# DM3 Mediastream Architecture Overview

05-0813-001

Publication Date: April, 1998

Part Number: 05-0813-001

Dialogic Corporation
1515 Route 10
Parsippany NJ 07054

**Technical Support**
Phone: 973-993-1443
Fax: 973-993-8387
BBS: 973-993-0864
Email: CustEng@dialogic.com

For **Sales Offices** and other contact information, visit our website at **http://www.dialogic.com**

# Table of Contents

# List of Tables

# List of Figures

# 1. Introduction

This document introduces the DM3™ mediastream architecture. This architecture provides computer telephony (CT) resource developers and application solution developers with an unparalleled degree of flexibility and performance, and represents a new paradigm for planning and deploying computer telephony solutions. The purpose of this document is to show the capabilities of the DM3 mediastream architecture, not to define any specific DM3 products.

This document provides an overview of the DM3 architecture, the process used to create products based on the architecture, and an introduction to the development environment.

## 1.1. Why Build Products on the DM3 Mediastream Architecture?

Computer telephony (CT) technology allows computers to make and receive calls, handle various communication media (such as audio and data), and make conversions between different media using mediastream resources (such as network interface, fax, automatic speech recognition, and text-to-speech).

### 1.1.1. CT Products Before DM3

Before DM3, a typical CT product offered fixed functionality at the board level, with resources from a single vendor.[1] In addition, these products were dependent on the DSP and other processors being used and had limited scalability. This paradigm made it difficult for developers to rapidly integrate new resources into their applications, or to reconfigure an application using one resource to use another resource.

[1] An exception is Dialogic's Antares product, which offers over a dozen speech technologies from multiple vendors on the same hardware platform (although they cannot run simultaneously).

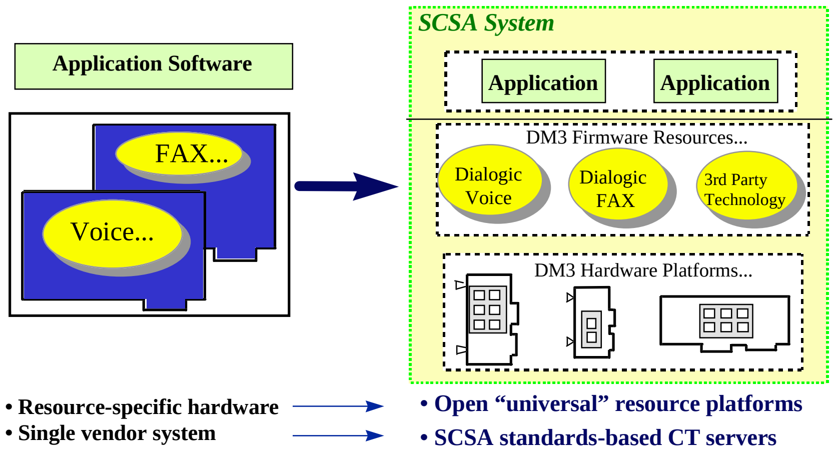

SCSA System
Application Software
Application
Application
FAX...
Voice...
DM3 Firmware Resources...
Dialogic Voice
Dialogic FAX
3rd Party Technology
DM3 Hardware Platforms...
• Resource-specific hardware
• Single vendor system
• Open "universal" resource platforms
• SCSA standards-based CT servers

**Figure 1. The New Development Paradigm**

### 1.1.2. DM3 Advantages

DM3 solves these problems with a new development paradigm that offers the following advantages:

- **Deploy multiple resources on a single platform**
  Resources such as call control, voice (playback, recording, and tone detection), fax, and automatic speech recognition (ASR) all operate on any DM3 platform.
- **Upgrade capabilities with modular hardware**
  Developers can rapidly upgrade and reconfigure their systems while protecting their investment in the underlying hardware. Modular DSP and processor-independent hardware makes modifications to a system easier and less expensive.
- **Develop in an open environment**
  The open DM3 mediastream architecture helps developers rapidly integrate multiple resources from multiple vendors. Using DMFast™ development tools, developers can build industry-standard, inter-operable solutions based on SCSA™ hardware and SCSA-compatible software.

- **Grow with an easily extensible platform**
  Since DM3 offers modular platforms, developers can rapidly adopt industry-leading processors and memories. Solutions are available on PCI, Compact PCI, and VME platforms.

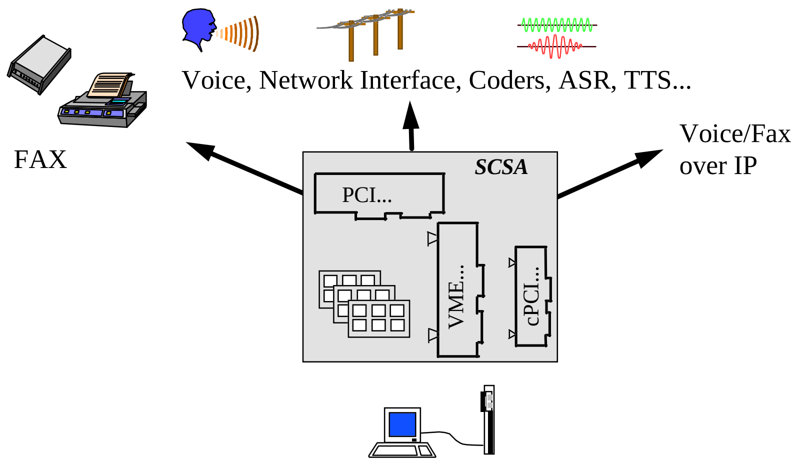


**Figure 2. Inter-operable Platforms and Technologies**

### 1.1.3. Solution: The Open DM3 Platform

The DM3 mediastream architecture is a flexible, layered, open CT resource architecture for network protocol and media processing resources that is designed for both firmware resource developers and application solution developers. Firmware resource developers can create or port algorithms onto the DM3 architecture to build firmware resources, while solution developers can create host applications that use the DM3 firmware resources. Firmware resources are embedded algorithms, such as network interface, voice, fax, and ASR, that are written to leverage the DM3 software kernel for inter-processor management and resource cooperation. This speeds CT resource development and integration for firmware resource developers and allows them to more quickly take advantage of industry advancements in signal processing technology.

### 1.1.4. DM3 Terminology

The DM3 mediastream *architecture* consists of embedded software modules and hardware that provide a platform for developing leading-edge call processing applications.

DM3 *platforms* are modular, scaleable hardware implementations of the architecture and include high density PCI, high density Compact PCI, and high density VME platforms. Lower density platforms (with 4 to 30 ports) will also become available.

Platforms are integrated with *resources* to create product bundles.

*Resources* perform the following functions:

- Digital network interfaces (T-1, E-1, ISDN)
- Internet telephony (voice over Internet)
- Fax
- Automatic speech recognition (ASR)
- Text-to-speech (TTS)
- Audio conferencing
- Call processing, including:
  - Voice record and play
  - Call progress
  - DTMF detection
  - Tone generation
  - Speed and volume control

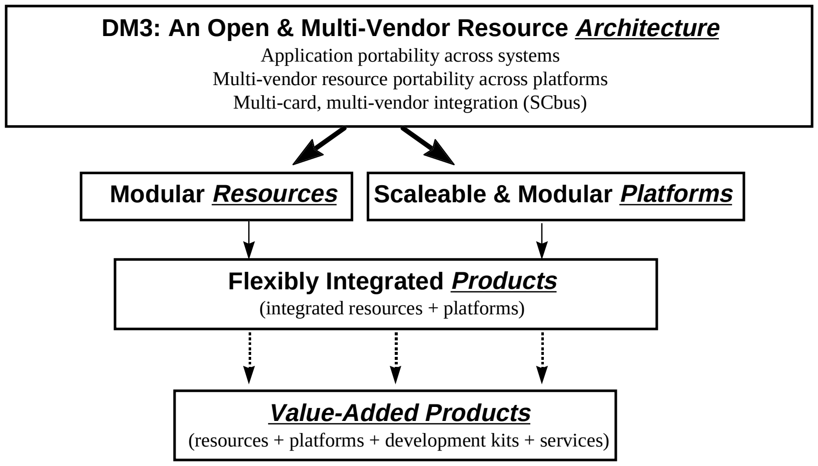

DM3: An Open & Multi-Vendor Resource Architecture
Application portability across systems
Multi-vendor resource portability across platforms
Multi-card, multi-vendor integration (SCbus)
Modular Resources
Scaleable & Modular Platforms
Flexibly Integrated Products
(integrated resources + platforms)
Value-Added Products
(resources + platforms + development kits + services)

**Figure 3. Architecture to Products**

### 1.1.5. Robust Programming Environments

DM3 is compliant with high-level APIs such as the ECTF industry standard S.100 API. Also, DM3 is designed for backwards-compatibility with existing Dialogic APIs. Lower-level API support based on the DM3 Direct Interface is provided as well. All of these interfaces are fully documented and available on all DM3 Compact PCI, PCI, and VME hardware platforms.

Resource developers can write their own algorithms to the DM3 Kernel, which helps to insulate the developer from the low-level details of the real-time operating system and increases processor independence, code portability, and supportability.

## 1.2. Features of the DM3 Mediastream Architecture

The following is a high-level list of the features of the DM3 mediastream architecture:

### Open Architecture
DM3 hardware platforms are available in standard form factors (PCI, cPCI, and VME) and are compatible with standard SCbus and CT Bus interboard buses.

### Resource independence from the hardware platform
The DM3 Kernel provides processor independence through a common low-level interface to all processors. The kernel also manages the resources of the platform. The hardware, firmware, and software can all be upgraded independently of each other, which allows for mix-and-match solutions.

### Flexible hardware platforms
The hardware design is modular, consisting of one baseboard with up to three stackable signal-processing daughterboards.

### High performance hardware platforms
The DM3 mediastream architecture contains at its core a high-performance custom ASIC called the Mediastream Management ASIC (MMA). The logical interfaces to all processors in a high density DM3 system are managed by the MMA. Furthermore, a high degree of processor independence is achieved by employing memory interfaces that are not specific to a particular processor.

The DM3 baseboard Control Processor (CP), a RISC Intel i960CF, controls the SC4000s and High-level Data Link Controls (HDLCs). There are multiple high-performance Signal Processors (SPs) available on stackable daughterboards. This includes the following processors:

- up to 18 Motorola 5630x fixed-point DSPs running at 66 or 100 MHz and faster.
- up to 8 Motorola 603E PowerPC RISC processors running at 166 or 200 MHz or above.

Other processors will also become available.

The Mediastream Management ASIC (MMA) performs μ-law to linear and A-law to linear companding, and manages the data flow between the host, the SCbus (or CT Bus), and all processors on the board. Two SC4000 ASICs (with HDLC) can access 256 full-duplex timeslots on the SCbus (or CT Bus).

For applications, there is a configurable global memory available and run-time control to increase application response times.

## High density

A single slot can support up to four T-1, E-1, or ISDN trunks running Dialogic's network interface and voice software. Also, up to 30 send and receive fax channels can be supported in one slot. Higher channel densities are planned for future releases of the DM3 platform.

## DMFast™ Development Environment

The development environment can be configured to best fit your budget or method of development:

- In Simulated Development Mode (SDM), the DM3 Kernel and demo program run on a development workstation using Wind River Systems' VxWorks to simulate the DM3 board environment.
- In Remote Access Development Mode (RADM), a remote host computer (a Sun workstation) communicates with the DM3 platform via Ethernet.
- In Host Access Development Mode (HADM), a VME Force 5V host communicates with the DM3 platform via the VME bus.

Wind River Systems' Tornado and VxWorks tools, along with other Dialogic and third party tools, make the development environment easy to use.

# 2. Software Architecture

The DM3 mediastream architecture provides several key benefits to both technology developers and application engineers:

- Resources (such as fax, voice, and ASR) are platform independent. They can be moved across platforms (PCI, Compact PCI, and VME) and operating systems (UNIX and WinNT).
- Frameworks for developing resources have pluggable components. Resource frameworks exist so that component code can be easily plugged into an existing message set. This allows interoperability with other pre-existing embedded components and Host APIs.
- Multiple application interfaces provide for low-level (Direct Interface) development and support high-level APIs (such as ECTF S.100).
- Open development environment, DMFast™, includes environments for remote and simulated development, various tools from Dialogic and third parties, a low-level kernel API, and development kits for voice, fax, ASR, TTS, voice coders, Internet coders and other resources.

The DM3 software architecture follows a straightforward model: a defined set of system services manage the software entities used to implement algorithm resources. The model, based on an architecture-neutral system-service layer called the DM3 Kernel, supports a real-time execution environment and manages resource entities, inter-component communication, and external device I/O.

The software architecture uses messages and data streams to provide a uniform communication method between all entities that access or are part of DM3 technology. Executable entities, called *components*, may be logically grouped into *resources* and physically implemented as *tasks*. This includes communication between internal software entities termed *component instances*.

The software environment also enables optimization of algorithms for improved performance when running on the DM3 platform. Resources may be split into multiple components that can distributed among different processors and processor types. For example, the CP can be used for control and management functions, while the DSP can perform signal processing functions. The kernel also shields developers from low-level specifics of the Real Time Operating System (RTOS), further facilitating the development process.

The kernel is a C functional interface that is consistent across all supported processor types, which allows resource developers to easily move components to different processors.

## 2.1. Division of Services

The DM3 software architecture is based on an architecture-independent kernel-services layer. The kernel defines a uniform set of interfaces that is consistent across the various physical implementations of DM3 and can be used by DM3-based resources. The kernel layer is independent of any processor or underlying RTOS. It also manages hardware dependencies such as byte-ordering and I/O interfaces, while normalizing RTOS features such as task management and inter-process communication.

In a hardware implementation that uses a control processor as well as signal processor(s), the kernel is replicated on each processor above the native RTOS. The processor(s) implement and use the standard messaging and data stream mechanism for communication between each other and between components.

Figure 4 details the division of kernel services on any DM3 platform.

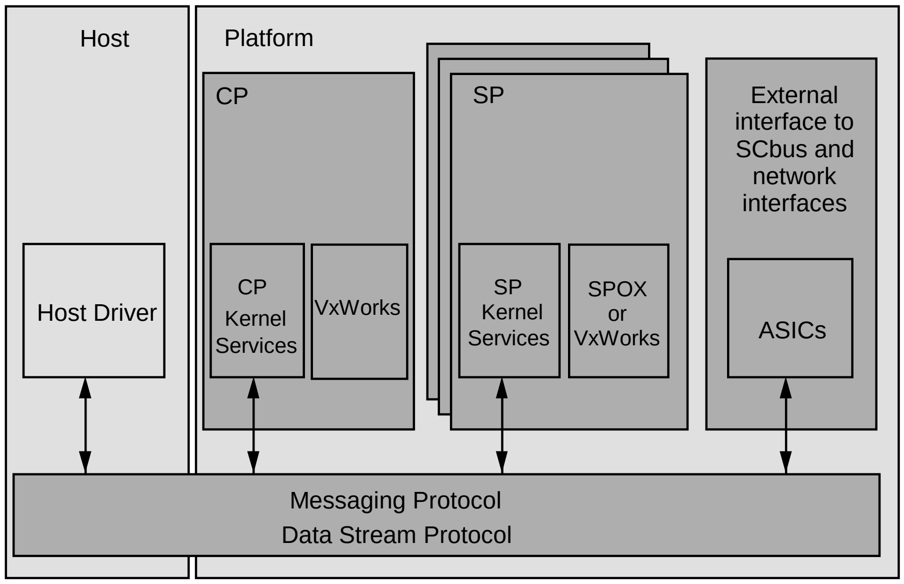


**Figure 4. Kernel Services on the DM3 Platform**

A DM3 *resource* is the primary logical entity in the DM3 embedded software environment and closely matches the SCSA concept of a resource (such as a Player, Recorder, Fax Transmitter, etc.). A resource represents a set of features to be performed and the functions of one resource are logically independent of any other resource.

DM3 *components* are logical entities that implement specific features within a resource. A resource may be separated into components to distribute functions among processors (for example, control functions versus signal processing algorithms), or to provide alternative or optional services within a resource (such as multiple coders in a player resource).

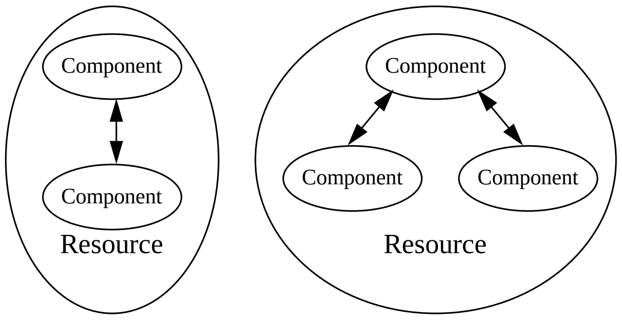


**Figure 5. Structure of a Resource**

A *component instance* is an addressable unit within the DM3 software architecture; it represents a single thread of control. The DM3 system resource management and messaging services operate at the instance level. A set of component instances that make up a resource instance communicate with one another using the DM3 system messaging services. A set of component instances is usually associated with a channel of call processing.

One implementation of a component actually encompasses two logical entities. It represents the implementation of the instances (analogous to a class for which the instances are the created objects of the class) and a separate execution context and address as a generator and manager of instances with its own address.

A *task* is an executable entity that represents an execution context for a component. The run-time environment, real-time scheduling, and inter-component messaging are all managed at a task level. Component instances execute in the context of a task, however, developers can decide to map one instance or multiple instances to a task.

The following figure shows a resource, its components, and component instances and their interaction.

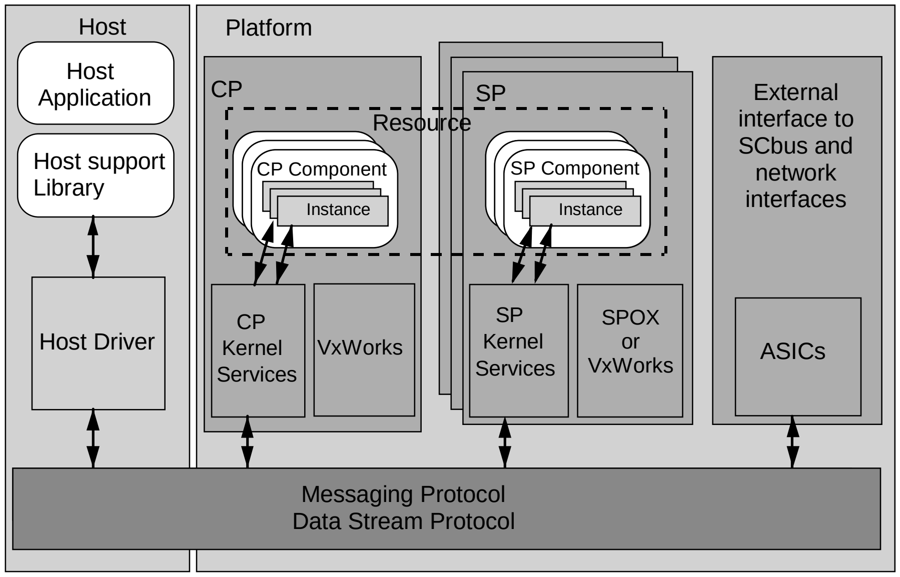


**Figure 6. Resource, Components, and Component Instances**

## 2.2. DM3 Kernel

The DM3 Kernel provides a wrapper around the processor's real time operating system, shielding the resource developer from the differences between the operating systems and the processors on which they run. For example, a developer can split a particular resource across processors, using the CP for management functionality and the SP for dedicated signal processing, without concern for processor location. This enables algorithm developers to optimize their resources and improve overall board and system performance. The DM3 Kernel provides all core platform services and has a well-defined, documented, open interface.

In addition to providing independence from the underlying physical architecture and real time operating system, the kernel provides the following set of services:

- management of call-processing components (timer services, resource management, configuration management, and memory management)
- message-sending mechanisms for command and data transfer between resources and between resources and host applications
- device I/O between resources and the external network (PSTN or Internet) and between resources via the internal TDM bus (SCbus or CT Bus)

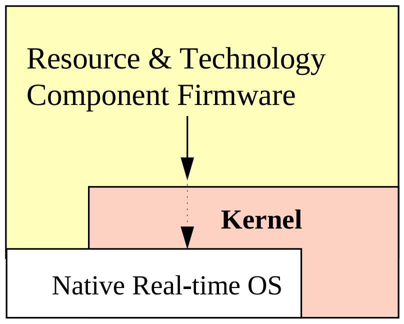


**Figure 7. Layered Software Structure**

**Component Management:** The kernel provides a complete set of resource and configuration management services. The architecture contains a flexible mechanism called the universal port scenario to concurrently support many types of call-processing resources on a single platform. In the future, an adaptable scheme will be implemented to dynamically load, unload, and configure resources on a platform, providing the ability to change the functionality of the platform over time. This future offering allows dynamic bandwidth allocation of a platform's processors, thus maximizing the number of concurrently active call-processing resources.

**Command and Data Transfer:** The kernel features a uniform communication model. DM3 uses the same communication mechanisms whether the communication is between two component instances, between component instances and host applications, or between component instances and the telephone network. The DM3 system uses messages and data streams as its two major communication mechanisms. Using the kernel, messages primarily pass commands, results, and other events between component instances and between the host applications and component instances. Data streams primarily pass large amounts of data, such as audio or fax data, between the host and component instances, and between component instances and the network (PSTN or Internet).

**Device I/O:** The DM3 Kernel services also provide the physical device drivers for all device interconnections to a DM3 platform. The drivers are not visible as a user-accessible service, but provide the underlying device support for the I/O and communications services. The driver services support the following devices:

- Mediastream Management ASIC (MMA)
- Interboard SCbus and CT Bus interfaces
- Local network interface (internal TDM bus)
- HDLC interfaces
- Host interface
- Internal timers
- External network interfaces (PSTN and Internet)
- SC message bus interface (planned)

## 2.3. DM3 Direct Interface

The DM3 Direct Interface is a low-level foundation interface that provides access to the DM3 Host Library. Available under VME Solaris and Windows NT, the Direct Interface provides full direct control of the DM3 platform utilizing standard messages for DM3 resources.

Shown in *Figure 8*, the Direct Interface shields applications from device driver specifics and essentially mirrors the DM3 Kernel functions on the host. Services provided by the Direct Interface include configuration management, message allocation, messaging services, cluster and time slot management, and bulk data stream services.

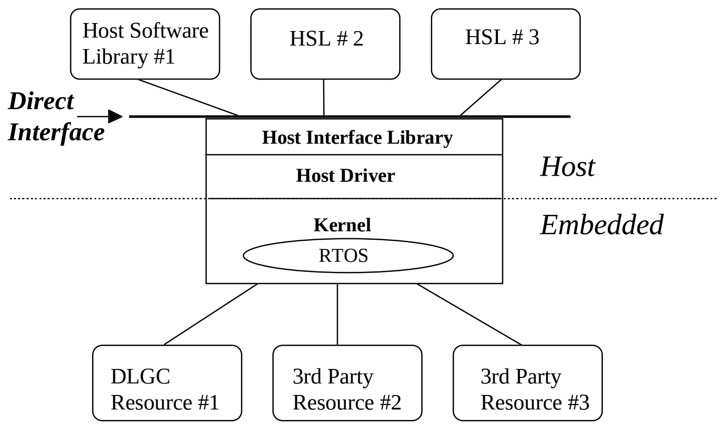


**Figure 8. DM3 Direct Interface**

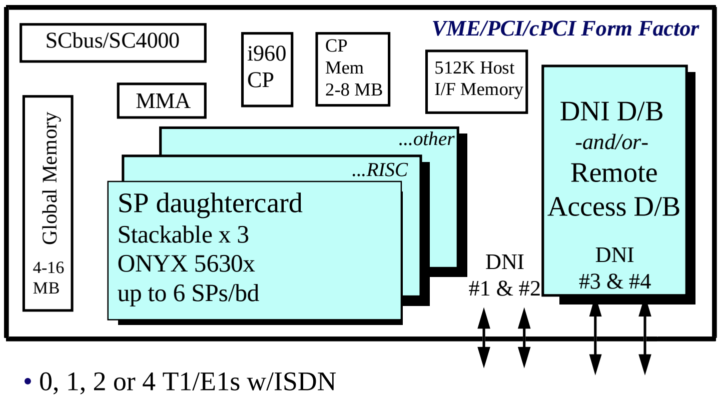
By writing to the Direct Interface, an application gains full control of the DM3 platform. This code, which directly accesses the DM3 device driver, can provide a base for building custom or proprietary APIs based on standard resource message sets. Additionally, developers using the Direct Interface must create their own resource and file management services.

# 3. High Density Hardware Architecture

This chapter discusses the architecture of the hardware used to implement high-density DM3 products. (A different hardware implementation will support lower channel densities.) The high-density hardware architecture consists of the following:

- DM3 PCI, cPCI, or VME baseboard containing the Control Processor (CP), Mediastream Management ASIC (MMA), global memory, control processor memory, host interface memory, SC4000 chips for SCbus or CT Bus access, and two digital network interfaces.
- Signal Processor (SP) daughterboards (available in 5630x or PowerPC versions).
- Digital Network Interface (DNI) daughterboard with two additional network interfaces, for a total of up to four T-1, E-1, or ISDN interfaces.
- Ethernet Network Interface Card (NIC) daughterboard which performs Internet Protocol (IP) processing and contains logic to perform connections between Internet-Ethernet networks.
- Remote access daughterboard—for development only

These items can be combined in a number of different fashions to create high-density solutions hosting a variety of technologies.



- 0, 1, 2 or 4 T1/E1s w/ISDN
- Multiple memory options
- Stackable, mix & match daughterboards

**Figure 9. High Density Platforms**

## 3.1. DM3 Baseboard

There are three baseboard types: Compact PCI, PCI, and VME. These baseboard types support daughterboards for signal processing and computing, additional network interfaces (T-1, E-1, or ISDN), and remote access during development (Ethernet and RS-232).

The baseboard contains an Intel i960CF Control Processor (CP), associated RAM and flash memory, two complete network interfaces,[2] two SC4000 ASICs, configurable global memory, host RAM for communication between the CP and the host, and a Mediastream Management ASIC (MMA).

Up to three 5630x SP daughterboards (or two PowerPC daughterboards) and a DNI (Digital Network Interface) daughterboard can be stacked on a single baseboard.

[2] The DM3 baseboard can be configured with zero, one, or two digital network interfaces.

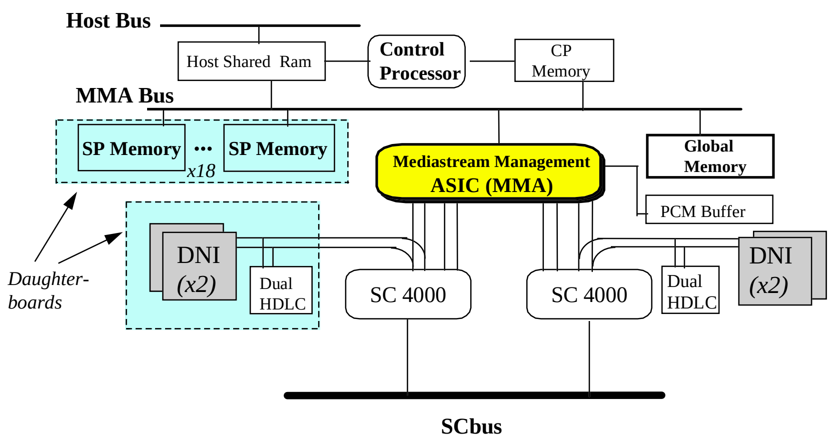


**Figure 10. DM3 Configuration Diagram**

### 3.1.1. Mediastream Management ASIC (MMA)

The heart of the DM3 high density hardware architecture is the MMA, a Dialogic-designed high-performance custom ASIC functioning as a multi-channel controller. The MMA uses a high-speed 32-bit DMA (direct memory access) bus to transfer bulk data between the DM3 global memory and the host shared RAM, the CP, the SP(s), and the two SC4000 ASICs.

By reading, controlling, and initiating interrupts, the MMA manages the data movement throughout the system without processor contention, freeing the control processor and signal processors to perform tasks other than data exchange, and eliminating individual dependencies on processor timing. Using the MMA enables very simple timing and ends complex DSP arbitration mechanisms. This design allows for rapid support of new processors and memories while using minimal overhead.

The MMA further reduces processing overhead by performing companding in hardware (µ-law-to-linear and A-law-to-linear). The MMA bus supports 8-, 16-, 24- or 32-bit data widths, enabling support of multiple signal processor types. A Byte Steering ASIC (BSA) complements the MMA and provides byte lane steering when converting from one processor bus width to another. This device provides byte or word packing as an option (for example, 12 bytes in three 32-bit words may be packed by the BSA into four 24-bit words). Since the MMA interfaces directly with the memory devices, processor bus type is not a concern, enabling support of a wide range of control and signal processors.

The MMA has two major functions:

- transfer of data between the global memory and device memory as instructed by the Global Memory Control Structure (GMCS) contents.
- transfer of data between the PCM buffer and device memory as instructed by the PCM Memory Control Structure (PMCS) contents.

### Global Memory Data Transfer

The movement of data to and from global memory is managed by the GMCS (Global Memory Control Structure), a circular structure that stores up to 256 commands. Each memory device in the system (host shared RAM, CP memory, and each SP memory) has a separate MMA interface and GMCS associated with it. The GMCS contains all the control information needed by the MMA for managing bulk data and message transfers between system memory resources.

The MMA services all the device memories in a round robin fashion. The MMA examines the GMCS associated with each memory device in a time sliced manner, servicing host shared RAM first, CP memory next, followed by all the SP memories (up to a maximum of 18). Once completed, the servicing cycle begins again. The time slice size is independently programmable for each MMA interface. (Currently, this value is set by Dialogic; in future releases, the time slice size will be customer-configurable.)

### PCM Data Transfer

The PMCS (PCM Memory Control Structure) is similar to the GMCS, but it controls PCM timeslot data transfers. The PCM buffer, shown in Figure 10, provides direct data access to and from the SC4000 chips and is read and filled in a ping-pong manner. The MMA preempts the GMCS data movement every 4 milliseconds in order to move PCM data from the 8-bit serial PCM buffer and into the appropriate SP or CP memories.

### 3.1.2. Control Processor (CP)

The single control processor (CP) on the DM3 baseboard manages access to the SCbus (or CT Bus) via the MMA and SC4000 ASICs, while also managing access to the SC message bus via an HDLC controller. The CP is an Intel i960 with built-in caches. This is a highly scaleable, industry standard RISC processor, running the VxWorks real time operating system, currently available with multiple memory options including 2, 4, or 8 Mbytes DRAM. The CP acts as a message router for DM3 control messages.

In other architectures, the control processor is typically burdened with the task of moving bulk data between the host and the board. The DM3 high-density hardware architecture eliminates this because data flow is controlled by the MMA, enabling bulk data to flow around the CP.

### 3.1.3. Global Memory

The DM3 global memory currently provides 4, 8, or 16 Mbytes of 32-bit wide DRAM which is accessible to the CP, all SPs and the host. For easy upgrades, the global memory is configured as standard 72-pin SIMM (VME form factor) or 72-pin SO DIMM (PCI and cPCI form factors) modules. This memory provides data storage and/or buffering and can act as a virtual file system (not just flat address space) for messages, ASR vocabularies, fax fonts, prompt caching, etc. Because so much memory is available, the data does not have to be frequently transferred to and from disk.

### 3.1.4. Moving Data to and from the Host

The architecture is designed so that the host can request a semaphore when data transfer is required, and then go back to work on other tasks. This means that the host is not polling, contending, or otherwise unnecessarily degrading CPU performance. When the board is available, it generates an interrupt and the host transfers block data at full bus speed. This optimizes host CPU performance. The semaphore system is controlled by bus interface logic.

### 3.1.5. Control Processor Memory

Up to 8 Mbytes of fast-page-mode DRAM is local and available to the CP.

### 3.1.6. SP Memory

SP memories are used for the processors on the SP daughterboards. The MMA supports up to 18 SP memories on up to three daughterboards (which means it supports a maximum of 18 SPs).

### 3.1.7. Host Shared RAM

512 Kbytes of host shared RAM are available on the baseboard, helping to prevent bottlenecking on the host bus. This RAM is triple-ported between the CP, the host bus, and the MMA bus.

### 3.1.8. PCM RAM

The PCM RAM provides temporary storage for 4 milliseconds of PCM data. The serial data is stored as bytes in two 32-byte ping-pong buffers per channel.

### 3.1.9. SCbus/SC4000

The SC4000 is a Dialogic-designed custom VLSI circuit optimized for use in the SCSA call-processing environment.[3] The two SC4000 chips on the DM3 baseboard provide a large-capacity interface of up to 256 full-duplex timeslots between internal (local) serial TDM streams and an external (expansion) TDM bus. Using all 256 timeslots, four T-1/E-1 spans can be accommodated plus reference timeslots for complete echo cancellation, thus achieving the DM3 goals of maximized performance and channel density.

The primary function of the SC4000 is to exchange digital data between the timeslots on the local bus and the timeslots on the expansion bus. A microprocessor interface allows the host CPU to define the timeslots and serial data streams used for data exchange.

[3] Computer telephony hardware developers can order the SCSA Hardware Development Kit from VLSI Technology, Inc. to design an SCbus interface using the SC4000 ASIC.

Internal buffering allows the exchange of data between local and expansion buses of different speeds. The SC4000 supports SCbus, CT Bus, and MVIP bus formats and provides full clock and data support for each. The switching functions and operational configurations of the SC4000 are fully software programmable.

On VME and cPCI boards, the backplane or a ribbon overlay may be used for interboard communication; on PCI boards, a ribbon cable is used as the physical SCbus or CT Bus to connect multiple boards together.

The SC4000s connect to the DNI (Digital Network Interface) daughterboard such that local timeslot switching is possible without consuming timeslots on the SCbus or CT Bus (if all the data is available locally). Either the network interface logic, the SCbus or CT Bus, or the local oscillator can act as the master clock.

### 3.1.10. Digital Network Interfaces

The DM3 hardware supports 0, 1, 2, or 4 digital network interfaces. Two network interfaces can be installed on the DM3 baseboard for connecting to T-1, E-1, or ISDN trunks. When the Digital Network Interface (DNI) daughterboard is mated to the baseboard, the number of network trunks is doubled.

### 3.1.11. High-level Data Link Communication (HDLC) Controllers

The DM3 baseboard contains three HDLC controllers; one each for the two T-1/E-1 interfaces and one for future support of the SC message bus. The DNI daughterboard also contains a dual HDLC controller providing up to four D (data) channels in a quad T-1/E-1 configuration.

## 3.2. SP Daughterboards

Each SP daughterboard is configured with Motorola® 5630x™ DSPs (up to six DSPs per daughterboard), or the Motorola PowerPC™ 603e processor (up to four processors per daughterboard). Up to three SP daughterboards of varying types may be stacked on each baseboard within power limits. The same SP daughterboards can be used on Compact PCI, PCI, and VME baseboards.

In the future, additional signal processors can be used on SP daughterboards, and daughterboards with different processors can be combined for maximum flexibility.

A baseboard with a single SP daughterboard requires only one VME, Compact PCI, or PCI slot. A baseboard with two or three stacked SP daughterboards utilizes two slots in a chassis.

### 3.2.1. Motorola-based 5630x Daughterboards

The Motorola 5630x daughterboards are available in several versions. The 5630x DSP itself offers the following advantages:

- 66 and 100 MIPS
- Built in DRAM controller
- Large program space (eliminates 64k limit)
- Compatible w/56000 software
- Power efficiency using 3.3V technology

The 5630x DSPs offer optimized price versus performance by providing DRAM for programs or large linear tables and local SRAM (one wait state) for random access of data. If a loop fits within a 1K cache, the chip can run zero wait states.

The DM3 5630x SP daughterboard features include:

- One to six (de-)stuffable 24-bit Motorola 5630x DSPs
- 256K/1Mwords DRAM with 1K program cache per DSP
- 128K SRAM per DSP

### 3.2.2. PowerPC Daughterboards

These high performance and cost effective boards contain between one and four Motorola 603e processors and offer the following advantages:

- 166, 200, 225, and 240 MHz
- 8 or 32 MB local SDRAMs (@ 66 MHz bus speed)
- High density reduces slots and system cost
- 32 KB built-in cache

- Excellent C programmability for efficient code compilation and processor independence
- Broad range of robust tools (e.g., WindRiver VxWorks/VxSim)
- Relatively low power dissipation at 3.3 V; less than 30 W for a PowerPC daughterboard with 4 installed processors.

## 3.3. Digital Network Interface (DNI) Daughterboard

The digital network interface (DNI) daughterboard provides two additional T-1, E-1, or ISDN network interfaces to supplement the two network interfaces on the baseboard. The DNI daughterboard also contains a dual HDLC controller providing up to four D (data) channels in a quad T-1/E-1 configuration.

## 3.4. Ethernet Network Interface Card Daughterboard

The Ethernet Network Interface Card (NIC) daughterboard contains a Motorola 603e PowerPC processor which performs Internet Protocol (IP) processing. The daughterboard also contains logic to perform connections between Internet and Ethernet networks.

## 3.5. Remote Access Daughterboard

An integral part of the DMFast software development kit, the remote access daughterboard contains 10 Base-T Ethernet and RS-232 serial port interfaces. Using C tools at a Sun workstation, you can boot, download, debug, and gain direct access to the DM3 baseboard through Ethernet. This board is only used during the development stage.

## 3.6. Data Communication

The DM3 mediastream architecture is message-driven; messages pass commands, results, and other events between component instances and between the host applications and component instances.

As shown in *Figure 11*, there are only two types of data under DM3: *messages* and *bulk data streams*. All *messages* go through the Control Processor allowing it to act as a message router. The CP, located on the baseboard, manages the *messages* passed between components without becoming a bottleneck for the *bulk data streams* being passed between the Host and Signal Processors or between the Host and SCbus (or CT Bus).

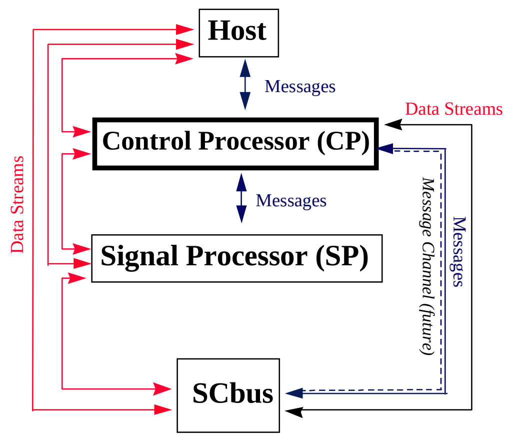


**Figure 11.  Messages and Data Flow**

Bulk data can flow around the CP directly to an SP located on one of the SP daughterboards. While the architecture supports bulk data streams to and from the CP, their usage should be limited so as not to overburden the CP.

# 4. Cluster Concepts

Under the DM3 Mediastream architecture, data switching is accomplished using a concept known as a cluster. A cluster acts as the communication manager for a group of component instances that share the same set of timeslots. This chapter presents general information on clusters and some guidelines on how clusters can be used.

This chapter is applicable for:

- Component developers creating DM3-compatible components
- Application developers using components to create a host application

## 4.1. Concepts and Terms

The following sections describe the most common concepts and terms used by the DM3 data switching architecture.

### 4.1.1. TDM Data

In the DM3 architecture, components process Time Division Multiplexed (TDM) data. For example, a tone generator component can generate DTMF digits (dual-tone multi-frequency digits) as a TDM data stream. Also, a recorder component can take a TDM data stream and convert it (encode it) into a form that can be written to a disk. Another example is a telephony service component (TSC) which can relay TDM data to and from a central office interface, typically a T-1 or E-1 interface. Figure 12 shows TDM data flowing between component instances.

Figure 12 also illustrates how the flow of TDM data is centralized around the TSC component. The player and tone generator instances send TDM data to the TSC instance, while the recorder and signal detector instances receive TDM data from the TSC instance. The TSC instance in this example represents the interface into the network (PSTN or Internet) and is called the **central instance** of the cluster, because it acts as a communication center.

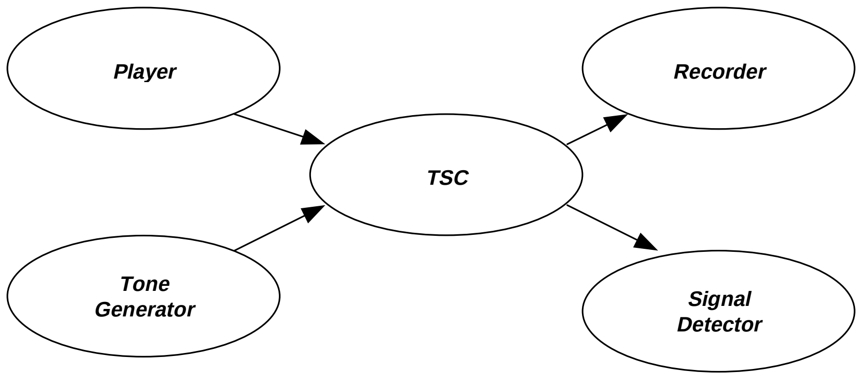


**Figure 12. TDM Data Exchange between Component Instances**

### 4.1.2. C-Streams

The Memory Management ASIC (MMA) transfers data to and from CP and SP RAM via data streams attached to circular buffers. Each circular buffer is associated with (connected to) an MMA timeslot to allow data to be sent and received from TDM busses. The streams of data are called **circular streams** or **C-Streams**. Data carried by DM3 C-Streams is the only type of data that can be managed by clusters.

### 4.1.3. Ports

In the DM3 switching architecture, the term **port** represents specific TDM data flow points. DM3 component instances use ports to transfer TDM data. Valid DM3 port types are:

| | |
|---|---|
| **N-Port** | Network port. It represents the point at which TDM data flows into or out of a network front-end interface (such as T-1, E-1, or ISDN interface). |
| **R-Port** | Resource port. It represents the point at which a circular stream carrying TDM data flows into or out of a component instance. For example, a player component instance sends TDM data out of an R-Port. |

| | |
|---|---|
| **S-Port** | SCbus port. It represents data flow to/from an SCbus timeslot. (This nomenclature is also used for CT Bus ports.) |

In relation to the port types described above, TDM data can be described as flowing in a particular direction. In order to fully characterize a port in the DM3 architecture, it must be specified with a direction as well as a type. A port can either be an **IN-port** or an **OUT-port**.

**IN** represents data flowing ***into the device or component instance***. An IN-port is a data consumer. For example, data flows into the network (PSTN or Internet), into the SCbus/CT Bus, into the recorder component instance, or into the signal detector component instance.

**OUT** represents data flowing ***from the device or component instance***. An OUT-port is a data producer. For example, data flows out of a network interface, out of the SCbus/CT Bus, out of the player component instance, or out of the tone generator component instance.

DM3 ports are referred to using a simple standard notation: $\mathbf{TYPE}_{\mathbf{direction}}$

For example, in Figure 13, $\mathbf{R}_{\mathbf{I}}$ is a resource IN-port, $\mathbf{R}_{\mathbf{O}}$ is a resource OUT-port, $\mathbf{N}_{\mathbf{I}}$ is a network IN-port, and $\mathbf{N}_{\mathbf{O}}$ is a network OUT-port. Notice in this figure that instances compete for control of a shared IN-port. For example, the tone generator and the player instances compete for the $\mathbf{N}_{\mathbf{I}}$ port of the TSC instance. There is no competition for a shared OUT-port, such as $\mathbf{N}_{\mathbf{O}}$, since both input ports connected to it get data from it at the same time. Port sharing is made possible by the DM3 cluster functionality.

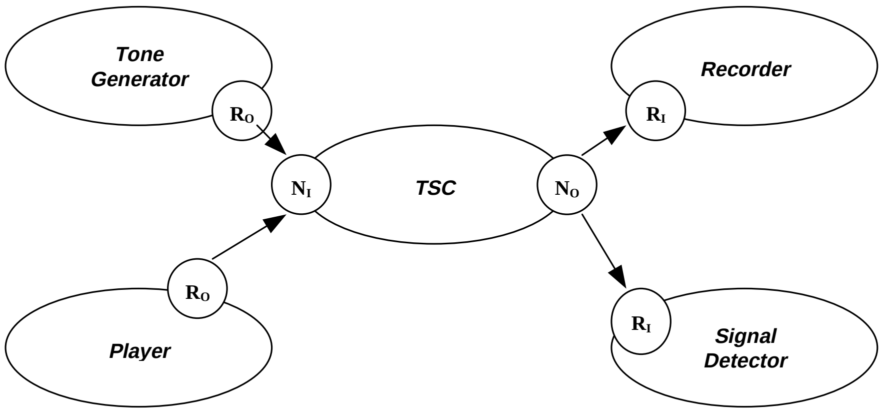


**Figure 13.  TDM Data Flow with Port Notation**

### 4.1.4.  Clusters

The figures above suggest that when component instances are sharing TDM data, it is natural to group them together and think of them as a single unit. This is exactly the approach used in the DM3 architecture. Specifically, the architecture defines a **cluster** as a managed collection of component instances and ports which are controlled by API function calls. Clusters control how TDM data is transferred between component instances and how TDM data is routed to and from the SCbus/CT Bus.

The communication center for a cluster is called a **central instance**, which is (usually) the first instance that was allocated into the cluster. A cluster consists of a central instance and one or more additional component instances. A component instance is allocated into a specific cluster. Once a cluster has been created, the kernel controls the cluster ports to allow the component instances within that cluster to share ports in an intelligent manner.

**NOTE:** The component instances and ports in a cluster must all exist on the same board.

A resource IN-port or resource OUT-port can be accessed from any processor in a DM3-based system. For example, a resource OUT-port can be opened for write operation by processor 1 and processor 2 in a multi-processor system.

Any components that exist on the same CP or SP, and execute under the same DM3 Kernel, can physically share the C-Streams in the processor's data space, but in the receive direction only. For example, if two instances on the same processor open the same resource IN-port, they share the local buffer in internal memory.

### 4.1.5. Connections

A TDM data flow path between two ports is called a **connection**. Establishing a TDM connection has two steps, *creation* and *activation*. Creating connections informs the DM3 kernel of potential data flow between ports. Creation establishes a map of port connections where data can flow. Activating a connection establishes the physical link between an input port and an output port and allows data to flow between them. A connection becomes active when an output port requests to talk or upon receiving a host request. The kernel establishes a physical connection according to the map defined in the creation process.

The DM3 kernel builds and stores a map of connections when they are created. It uses this map to efficiently and intelligently create physical connections (that is, switch between ports) when they are activated. Since the kernel is aware of the underlying hardware, it utilizes the hardware in the most efficient manner. In addition to intelligent routing, the kernel also removes the burden of storing connection map information inside an application or a component, thus easing application design.

The kernel creates and activates connections according to the following rules:

- A connection can only be created between an OUT-port and an IN-port; a connection between two IN-ports or two OUT-ports is illegal.
- An OUT-port can have active connections to multiple IN-ports (it can broadcast data to multiple IN-ports).
- An IN-port can have an active connection to **only one** OUT-port at any one point in time (it can only listen to a single OUT-port).

Note that connections may be created for multiple OUT-ports to one IN-port, but that only one of these connections can be active at any given time.

A central component instance is usually the first instance allocated to a cluster and is also the instance that all the other instances in a cluster exchange TDM data with. This definition can be extended to include the concept of **central ports**. If a TSC component instance is the central instance in a cluster, the cluster will have two central ports, the $\mathbf{N_I}$ port and the $\mathbf{N_O}$ port. When additional component instances are allocated into this cluster, connections are automatically created (but not activated) between the ports on those component instances and the central ports whenever that connection is possible.

For example, a cluster has a TSC with central $\mathbf{N_I}$ and $\mathbf{N_O}$ ports. When a player instance is allocated into this cluster, a connection is automatically created between the player instance's $\mathbf{R_O}$ port and the TSC's $\mathbf{N_I}$ port. The player's $\mathbf{R_O}$ port is not connected to the TSC's $\mathbf{N_O}$ port since this connection is not possible (two OUT-ports may not be connected).

### 4.1.6. Talker Protocol

An important concept in the DM3 switching architecture is the Talker protocol, in which the DM3 kernel automatically activates connections at the request of component instances. The Talker protocol is used by component instances with OUT-ports that need to transmit TDM data.

Component instances that want to send TDM data out of $\mathbf{R_O}$ ports must first make a request to the kernel for permission. The kernel examines its connection map and determines which IN-port(s) the $\mathbf{R_O}$ port is connected to. The kernel then pinpoints the desired IN-port, deactivates any other connection using this IN-port, and activates the connection from the requesting $\mathbf{R_O}$ port to the IN-port. The kernel then informs the component instance that it can start transmitting TDM data. Once transmission is complete, any interrupted connections will be restored. This protocol ensures that an input port is only listening to one output port at any one point in time.

The Talker protocol is transparent for DM3 $\mathbf{R_O}$ ports, because the host application needs to do nothing to support it. A host application only needs to inform a player instance to play TDM data, and the connections are automatically set up by the kernel.

For SCbus (**$S_O$**) ports, the host application must provide additional information to support the Talker protocol because an SCbus port represents a bus connection. Bus connections are not DM3 addressable components, so the kernel cannot make stop transmission requests when it wishes to deactivate the connection. The host must therefore provide:

- either a proxy component that carries out the Talker protocol on behalf of the device using the **$S_O$** port,
- or default actions to carry out when the kernel tries to activate and deactivate the connection (such as, always reject the kernel request, or always allow the kernel to interrupt).

### 4.1.7. Data Switching via the SCbus (or CT Bus)

To allow the host to control SCbus (or CT Bus) connections, SCbus timeslots are modeled after DM3 component instances using the concept of an *SCbus resource*. (The term *SCbus resource* in this case also applies to components operating on the PCI DM3 platform, which can use either the SCbus or CT Bus for interboard connectivity.) SCbus resources, while part of the DM3 kernel, are treated like other component instances as shown in Figure 14. SCbus resources are allocated to a cluster using the **SCRES_Std_ComponentType** attribute.

In Figure 14, the central component instance (the TSC instance) is shaded. Also in that figure, one SCbus resource is transmitting TDM data to the TSC instance. For example, a Text to Speech device on a different board may be outputting speech data on SCbus timeslot 200. The SCbus resource is added to the cluster and configured to have an SCbus OUTPUT port (taking TDM data from SCbus timeslot 200) which is connected to the TSC's central INPUT port.

The second SCbus resource in Figure 14 is receiving TDM data from the TSC. An example is an ASR instance on another DM3 board that needs to receive data from the network (PSTN or Internet). The ASR instance would be configured to receive data from the SCbus time slot to which the SCbus IN-port in Figure 14 is connected.

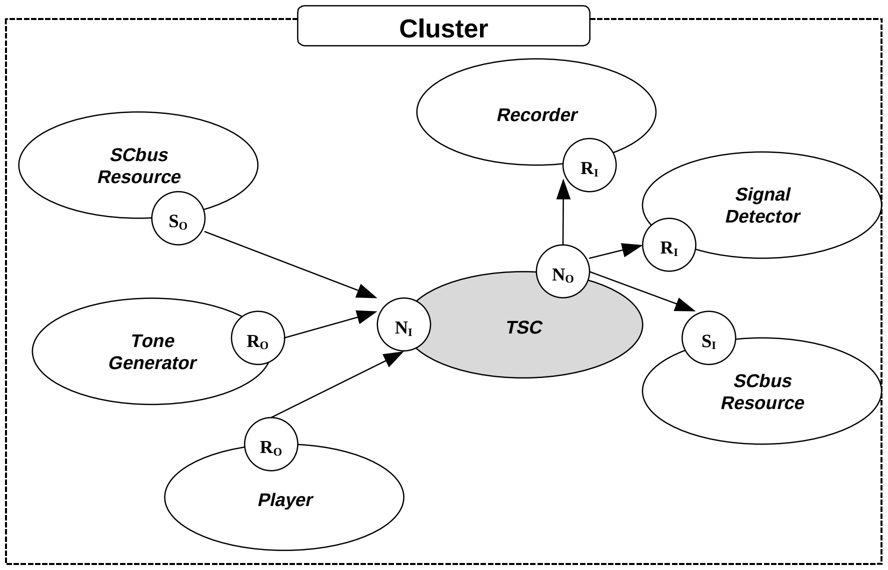


**Figure 14. DM3 Cluster with SCbus Resources**

## 4.1.8. Connecting Clusters

The figure below shows connections between clusters, which are called **inter-cluster** connections. Note that central instances are shaded and SCbus (or CT Bus) timeslot connections are dashed lines. The connection between the TSC and ASR component instances is an on-board or direct connection. The connection between the two SCbus resources is a host-managed connection.

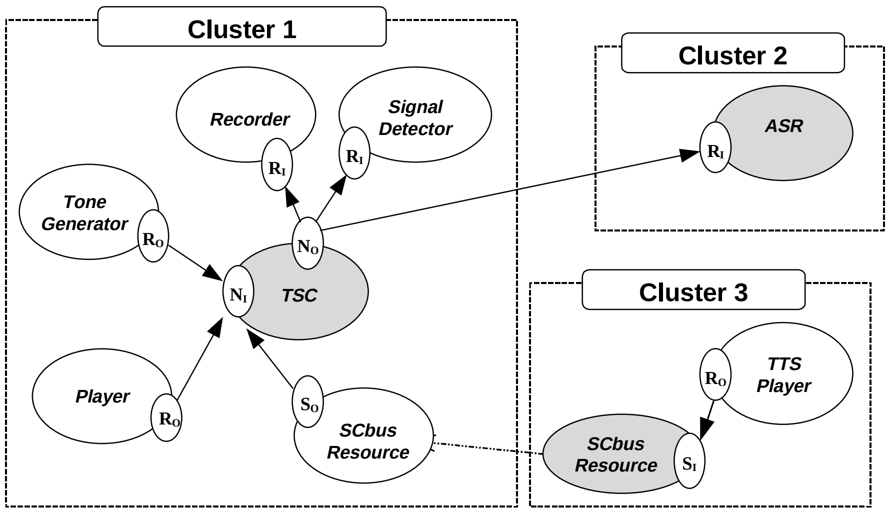


**Figure 15. Inter-Cluster Connections**

Connections between clusters can occur in a variety of ways through different port types. Inter-cluster connections are created by an external entity, usually a host application. In certain situations, the activation of the connection is controlled by the application and in other situations it can be controlled by the component instance itself using Talker protocol. The following paragraphs describe the two cases of inter-cluster connections shown in Figure 15.

## Direct Cluster Connection

For a direct connection, both clusters must reside on the same board. The direct connection shown in Figure 15 shows an inter-cluster connection between a TSC instance in Cluster 1 and an ASR instance in Cluster 2. This connection joins an $\mathbf{N_O}$ to an $\mathbf{R_I}$ port and uses on-board DM3 hardware devices such as the memory management ASIC (MMA) and occurs under control of the DM3 kernel. In this scenario, a host application must make a request to the kernel to create this connection.

## Cluster Connection via SCbus (or CT Bus)

An SCbus connection can exist between resources in clusters that exist on two separate DM3 boards. This type of connection can also be made between component instances on the **same** DM3 board, however, it is an inefficient use of SCbus (or CT Bus) timeslots. Figure 15 shows a connection between a TSC instance in Cluster 1 and a TTS instance in Cluster 3. This connection links an $\mathbf{S_O}$ to an $\mathbf{S_I}$ port and uses SCbus (or CT Bus) timeslots to exchange TDM data. Allocation of SCbus resources originates upon request from the host, either by an application or server software, such as an SCSA Server. An SCbus connection is set up by allocating an SCbus resource in Cluster 1 with an $\mathbf{S_O}$ port and assigning an SCbus (or CT Bus) timeslot to it. Cluster 3 was created with an SCbus resource as the central instance and the TTS instance is transmitting to the $\mathbf{S_I}$ port. Both SCbus resources are configured to use the same SCbus timeslot.

Because SCbus ports are really bus connections and not addressable component instances, Talker protocol in Cluster 1 must be managed by the host application. Cluster 1 must be able to switch between the SCbus connection and the player and tone generator connections inside the cluster. The host can support Talker protocol by any one of the following:

- informing Cluster 1 of the TTS component instance address so that Cluster 1 can send Talker protocol messages to the TTS component instance.
- proactively informing Cluster 1 via messages from the host when the SCbus resource wants to transmit and stop transmitting.
- setting the default Talker protocol response of the SCbus resource in Cluster 1 to always accept interruption. In this case, the connection between the TSC and the SCbus resource is only active when both the player and tone generator are not transmitting.
- setting the default Talker protocol response of the SCbus resource in Cluster 1 to always reject interruption so the connection between the TSC and the SCbus resource is always active. When the host wishes to stop, it must deallocate the SCbus resource.

## 4.2. Host Application Cluster Control

This section deals with the mechanics of how a host application controls clusters, how it uses them to exchange TDM data on the SCbus or CT Bus, and how it directs component instances to exchange TDM data on the network front end. A host application using the DM3 platform controls clusters by performing the following:

1. Find a cluster.
2. Add component instances to a cluster.
3. Add SCbus resources with input or output ports to a cluster.
4. Assign SCbus or CT Bus timeslots to SCbus resources.
5. Remove SCbus resources from a cluster.
6. Maintain Talker protocol for SCbus output ports.

**Advanced Tasks:**

7. Change default cluster connections.
8. Connect clusters on the same board together.

**NOTE:** The DM3 Direct Interface Host Library supports two operating systems: VME Solaris and Windows NT. The function names for each operating system are generally similar, except for the prefix. (A **q** prefix is used for VME Solaris functions and an **mnt** prefix identifies Windows NT function.) In this chapter, *nn* is used to represent either prefix. For further details on any of the functions mentioned here, refer to the *DM3 Direct Interface Reference* or *DM3 Host Library Reference* for a particular operating system.

Table 1 gives a summary of the host library functions used to accomplish each task.

**Table 1. Host Cluster Control Tasks**

| **To Do This…** | **Use Host Library Function(s)…** |
|---|---|
| Find a cluster | *nn***CompFind( )**<br>*nn***ClusterByComp( )** |
| Add components to the cluster | *nn***CompAllocate( )** |
| Add SCbus resources with input and output ports to clusters | *nn***CompAllocate( )** |
| Assign SCbus or CT Bus timeslots to SCbus resources | *nn***ClusterTSAssign( )** |
| Remove SCbus resources from cluster | *nn***CompFree( )** |
| Manage Talker protocol for SCbus output ports | **Full Talker Functions:**<br>*nn***ClusterActivate( )**<br>*nn***ClusterDeactivate( )** |
| **For Advanced Tasks…** | **Use Host Library Function(s)…** |
| Change default cluster configuration | *nn***ClusterDisconnect( )**<br>*nn***ClusterConnect( )** |
| Connect cluster ports in different clusters on the same board | *nn***ClusterConnect( )** |

### 4.2.1. Finding a Cluster

Existing clusters may have component instances added to them. If a TSC component exists on the board, most applications will allocate components into a TSC's pre-existing cluster. To add instances to an existing cluster, first find a cluster with the necessary attributes.

To find clusters, two functions are used:

| | |
|---|---|
| *nn***CompFind( )** | Finds a component with the specified set of attributes |
| *nn***ClusterByComp( )** | Finds the cluster that a specified component belongs to |

For example, to find the cluster controlling T-1 timeslot 6, use the *nn***CompFind( )** function specifying a TSC component with timeslot 6. After the component is found, retrieve the TSC's cluster by calling the *nn***ClusterByComp( )** function using the found component address.

### 4.2.2. Adding Components to Clusters

DM3 component instances are added to clusters during component allocation. When a host application allocates a component with the *nn***CompAllocate( )** function, it must specify the cluster to which it belongs.

When component instances are added to a cluster, a set of default connections are automatically established. The kernel maps central ports to valid non-central ports, that is, IN-ports are connected to OUT-ports. A TSC instance always has a pair of central ports, therefore, whenever a TSC instance is in a cluster, it will be connected to any instances added to that cluster.

Occasionally, SCbus resources are configured as central ports, typically if the cluster does not contain a TSC instance.

Figure 16 is an example of the default connection map created when a cluster contains a TSC, player, recorder, tone generator, and signal detector. The TSC instance is shaded to indicate that it is the central instance with two central ports. For most applications, it is not necessary to configure clusters differently than the default configuration.

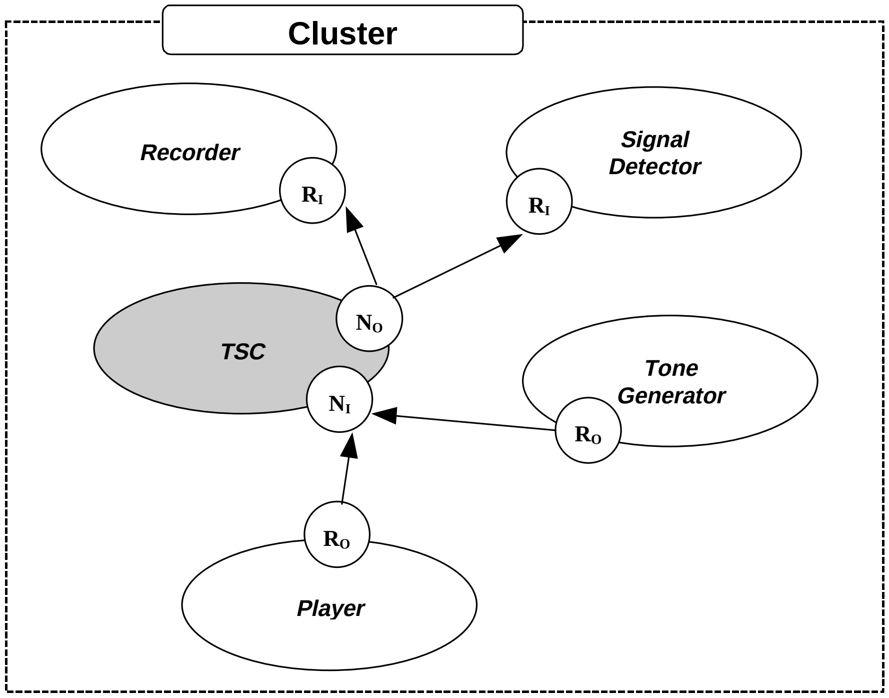


**Figure 16. Default Cluster Connections Example**

### 4.2.3. Assigning a timeslot to an SCbus Resource

When an SCbus resource is created, it does not have a specific SCbus (or CT Bus) timeslot assigned to it. The SCbus IN-ports are used to transmit TDM data into a specific SCbus (or CT Bus) timeslot and the SCbus OUT-ports are used to receive data from a specific SCbus (or CT Bus) timeslot.

To assign a timeslot, an application uses the *nn***ClusterTSAssign( )** function call specifying the:

- cluster
- SCbus resource component address
- SCbus resource port identity
- SCbus (or CT Bus) timeslot number

To stop data from being transmitted over the SCbus or CT Bus, *nn***ClusterTSUnassign( )** can be called to clear any timeslot assignments from the SCbus port.

### 4.2.4. Talker Protocol

When an SCbus resource with an $\mathbf{S_O}$ port is part of a cluster, DM3 talker protocol must be followed. The host application has several choices:

- Follow full DM3 talker protocol.
- Follow a simple DM3 talker protocol.
- Provide the address of a component that follows full DM3 Talker protocol.

#### Simple Talker Protocol

Simple Talker protocol provides the means for a host application to add SCbus resources that "talk" (transmit TDM data) with minimal application talker protocol overhead. This is accomplished by using the *nn***ClusterActivate( )** and *nn***ClusterDeactivate( )** host library function calls.

The *nn***ClusterActivate( )** call is used to activate the connections from the SCbus OUT-port to the IN-ports inside the cluster. For example, in the figure below, when the SCbus OUT-port is made active by the host application, the network IN-port is accepting TDM data from the SCbus and the Tone generator and the Player connections are not active.

Once the connection is active, what happens when the Player wants to generate TDM data? This is dependent on how *nn***ClusterActivate( )** was used.

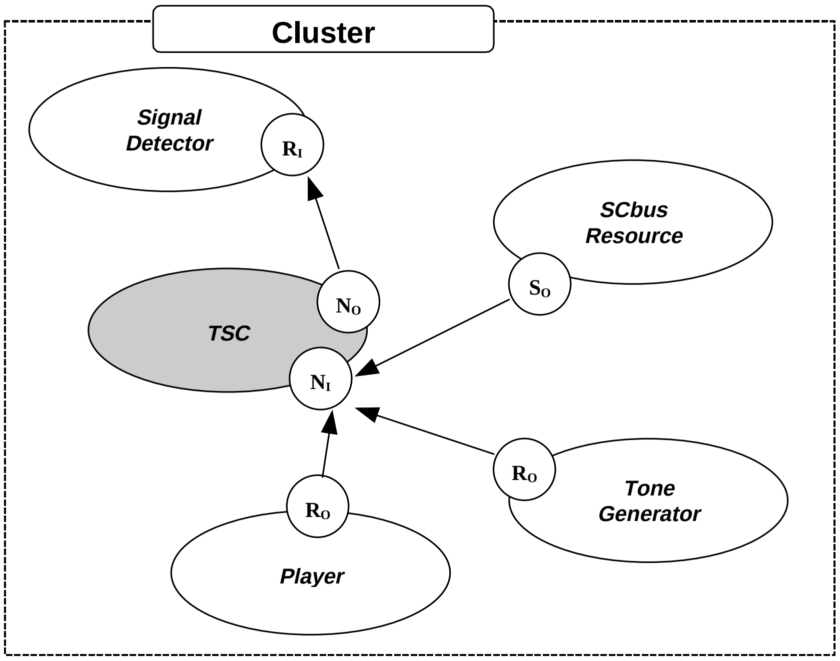


**Figure 17. SCbus Resource Talking**

When the function is called, one of two *default talker protocol response* options is supplied. The default response informs the kernel how to handle the situation. Valid options are defined in *mercdefs.h*. They are:

| | |
|---|---|
| *QCLUST_AutoReject* | Data from the SCbus (or CT Bus) cannot be interrupted by any other OUT-port resource in the cluster. The connection between the $\mathbf{S_O}$ port and all the input ports it connects to will remain active until the host application explicitly deactivates the connection with a *nn***ClusterDeactivate( )** function call. |

| | |
|---|---|
| *QCLUST_AutoAccept* | Data from the SCbus (or CT Bus) can be interrupted by any other OUT-port resource in the cluster. The connection between the $\mathbf{S_O}$ port and cluster input ports can be temporarily suspended and re-established after the interrupting resource has finished. No notification will take place if this occurs. |

### Full Talker Protocol

A host application can act as a proxy for a resource that outputs data on the SCbus (or CT Bus) and transmits data into the $\mathbf{S_O}$ port of a cluster.

A resource that outputs data must be able to send a set of commands to request to talk, and must be able to reply to kernel requests for interruptions with a specific set of messages.  These messages are summarized below:

| **Message Name** | **Action** | **Response Message** |
|---|---|---|
| *QClusterSuspend* | kernel request to stop output of data | *QClusterSuspendResult* |
| *QClusterResume* | kernel request to resume output of data | *QClusterResumeResult* |
| *QClusterActivate* | component request to output data | *QClusterActivateComplete* |
| *QClusterDeactivate* | component informing kernel that it has stopped outputting data | *QClusterDectivateComplete* |

### 4.2.5.  Changing the Default Cluster Configuration

This is considered an advanced task since the default cluster configuration should handle most situations.

Reconfiguring a cluster means that the host application will connect ports inside a cluster to each other in a configuration that is different than the default behavior. The figure below is a default cluster connection map that results when a TSC is in a cluster with a player instance and an SCbus resource.

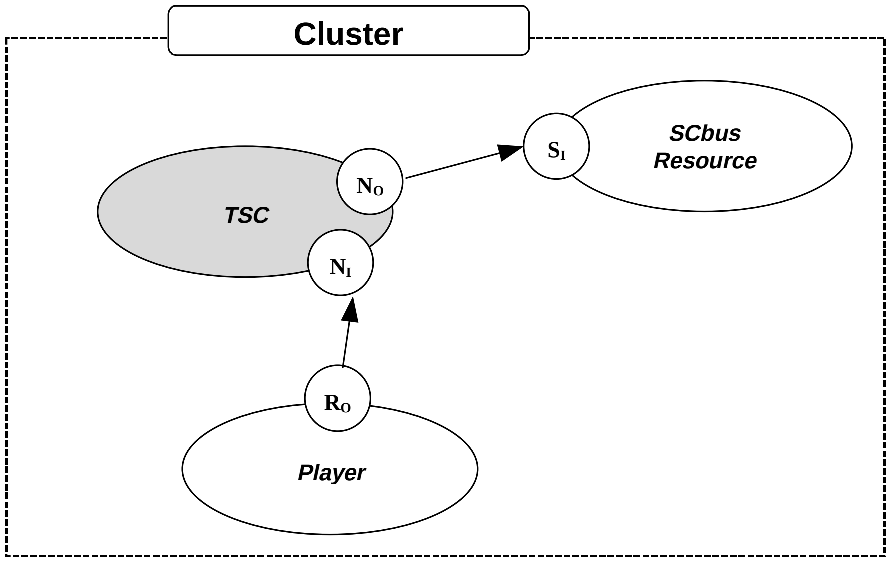


**Figure 18. Default Cluster Connections Example**

It may be desirable to configure the player resource to output to the $\mathbf{S_I}$ port as well as the $\mathbf{N_I}$ port temporarily in a drop and insert situation. To establish a connection between the $\mathbf{R_O}$ and $\mathbf{S_I}$ ports, call *nn***ClusterConnect( )** specifying the cluster, $\mathbf{S_I}$ port, and $\mathbf{R_O}$ port. This results in a new connection map as shown in Figure 19.

To return to the original connection map, call *nn***ClusterDisconnect( )** specifying the cluster, $\mathbf{S_I}$ port and $\mathbf{R_O}$ port.

### 4.2.6. Finding Cluster Assignment

To find the cluster that a component instance is part of, call:

| | |
|---|---|
| *nn***ClusterbyComp( )** | Given an instance descriptor, finds the cluster to which it is allocated. |

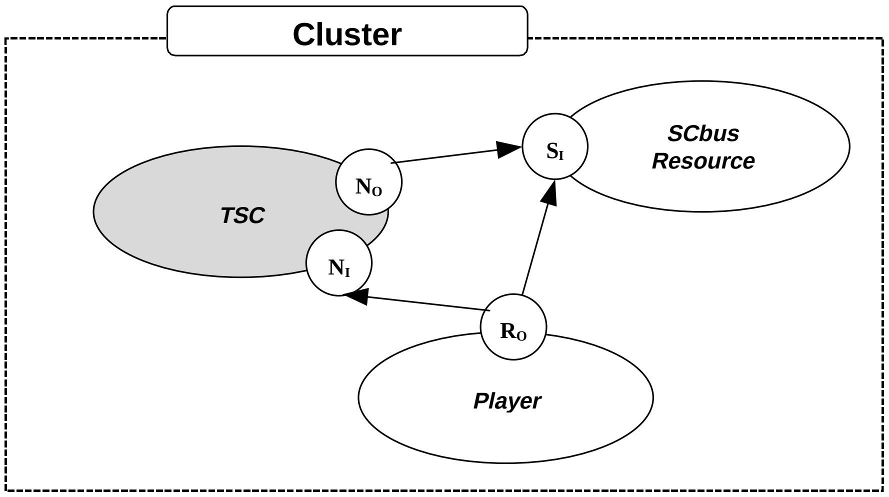


**Figure 19. Reconfigured Cluster**

### 4.2.7. Connecting Ports on the Same Board

Clusters on the same board can sometimes be connected without using SCbus (or CT Bus) timeslots. To connect two clusters together, use the *nn***ClusterConnect( )** function. If the cluster ports specified can be connected without SCbus (or CT Bus) timeslots, the function will succeed. If they cannot, the connection will fail.

# Index

## NOTES

## NOTES