable success are :

- •Simulation
- •Training
- •Instrumentation and Control
- •Virtual Reality

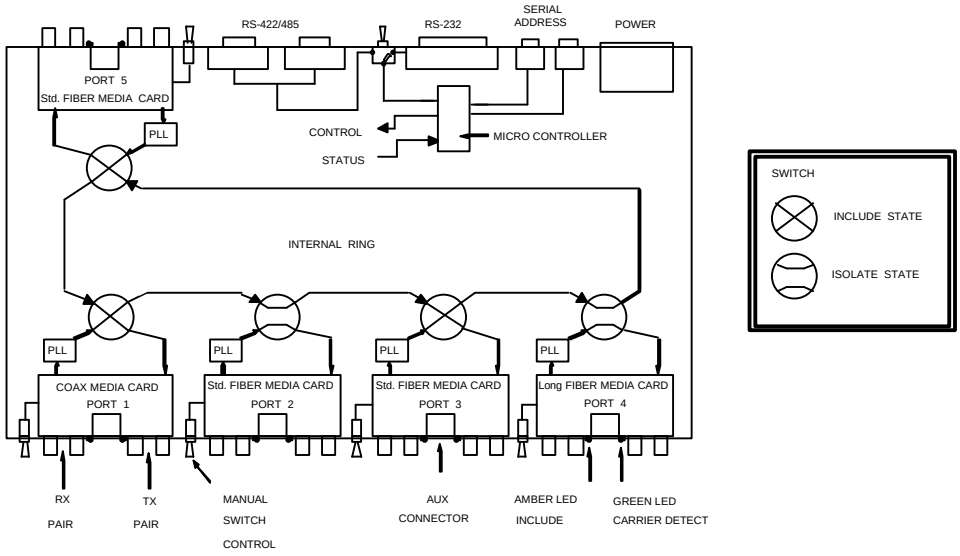


Figure 19
Quad Switch Functional Block Diagram

In the CPU intensive world of *Simulation* customers like CAE-LINK discovered that SCRAMNet was able to supply the communications backbone for its Multiple Simulator Networking program where traditional communications methods like ETHERNET and FDDI could not. Ever since Edwin Link patented his pilot training device in 1929, the LINK name has stood for simulation and training. Today CAE-Link is a major developer of flight, naval, tactical, command/control and space operator training/simulation systems. So when CAE-Link was contracted to interconnect the Army's full-fidelity Synthetic Flight Training Simulators for the UH-60 Blackhawk and the AH-64 Apache helicopters with the Force Level Simulation threat environment simulator they quickly recognized that only SCRAMNet had the flexibility and power to provide the data latency, network throughput and growth capabilities they had to have. CAE-Link now has a powerful network medium that enables it to create distributed simulations varying not only in the numbers of simulators they contain but also in the type of simulators they use.

The successful operation of a *Range & Telemetry* control room application is in it's ability to acquire data on airborne objects, process this data, and translate it to a visual representation such that command and control responses can be determined and implemented as quickly as possible. An anti-aircraft-artillery (AAA) gun battery is a prime example of this type of operation. Recent military actions like the Gulf War proved once again that "tried-and-true" anti-aircraft-artillery still has a major role to play in a country's air defense and the skill of the gun crews is at the heart of its effectiveness. One country that has recognized this is Switzerland. In seeking a new way to maximize the skill of its gun crews the Swiss Army has developed an extensive data collection and evaluation capability in the Firing Evaluation And Training System (FEATS). FEATS is comprised of five modules: firing evaluation, result data distribution, result data analysis, engagement and firing training, and firing range control. From a hardware standpoint, FEATS consists of 45 "ruggedized" 3U VME-based gun emplacement computers connected to a 6U VME-based firing-range control computer. The requirement to have firing results available to gun crews instantly upon engagement completion required a very high-speed communication between the AAA computers. The unique operating characteristics of FEATS made achieving this

level of communication difficult. Deterministic, low latency, high bandwidth communication between computers and the ability for each gun emplacement computer to “come and go” with their gun crews with minimum disruption to communications was essential. The Swiss Army chose to use the SCRAMNet+ VME3U and VME6U network nodes and the Quad switch. Together, the SCRAMNet nodes and the Quad switch provided all the communications requirements of FEATS. The Quad Switch provided the decentralized configuration control, automatic fault isolation and fast electronic switching and the SCRAMNet nodes provided the deterministic, low latency, data synchronization computer communications need. Thanks to the SCRAMNet Network the Firing Evaluation And Training System has the high-performance communications link it needs for its AAA installations.

When one thinks of *Virtual Reality* the picture that most frequently comes to mind is a high-tech device enabling participants to enter a three dimensional futuristic game or exotic artificial world. One California company, Xtensory Inc. has developed a methodology and framework for bringing virtual reality to the fast paced financial world of options trading. Using real-time data collection, analysis and presentation from multiple sources, they synthesized and presented to an on-line collection of traders a colorful 3-D representational view of futures trading. The representational graphics is viewed and manipulated with peripheral devices (stereo glasses, input glove, head tracker, wand, mouse, voice input) that promote smooth interaction between the traders and the shared data to allow quicker reactions to changing market conditions. To be effective, the rate of data availability had to occur at speeds that matched human perception and cognitive abilities - i.e., real time. Since data was collected from and shared by multiple computers and output devices the data network had to be able to interconnect multiple heterogeneous computers and do so without network performance lags that might cause uncertainties in the accuracy of the data and thereby result in slower market reactions. Xtensory recognized that SCRAMNet’s shared-memory technology coupled with 150 Mbit/sec network speed provided the level of real-time communications performance their system needed.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP