Abstract

A Flexible Data Recorder Architecture

A system architecture for a low-cost digital data recorder is described. The recorder is based on an Intel PC hardware platform and the Windows 95 software environment. The hardware architecture is based on Commercial-Off-The-Shelf components with the exception of the physical interfaces to the recorder. This architecture provides a low life cycle cost for the recorder and offers a flexible migration path as newer and faster storage peripherals become available. The software follows a distributed process, object oriented architecture that easily supports the addition of physical interfaces, transports, and analysis tools to the recorder

Topics

- Development goals
- System architecture
- Hardware architecture
- Software architecture
- Development challenges

Development Goals

- Requirements
  - No calibration or complex periodic maintenance
  - Low life-cycle cost
  - Unattended operation
  - Suitable for long term archive of data
- Design Goals
  - Use commercial tape transports
  - Scaleable architecture
    - New transports
    - New interfaces
    - Faster bit rates
  - User friendly
  - Enhanced features

Development Goals

The recorder was developed initially for the Meteorological Satellite Ground Station market. These satellites have telemetry links running in the range of 500 kbps to 2 Mbps.

At that time a set of requirements was formulated based on our experiences in designing and building satellite ground stations and using the currently available recorders:

- No calibration or complex periodic maintenance - Many of our customers are in third world countries with little infrastructure to support these activities.
- Low life-cycle cost - proprietary media for many recorders is expensive and not easily procured in many countries.
- Unattended operation - ground stations acquire data around the clock and many run unattended overnight.
- Capability to store all required information to process the telemetry data on the media - in many systems ancillary data is required by the processing system when processing the telemetry data. We wanted a way to record this data along with the telemetry for a completely self contained archive.

A set of design goals was formulated in order to support making the recorder a viable and supportable product:

Development Goals

- Use commercial tape transports and not to modify the transports in any way if possible. We do not have the expertise to design and manufacture tape transports - nor any desire to do so.
- Scaleable Architecture - to be competitive we must respond quickly to customer requests for new technologies.
  - Faster and bigger tape transports are being announced at regular intervals
  - Newer storage technologies are becoming available - e.g. magneto-optical.
  - Record data direct to disk.
  - Handle serial data rates from 1 kbps to >30 Mbps
- In order to differentiate our product in the market place we felt we needed a unique, powerful, flexible, and yet easy to use interface between the users and the recorder.
- A further differentiating factor would be to use the computer in the recorder to support the addition of features to provide telemetry testing capability. We envisioned telemetry data synthesis, telemetry analysis on the record in either real-time or on the recorded data. We wished to provide data import and export in standard file formats on floppy disk or other file structured media.

System Architecture

The recorder architecture is based around a set of data transfer buffers. These buffers are managed by software that allows applications to use them to pass data to each other. These buffers are used to support both the inter-process communication and DMA to physical devices.

The hardware is designed to use these buffers for performing I/O to the media and to telemetry equipment.

The software takes advantage of the buffers by allowing many different applications to use the buffers to exchange data.

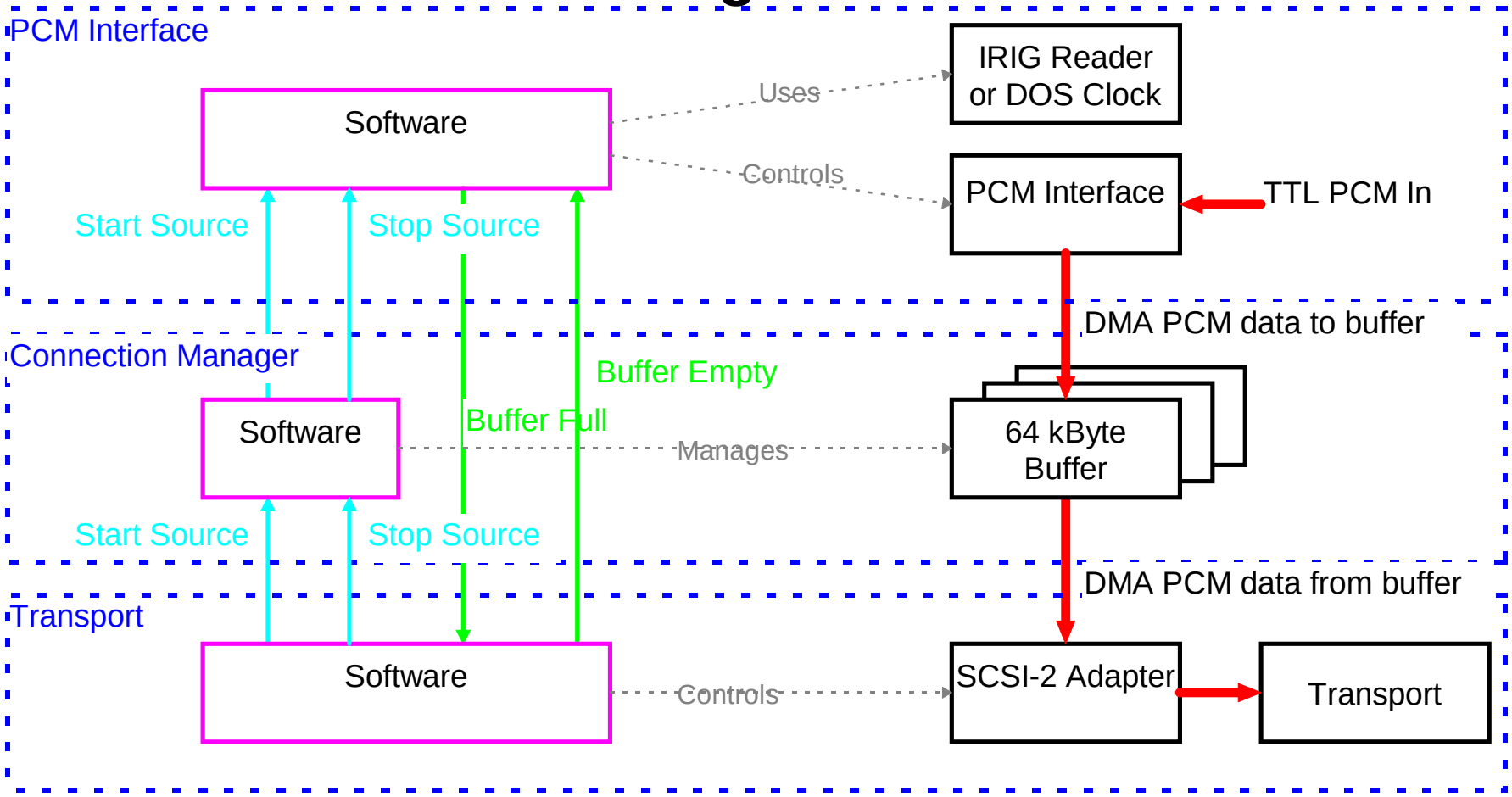
System Architecture

The data transfer buffers architecture is a very powerful tool. With it we can:

- Connect a Transport application to a PCM Interface application to provide the functionality of a tape recorder.
- Connect two Transport applications to copy data between media.
- Connect the DOS File Interface application to a Transport application and import or export PCM data in standard DOS file format.
- Connect the Satellite Simulator application to a Transport application and record synthesized satellite telemetry to tape.
- Connect the Satellite Simulator application to a PCM Interface application and play synthesized satellite telemetry directly out the PCM Interface.

System Architecture

Recording PCM Data



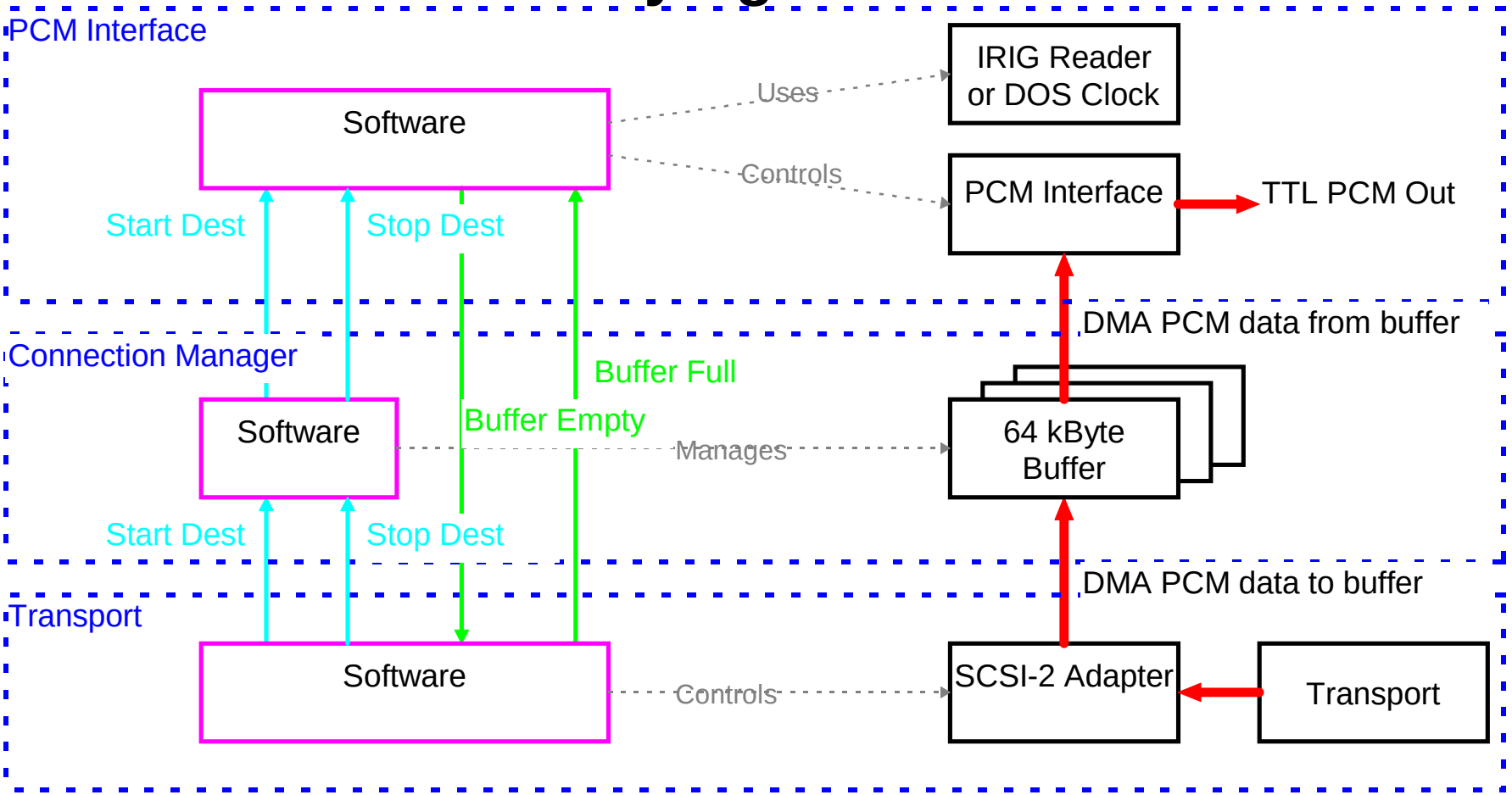
System Architecture

Recording PCM data:

1. The PCM Interface application is started.
2. The Transport application is started.
3. A Data Path is established from the PCM Interface to the Transport.
4. The user presses **record** - the Transport sends a Start Source message to the Connection Manager, the Connection Manager passes the message to the PCM Interface.
5. The PCM Interface places time tagged data in a buffer and sends a Buffer Full message to the Transport.
6. The Transport writes the data to the media and sends a Buffer Empty message to the PCM Interface.
7. Steps 5 and 6 are repeated until the user selects **stop**, the end of media is reached, or an error occurs.

System Architecture

Playing PCM Data



System Architecture

Playing PCM data:

1. The PCM Interface application is started.
2. The Transport application is started.
3. A Data Path is established from the Transport to the PCM Interface.
4. The user presses **play** - the Transport sends a Start Destination message to the Connection Manager, the Connection Manager passes the message to the PCM Interface.
5. The Transport reads the data from the media and sends a Buffer Full message to the PCM Interface.
6. The PCM Interface outputs the data and then sends a Buffer Empty message to the Transport.
7. Steps 5 and 6 are repeated until the user selects **stop**, the end of the data is reached, or an error occurs.

System Architecture

Media Structure

- The media is structured with each recording session being a Data Set that contains a header followed by the PCM data.
- The media has a directory of all Data Sets that is read and displayed when the media is loaded.
- PCM data is recorded on the media in 64 kByte blocks. Each block has an 8 byte header that contains the time stamp of the last word in the block and the channel Id.
- The media structure is proprietary as there was no standard at the time of development.
- Different transports all implement the same structure.

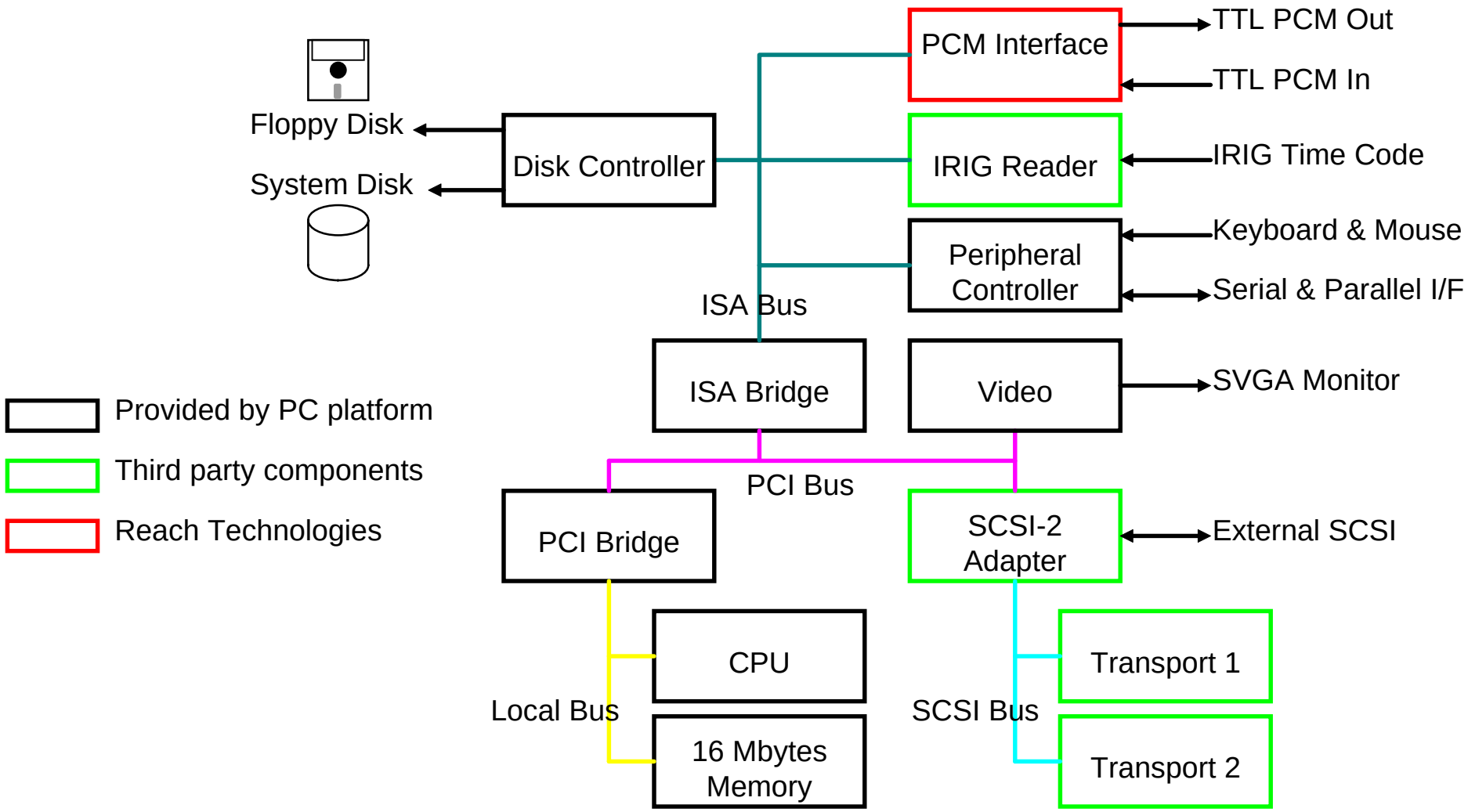
Hardware Architecture

The recorder is based on a standard “Wintel” Personal Computer platform.

- Microsoft Windows 95 operating system
- Intel 80x86 or Pentium processor
- Standard peripheral devices
  (disk, mouse, keyboard, etc.)

This allows us to use low cost (high production volume) personal computers, and provides for powerful software and hardware development tools at reasonable prices.

Hardware Architecture



Hardware Architecture

All system components except the PCM Interface are Commercial Off the Shelf (COTS). This reduces life cycle cost by allowing the customer to replace components in the field from local suppliers.
In the portable version of the recorder there is no PCI bus, all components shown on the PCI bus are located on the ISA bus.
The serial ports provide remote control capability and Datum 9700 IRIG Time Code Translator emulation functionality.
The parallel port allows a standard PC printer to be connected. The printer is used to print tape content listings, tape labels, tape case liners, and operations schedules.

Software Architecture

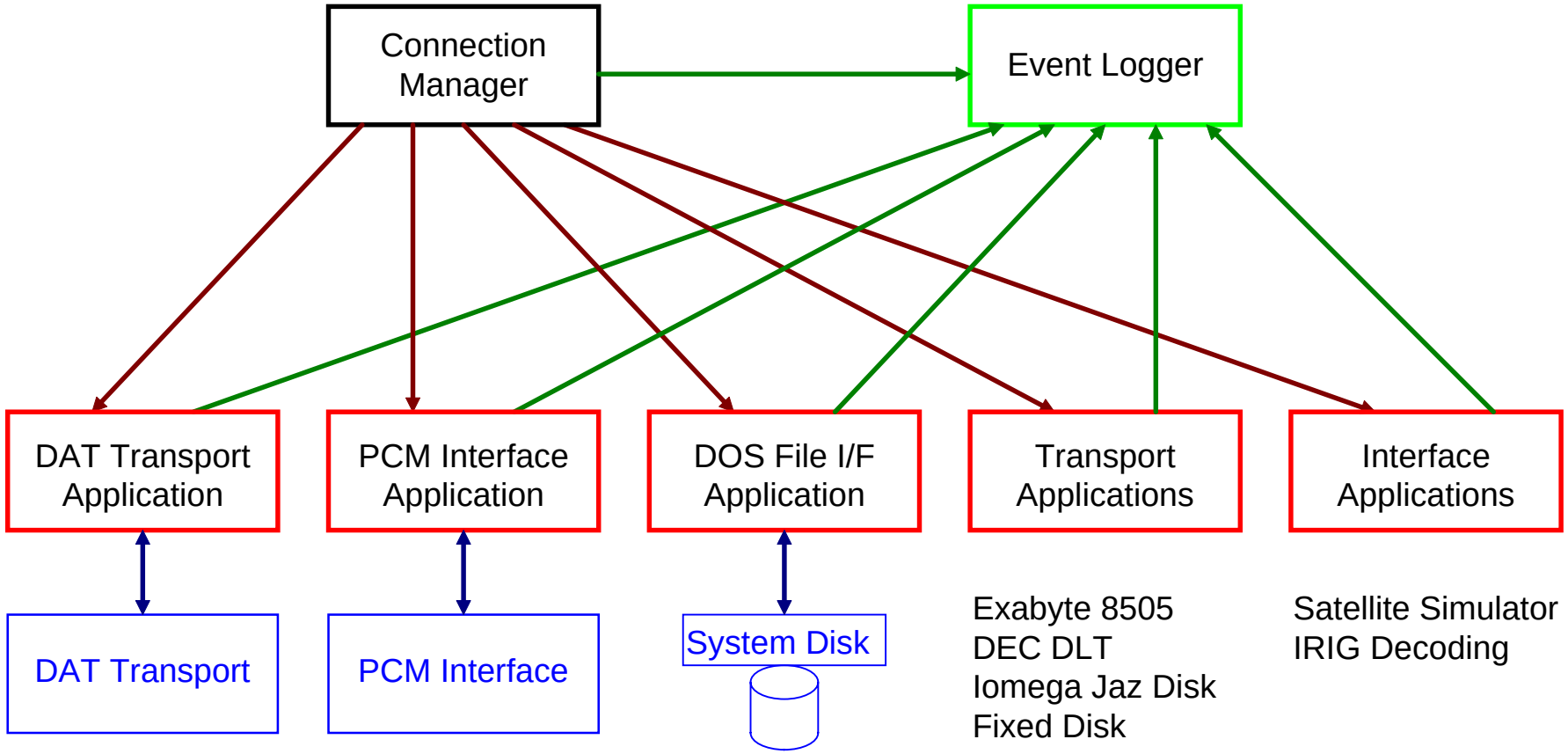
In order to meet the requirements and design goals described we developed a distributed process architecture and implemented it using object oriented rapid application development (RAD) tools.

- One process (Windows application) per transport / interface.
- Inter-process communication (Connection Manager)
- Centralized event logging (Event Logger)
- Utilize Windows support for distributed objects (OLE/COM)

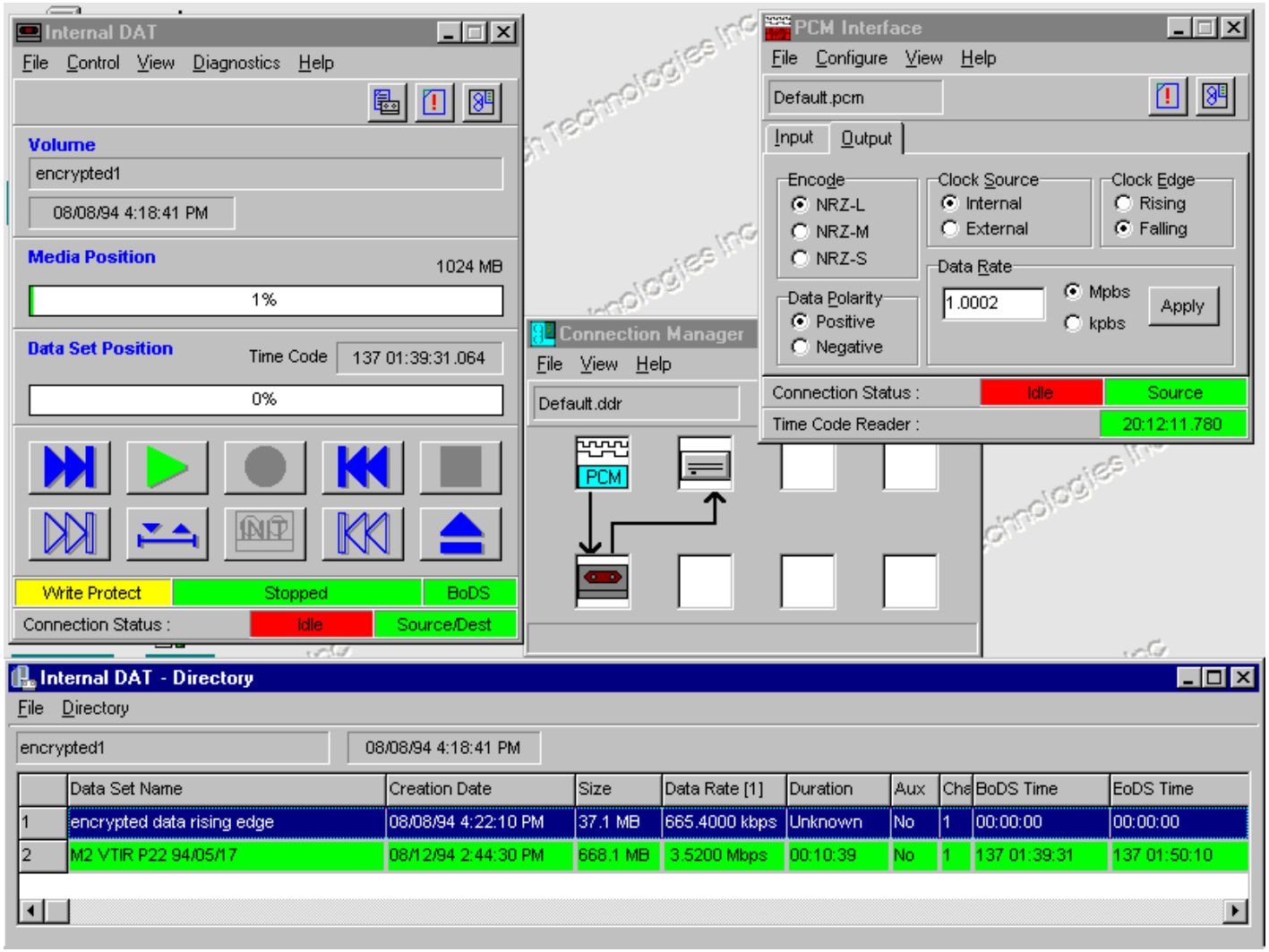
The implementation of the Inter-Process communications is designed for the high speed transfer of large blocks of data. It runs at over 100 Mbps on a 120 MHz Pentium system.

Software Architecture

Distributed Application Environment



Software Architecture



Software Architecture

Connection Manager

- Controls the assignment of connections (“Data Paths”) between applications.
- Passes data transfer control messages between applications.
- Graphically displays the recorder configuration.
- Controls the saving and restoring of test configurations - saves the state of each application and the window positions for later restoration.

Software Architecture

Event Logger

- Used by all applications to display and log events
- Provide consistent application behavior when running in scheduled and remote mode. In these modes no user input can be requested in error situations.
- Used to provide customer support when errors occur - users can print the event log and fax it to us, or copy it to diskette and email it to us for help with problems.

Software Architecture

Transport Applications

- Control the transport for Recording, Playing, and Positioning.
- One application for each transport in the recorder.
- Provide a consistent user interface and operational paradigm across different transports.
- Provides user control of the data transfer.
- Allow the easy connection of external transports for transcription.

Software Architecture

Interface Applications

- An interface is a source and/or destination of data.
- Can control physical devices.
  (e.g. PCM Interface)
- Can control software data sources.
  (e.g. Satellite Simulator)
- Can control software data destinations.
  (e.g. IRIG decoder)
- May or may not provide user control of the data transfer.
- If an interface provides user control of the data exchange it can connect to another interface.

Development Challenges

Implementing the recorder under the Microsoft Windows 95 operating system presented several challenges that had to be overcome:

- Virtual Memory
- Interrupt Latency
- Inter-process Communications
- Low Bit Rates

Development Challenges

Virtual Memory

Windows 95 is a virtual memory based operating system. This means that any large memory blocks allocated by software are likely not to be allocated from contiguous physical memory. This has a large impact on the performance of our DMA transfers.

Windows uses a 4kByte page size and we use 4 kByte buffers. If these buffers are allocated by Windows they may have up to 16 fragments. Each fragment requires a separate DMA transfer with associated overhead. The overhead is caused by interrupt servicing latency and programming the DMA controller for the next transfer.

Development Challenges

Virtual Memory

To solve this problem we have implemented a device driver that loads at system boot and allocates the buffer memory from system physical memory before Windows 95 maps it. The driver performs three functions:

- Provides physically contiguous memory for the DMA buffers
- Provides physical addresses for programming the DMA controllers
- Maps the memory and provides virtual addresses to applications to allow software access to the data.

Development Challenges

Interrupt Latency

Windows virtualizes all hardware devices to allow the sharing of system resources between applications. This introduces sometimes severe delays in interrupt servicing. The solution was to design buffering into the PCM Interface card to handle the latency, and to implement a Virtual Device Driver (VxD) to reduce the interrupt latency.

We have implemented 64 kbits of buffering in the PCM Interface that allows us to handle interrupt latencies of up to 6.4 mSec when running at 10 Mbps.

The VxD allows us to immediately program the DMA controller upon receiving the DMA completion interrupt.

Development Challenges

Inter-process Communications

The standard Windows Inter-process communication features are not suitable for high volume low latency data transfers.

In order to handle real time buffer exchange we had to implement our own inter-process communications protocol. This communications protocol is based on the large memory buffers and windows messages passed between applications.

An application will receive a message from its partner application telling it that a buffer is available for processing. When the application has finished with the buffer it sends a message to the partner application to let it know the buffer is ready.

This protocol is encapsulated in a RAD visual component that makes development of new applications very fast and easy.

Development Challenges

Low Bit Rates

The recorder was initially developed for bit rates from 500 kbps to 2 Mbps. The architecture was optimized for data transfers at these rates. The 64 kByte buffer size is a result of this.

We have since been requested to provide solutions for data rates as low as 1 kbps. At this rate a single buffer contains almost 7 minutes of data. It is unacceptable to require several minutes of recording or playing between pressing the stop button and actually completing the operation.

We had to cleanly handle transfers consisting not only of a single buffer, but of partially filled buffers as well. This required modifications to the device drivers the inter-process communications protocol and the applications themselves.

A Flexible Data Recorder Architecture

Glenn Jones, Reach Technologies Inc
Phone: +1-604-220-6261
Fax: +1-604-597-2282
glenn@reach.bc.ca

Presented at the THIC meeting in Seattle WA
January 21, 1997

Abstract

A Flexible Data Recorder Architecture

A system architecture for a low-cost digital data recorder is described. The recorder is based on an Intel PC hardware platform and the Windows 95 software environment. The hardware architecture is based on Commercial-Off-The-Shelf components with the exception of the physical interfaces to the recorder. This architecture provides a low life cycle cost for the recorder and offers a flexible migration path as newer and faster storage peripherals become available. The software follows a distributed process, object oriented architecture that easily supports the addition of physical interfaces, transports, and analysis tools to the recorder

Topics

- Development goals
- System architecture
- Hardware architecture
- Software architecture
- Development challenges

Development Goals

- Requirements
  - No calibration or complex periodic maintenance
  - Low life-cycle cost
  - Unattended operation
  - Suitable for long term archive of data
- Design Goals
  - Use commercial tape transports
  - Scaleable architecture
    - New transports
    - New interfaces
    - Faster bit rates
  - User friendly
  - Enhanced features

Development Goals

The recorder was developed initially for the Meteorological Satellite Ground Station market. These satellites have telemetry links running in the range of 500 kbps to 2 Mbps.

At that time a set of requirements was formulated based on our experiences in designing and building satellite ground stations and using the currently available recorders:

- No calibration or complex periodic maintenance - Many of our customers are in third world countries with little infrastructure to support these activities.
- Low life-cycle cost - proprietary media for many recorders is expensive and not easily procured in many countries.
- Unattended operation - ground stations acquire data around the clock and many run unattended overnight.
- Capability to store all required information to process the telemetry data on the media - in many systems ancillary data is required by the processing system when processing the telemetry data. We wanted a way to record this data along with the telemetry for a completely self contained archive.

A set of design goals was formulated in order to support making the recorder a viable and supportable product:

Development Goals

- Use commercial tape transports and not to modify the transports in any way if possible. We do not have the expertise to design and manufacture tape transports - nor any desire to do so.
- Scaleable Architecture - to be competitive we must respond quickly to customer requests for new technologies.
  - Faster and bigger tape transports are being announced at regular intervals
  - Newer storage technologies are becoming available - e.g. magneto-optical.
  - Record data direct to disk.
  - Handle serial data rates from 1 kbps to >30 Mbps
- In order to differentiate our product in the market place we felt we needed a unique, powerful, flexible, and yet easy to use interface between the users and the recorder.
- A further differentiating factor would be to use the computer in the recorder to support the addition of features to provide telemetry testing capability. We envisioned telemetry data synthesis, telemetry analysis on the record in either real-time or on the recorded data. We wished to provide data import and export in standard file formats on floppy disk or other file structured media.

System Architecture

The recorder architecture is based around a set of data transfer buffers. These buffers are managed by software that allows applications to use them to pass data to each other. These buffers are used to support both the inter-process communication and DMA to physical devices.

The hardware is designed to use these buffers for performing I/O to the media and to telemetry equipment.

The software takes advantage of the buffers by allowing many different applications to use the buffers to exchange data.

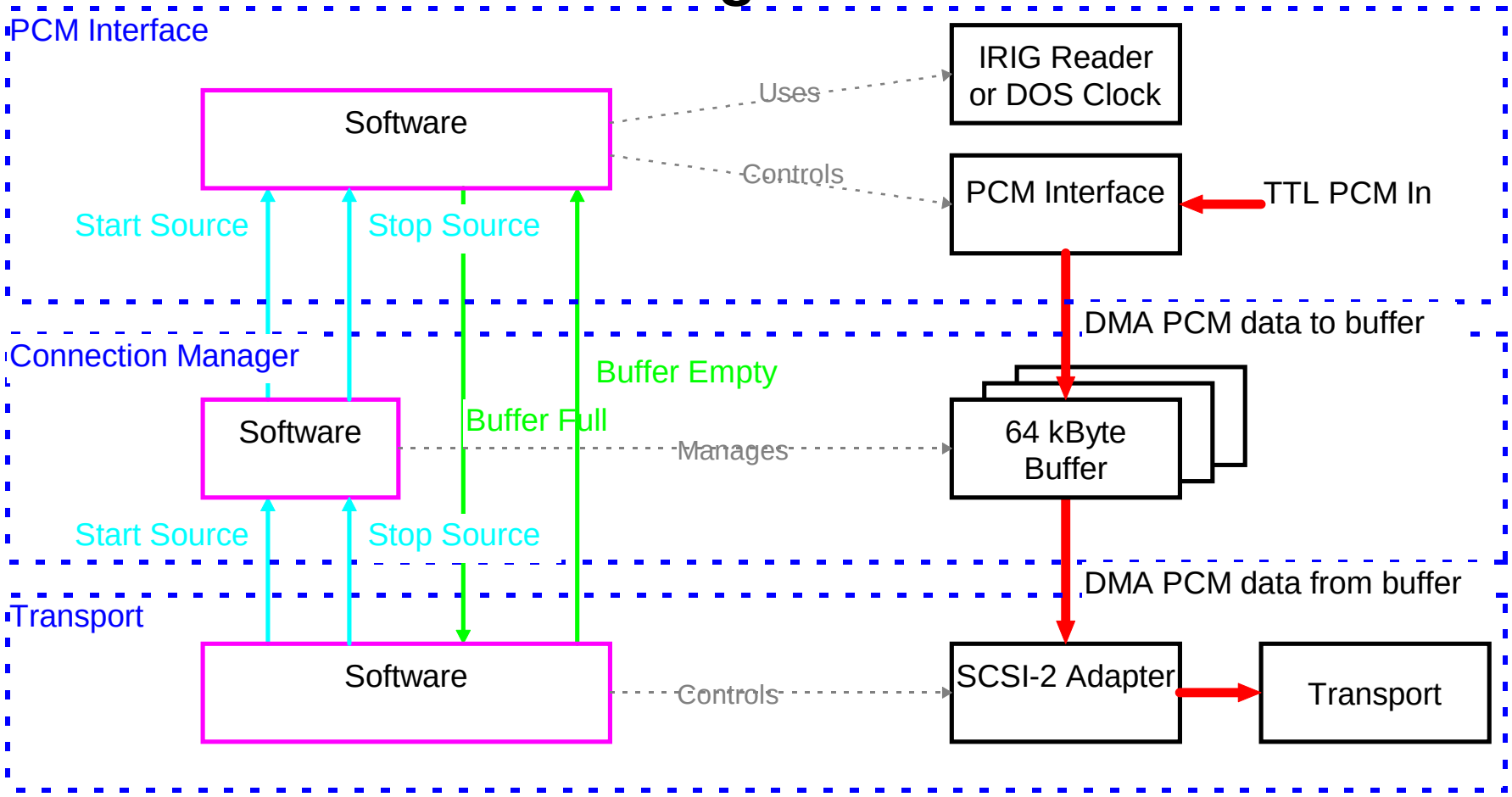
System Architecture

The data transfer buffers architecture is a very powerful tool. With it we can:

- Connect a Transport application to a PCM Interface application to provide the functionality of a tape recorder.
- Connect two Transport applications to copy data between media.
- Connect the DOS File Interface application to a Transport application and import or export PCM data in standard DOS file format.
- Connect the Satellite Simulator application to a Transport application and record synthesized satellite telemetry to tape.
- Connect the Satellite Simulator application to a PCM Interface application and play synthesized satellite telemetry directly out the PCM Interface.

System Architecture

Recording PCM Data



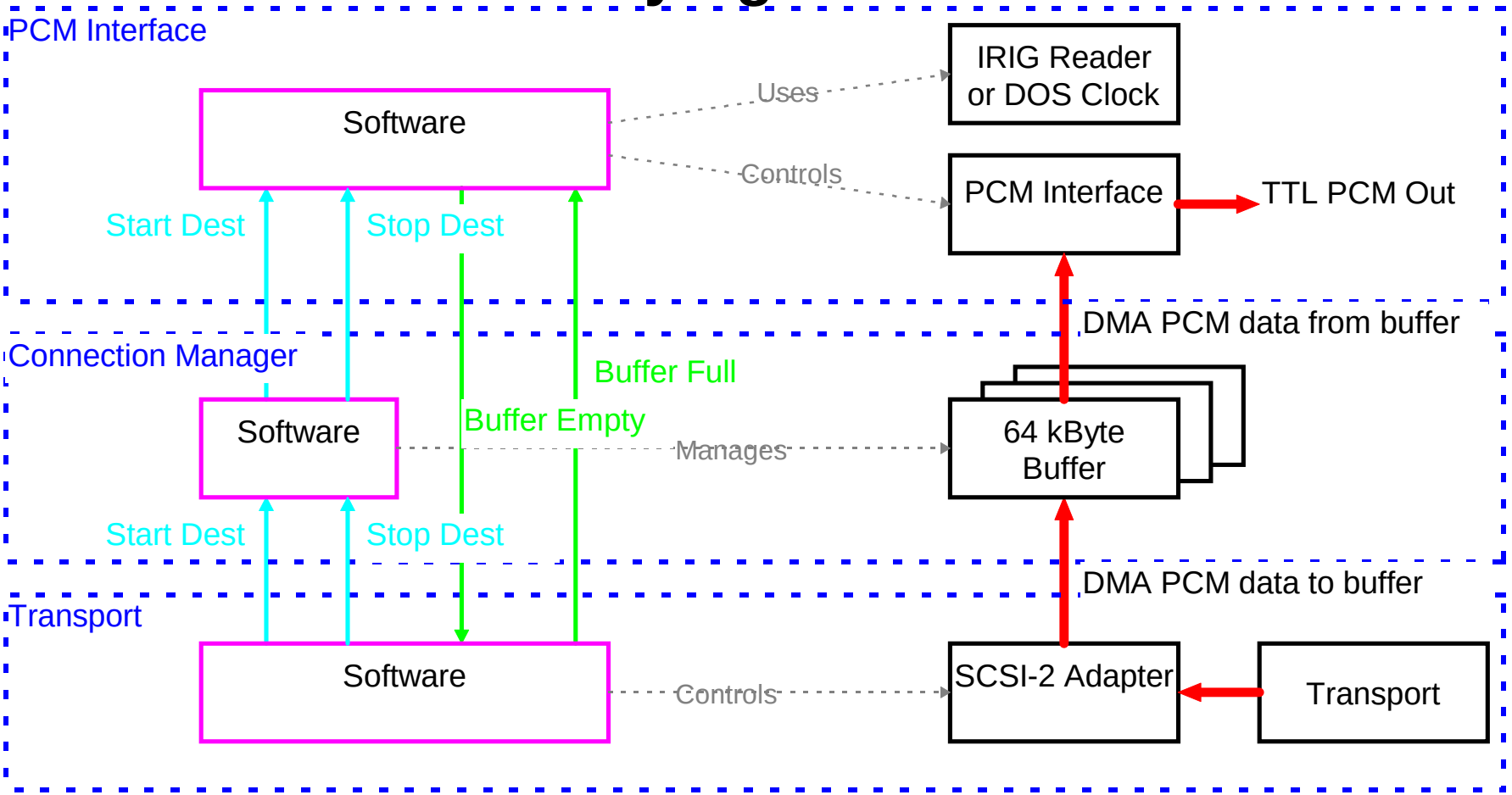
System Architecture

Recording PCM data:

1. The PCM Interface application is started.
2. The Transport application is started.
3. A Data Path is established from the PCM Interface to the Transport.
4. The user presses **record** - the Transport sends a Start Source message to the Connection Manager, the Connection Manager passes the message to the PCM Interface.
5. The PCM Interface places time tagged data in a buffer and sends a Buffer Full message to the Transport.
6. The Transport writes the data to the media and sends a Buffer Empty message to the PCM Interface.
7. Steps 5 and 6 are repeated until the user selects **stop**, the end of media is reached, or an error occurs.

System Architecture

Playing PCM Data



System Architecture

Playing PCM data:

1. The PCM Interface application is started.
2. The Transport application is started.
3. A Data Path is established from the Transport to the PCM Interface.
4. The user presses **play** - the Transport sends a Start Destination message to the Connection Manager, the Connection Manager passes the message to the PCM Interface.
5. The Transport reads the data from the media and sends a Buffer Full message to the PCM Interface.
6. The PCM Interface outputs the data and then sends a Buffer Empty message to the Transport.
7. Steps 5 and 6 are repeated until the user selects **stop**, the end of the data is reached, or an error occurs.

System Architecture

Media Structure

- The media is structured with each recording session being a Data Set that contains a header followed by the PCM data.
- The media has a directory of all Data Sets that is read and displayed when the media is loaded.
- PCM data is recorded on the media in 64 kByte blocks. Each block has an 8 byte header that contains the time stamp of the last word in the block and the channel Id.
- The media structure is proprietary as there was no standard at the time of development.
- Different transports all implement the same structure.

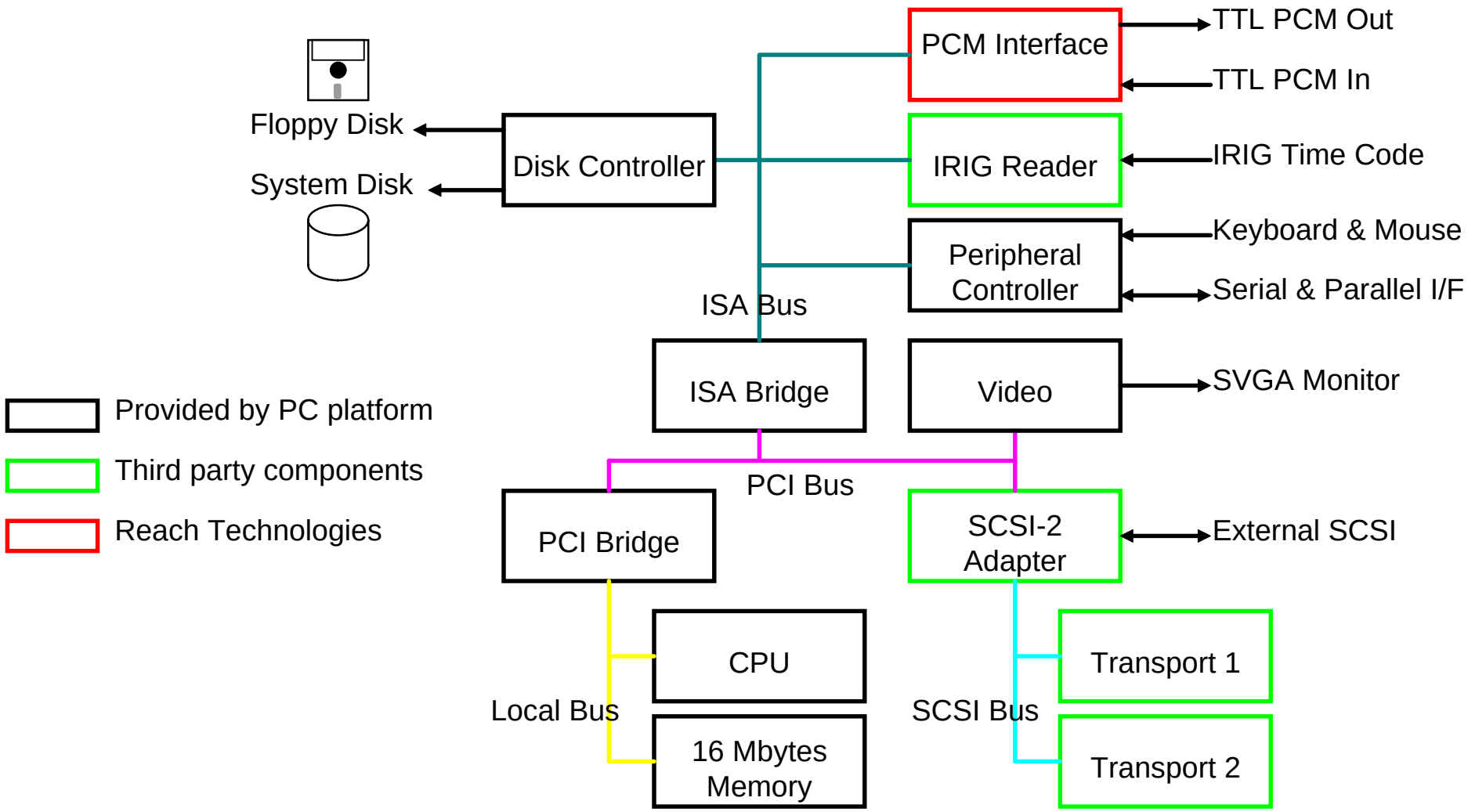
Hardware Architecture

The recorder is based on a standard “Wintel” Personal Computer platform.

- Microsoft Windows 95 operating system
- Intel 80x86 or Pentium processor
- Standard peripheral devices
  (disk, mouse, keyboard, etc.)

This allows us to use low cost (high production volume) personal computers, and provides for powerful software and hardware development tools at reasonable prices.

Hardware Architecture



Hardware Architecture

All system components except the PCM Interface are Commercial Off the Shelf (COTS). This reduces life cycle cost by allowing the customer to replace components in the field from local suppliers.
In the portable version of the recorder there is no PCI bus, all components shown on the PCI bus are located on the ISA bus.
The serial ports provide remote control capability and Datum 9700 IRIG Time Code Translator emulation functionality.
The parallel port allows a standard PC printer to be connected. The printer is used to print tape content listings, tape labels, tape case liners, and operations schedules.

Software Architecture

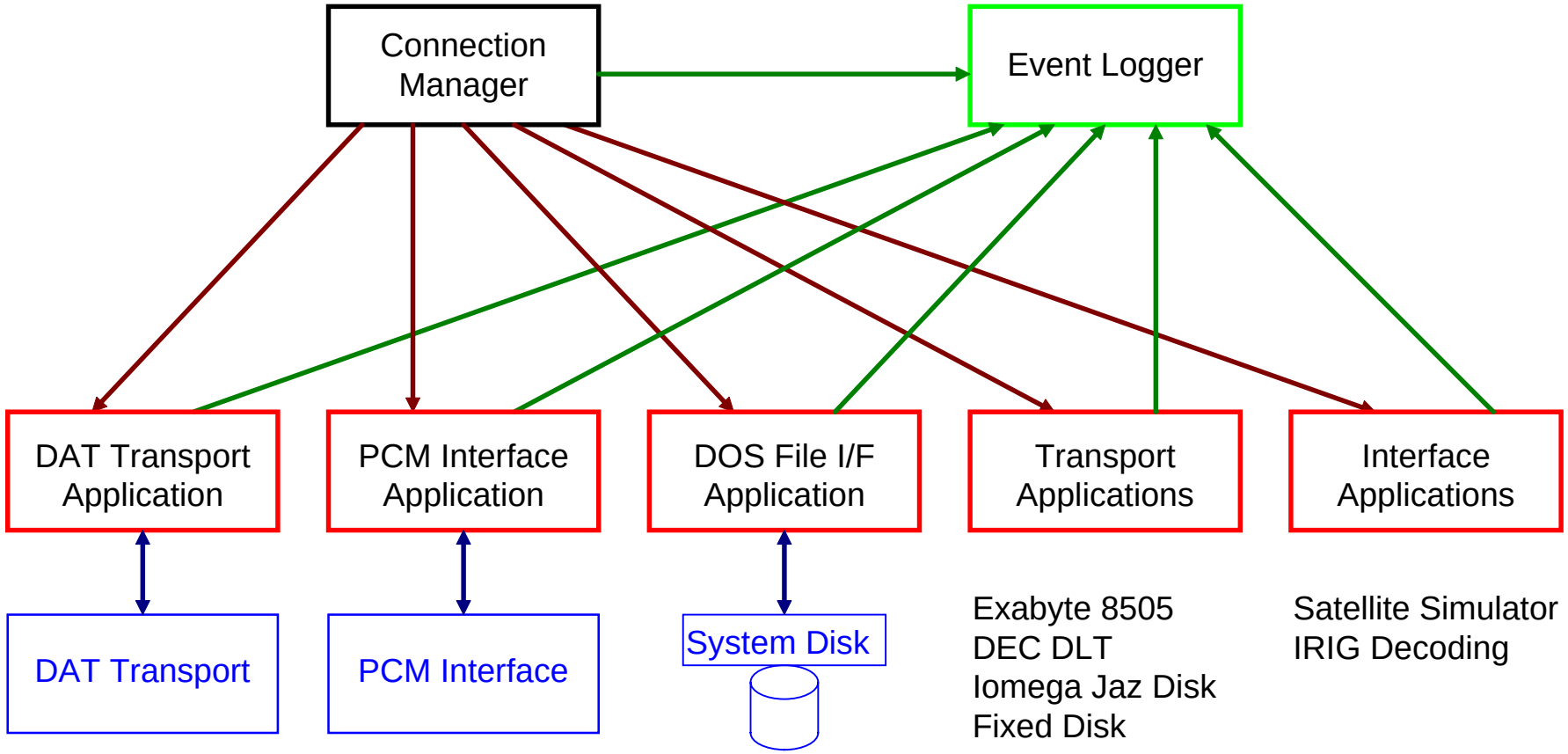
In order to meet the requirements and design goals described we developed a distributed process architecture and implemented it using object oriented rapid application development (RAD) tools.

- One process (Windows application) per transport / interface.
- Inter-process communication (Connection Manager)
- Centralized event logging (Event Logger)
- Utilize Windows support for distributed objects (OLE/COM)

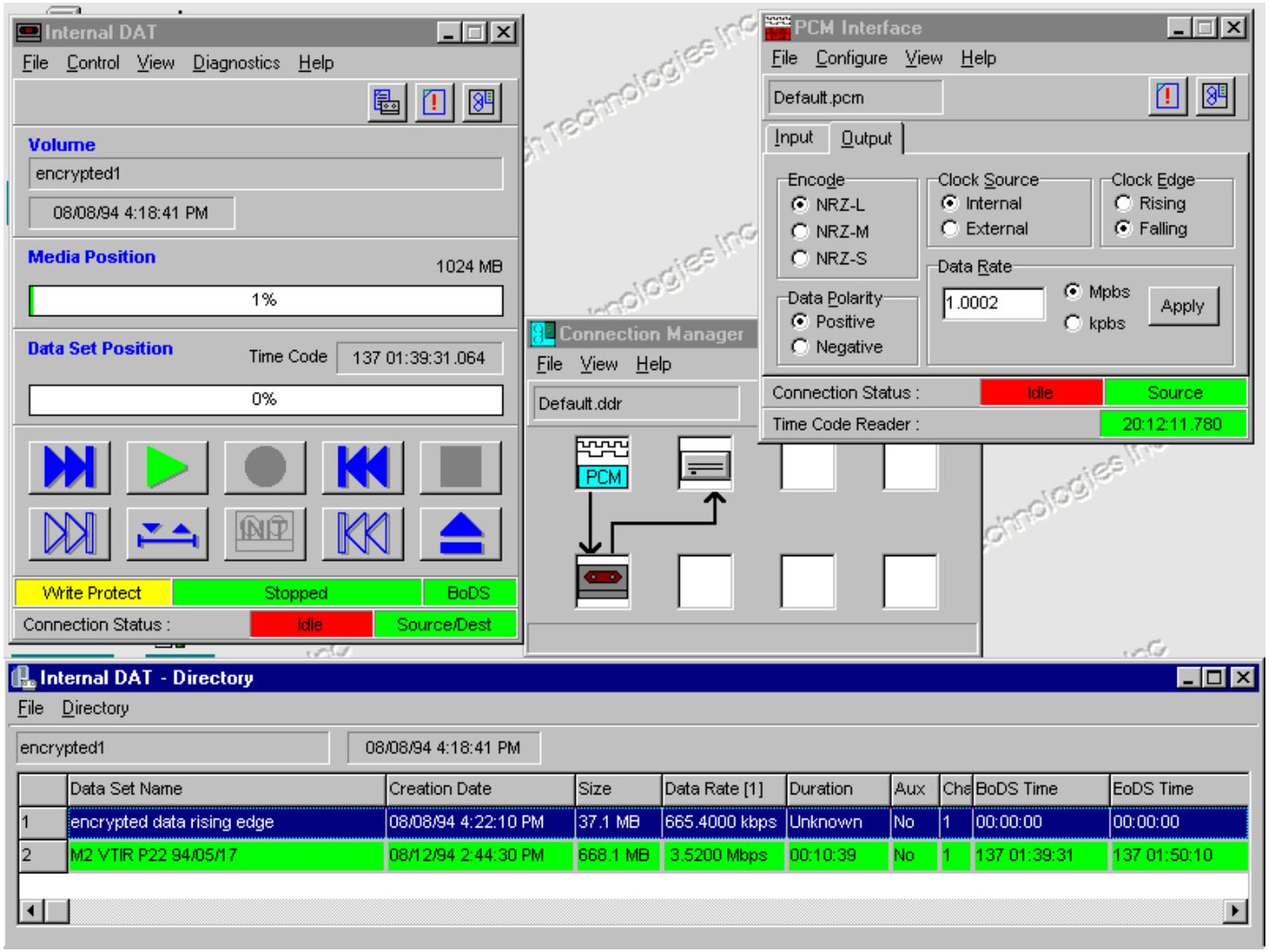
The implementation of the Inter-Process communications is designed for the high speed transfer of large blocks of data. It runs at over 100 Mbps on a 120 MHz Pentium system.

Software Architecture

Distributed Application Environment



Software Architecture



Software Architecture

Connection Manager

- Controls the assignment of connections (“Data Paths”) between applications.
- Passes data transfer control messages between applications.
- Graphically displays the recorder configuration.
- Controls the saving and restoring of test configurations - saves the state of each application and the window positions for later restoration.

Software Architecture

Event Logger

- Used by all applications to display and log events
- Provide consistent application behavior when running in scheduled and remote mode. In these modes no user input can be requested in error situations.
- Used to provide customer support when errors occur - users can print the event log and fax it to us, or copy it to diskette and email it to us for help with problems.

Software Architecture

Transport Applications

- Control the transport for Recording, Playing, and Positioning.
- One application for each transport in the recorder.
- Provide a consistent user interface and operational paradigm across different transports.
- Provides user control of the data transfer.
- Allow the easy connection of external transports for transcription.

Software Architecture

Interface Applications

- An interface is a source and/or destination of data.
- Can control physical devices.
  (e.g. PCM Interface)
- Can control software data sources.
  (e.g. Satellite Simulator)
- Can control software data destinations.
  (e.g. IRIG decoder)
- May or may not provide user control of the data transfer.
- If an interface provides user control of the data exchange it can connect to another interface.

Development Challenges

Implementing the recorder under the Microsoft Windows 95 operating system presented several challenges that had to be overcome:

- Virtual Memory
- Interrupt Latency
- Inter-process Communications
- Low Bit Rates

Development Challenges

Virtual Memory

Windows 95 is a virtual memory based operating system. This means that any large memory blocks allocated by software are likely not to be allocated from contiguous physical memory. This has a large impact on the performance of our DMA transfers.

Windows uses a 4kByte page size and we use 4 kByte buffers. If these buffers are allocated by Windows they may have up to 16 fragments. Each fragment requires a separate DMA transfer with associated overhead. The overhead is caused by interrupt servicing latency and programming the DMA controller for the next transfer.

Development Challenges

Virtual Memory

To solve this problem we have implemented a device driver that loads at system boot and allocates the buffer memory from system physical memory before Windows 95 maps it. The driver performs three functions:

- Provides physically contiguous memory for the DMA buffers
- Provides physical addresses for programming the DMA controllers
- Maps the memory and provides virtual addresses to applications to allow software access to the data.

Development Challenges

Interrupt Latency

Windows virtualizes all hardware devices to allow the sharing of system resources between applications. This introduces sometimes severe delays in interrupt servicing. The solution was to design buffering into the PCM Interface card to handle the latency, and to implement a Virtual Device Driver (VxD) to reduce the interrupt latency.

We have implemented 64 kbits of buffering in the PCM Interface that allows us to handle interrupt latencies of up to 6.4 mSec when running at 10 Mbps.

The VxD allows us to immediately program the DMA controller upon receiving the DMA completion interrupt.

Development Challenges

Inter-process Communications

The standard Windows Inter-process communication features are not suitable for high volume low latency data transfers.

In order to handle real time buffer exchange we had to implement our own inter-process communications protocol. This communications protocol is based on the large memory buffers and windows messages passed between applications.

An application will receive a message from its partner application telling it that a buffer is available for processing. When the application has finished with the buffer it sends a message to the partner application to let it know the buffer is ready.

This protocol is encapsulated in a RAD visual component that makes development of new applications very fast and easy.

Development Challenges

Low Bit Rates

The recorder was initially developed for bit rates from 500 kbps to 2 Mbps. The architecture was optimized for data transfers at these rates. The 64 kByte buffer size is a result of this.

We have since been requested to provide solutions for data rates as low as 1 kbps. At this rate a single buffer contains almost 7 minutes of data. It is unacceptable to require several minutes of recording or playing between pressing the stop button and actually completing the operation.

We had to cleanly handle transfers consisting not only of a single buffer, but of partially filled buffers as well. This required modifications to the device drivers the inter-process communications protocol and the applications themselves.