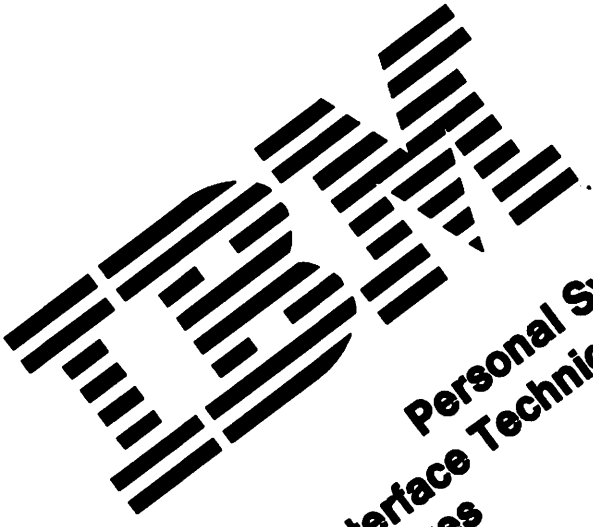


Personal System/2
Hardware Interface Technical Reference
- Architectures

84F8933

First Edition (October 1990)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

© Copyright International Business Machines Corporation 1989, 1990. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Special Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
Micro Channel
Personal Computer AT
Personal System/2
PS/2

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Hitachi	Hitachi Corporation
Intel	Intel Corporation
Motorola	Motorola, Incorporated

Preface

The Technical Reference library provides hardware and software interface information for IBM Personal System/2 products. The volumes in the library are:

IBM Personal System/2 Hardware Interface Technical Reference — Architectures:
This volume describes the architectures used with Personal System/2 products that are based on the Micro Channel architecture.

IBM Personal System/2 Hardware Interface Technical Reference — Common Interfaces:
This volume describes those devices and interfaces that are common to PS/2 systems. It includes the technical information describing devices, such as the serial port and parallel port controllers, and general information, such as the microprocessor instruction set and the characters associated with each keystroke.

IBM Personal System/2 Hardware Interface Technical Reference — System-Specific Information:
These technical references provides information concerning hardware implementation and performance information for specific models of PS/2 systems.

IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference:
This volume provides BIOS and Advanced BIOS interface information.

Option and Adapter Technical References:
These technical references provide hardware and programming information about individual PS/2 options and adapters.

© Copyright IBM Corp. 1989, 1990

This volume is divided into the following:

"Micro Channel Architecture" describes the Micro Channel signals, timing, electrical characteristics, interrupts, and arbitration.

"Micro Channel Adapter Design" provides information such as adapter dimensions and design guidelines.

"Setup Architecture" describes adapter setup, system configuration utilities, and optional configuration procedures used with PS/2 systems.

Suggested Reading:

- *BASIC for the IBM Personal Computer*
- IBM *Disk Operating System (DOS)*
- IBM *Operating System/2*
- *Macro Assembler for the IBM Personal Computer*

Warning: In this technical reference, the term "reserved" is used to describe certain signals, bits, and registers. Use of reserved areas can cause compatibility problems, loss of data, or permanent damage to the hardware.

When modifying a register, the state of the reserved bits must be preserved. When possible, read the register first and change only the bits required.

Note: The vertical bars in the left margin indicate technical changes. The technical changes consist of additions or corrections to the previous information. Only the changes from the last release of that specific technical reference are indicated.

Micro Channel Architecture

Description 1
Micro Channel Participants 2
Masters 2
Slaves 3
Micro Channel Structure 5
Addressing Model 6

Micro Channel Buses and Signals 7
Arbitration Bus and Associated Signals 7
Address Bus and Associated Signals 9
Address Signal Combinations 12
Data Bus and Associated Signals 13
Interrupt Signals 18
Other Micro Channel Signals 19
Channel Signal Groups (Digital) 20
Matched-Memory Extension Signal Descriptions 23
Auxiliary Video Extension Signal Descriptions 23
Micro Channel Architecture Base and Extensions 24
32-Bit Connector and Extensions 25
16-Bit Connector and Extensions 26

Micro Channel Procedures 27
Arbitration 27
Channel Ownership 27
Central Arbitration Control Point 28
Local Arbiters 30
Burst Transfer 32
Preemption 33
Fairness Feature and the Inactive State 34
Arbitration Timing 35
System Master and Arbitration 39
Data Transfer 40
Basic Transfer 40
Basic Transfer Procedure 41
Abort Cycle 43
Default Cycle Timing (200 ns Minimum) 43
Synchronous-Extended Cycle Timing (300 ns Minimum) 47
Asynchronous-Extended Cycle Timing (≥300 ns) 50
Data Steering 53
Data Steering Control 54
Master Replication of Data Bytes 55
Channel Support 55

© Copyright IBM Corp. 1989, 1991

Address Bus Translator 55
Central Data Steering 55
TR 32 56
Central Steering Logic 56
Central Translator Logic 56
-BE(0-3) 56
Streaming Data Transfer 58
Streaming Data Procedure (16- or 32-Bit) 59
Streaming Data Procedure (64-Bit) 60
Streaming Data Transfer Rates 62
Address Bus Management 62
Address Boundary Alignment 63
Streaming Data Cycles 65
64-Bit Streaming Data Cycles 67
Starting the Cycle 69
Deferring the Start of the Cycle 69
Pacing the Data 71
Terminating the Cycle 73
Streaming Data Timing 76
Direct Memory Access Procedure 83
Read and Write Operations 83
DMA Slave Selection 83
Address During DMA Operation 83
Transfer Data Length 84
DMA Procedure 84
Single Transfer 84
Burst Transfer (Multiple Transfer Cycles) 86
DMA Terminating Conditions 86
Use of -BURST 87
DMA Timings 88
Single DMA Transfer (DMA-Controller Controlled) 88
Burst DMA Transfer (DMA-Controller Terminated) 90
Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns) 92
Burst DMA Transfer (DMA-Slave Terminated - Synchronous-Extended Cycle, 300 ns) 94
Burst DMA Transfer (DMA Slave Terminated - Asynchronous-Extended Cycle, ≥300 ns) 96
Interrupt Procedures 98
Refresh 100
System Configuration 101
Multiple Card Adapters 102
Adapter Configuration Data 103
Required Configuration Fields 103
Adapter ID Field 104
Card Enable Field 104

Channel Check Field 104
Conditionally-Required Configuration Fields 104
Adapter I/O Address Select Field 105
Adapter ROM/RAM Address Select Field 105
Arbitration Level Field 105
Auto-Incrementing Enable Field 105
Channel Check Exception Reporting Field 106
Channel Check Status Indicator Field 106
Data and Address Parity Enable Field 106
Device I/O Address Assignment 106
Matched Memory Enable Field 107
Selected Feedback Return Exception Enable Field 107
Streaming Data Enable Field 108
Optional Configuration Fields 108
Option Select Data Field 108
Fairness Feature Enable Field 108
POS Subaddress Extension Field 108
Status Information Field 111
System Configuration Procedure 112
Address Bus During Setup Cycles 112
Card Enable Field During Setup Cycles 113
Channel Check Status During Setup Cycles 113
-CD SETUP During Setup Cycles 113
CD CHRDY During Setup Cycles 113
System Configuration Timing Diagrams 114
Exception Condition Reporting and Handling 118
Exception Conditions 118
Channel Data and Address Parity Exceptions 119
Data Parity 120
Read Operations - Basic Transfer 120
Read Operations - Streaming Data Transfer 122
Write Operations - Basic Transfer 122
Write Operations - Streaming Data Transfer 124
Data Parity Exception Handling 126
Bus Master 126
System Master 126
DMA Controller 126
Slaves 126
Address Parity 127
Channel Non-Parity Exceptions 128
Invalid -DS 16 RTN and -DS 32 RTN Combination 128
Selected Feedback Return Exception 129
Master and Slave Dependent Exceptions 130
Channel Timeout Exception 130
Exception Reporting 131
Exception Reporting using -CHCK 131

Synchronous -CHCK 132
Asynchronous -CHCK 137
Exception Reporting Using Interrupts 138
Exception Handling 138
Bus Master 138
System Master 138
DMA Controller 139
Slaves 139
Select Feedback Exception Handling 140
Bus Master 140
System Master 140
DMA Controller 140
Channel Reset 141

Connector Definition 142
Micro Channel Connector 144
Matched-Memory Extension 148
Auxiliary Video Extension 149
Base Video Extension 149
Adapter Physical Specifications 150
Adapter Size 151
Connector Spacing 151
Power 152
Voltage Regulation 153

Index 154

Figures

1. Master-Slave Relationships 4
2. Micro Channel Structure 5
3. I/O and Memory Address Space 6
4. Address Bus Support - Memory Cycle 12
5. Address Bus Support - I/O Cycle 12
6. Port Size of Selected Slave 16
7. Streaming Data Rates 17
8. I/O and Memory Transfer Controls 17
9. Driver and Receiver Requirements and Options 21
10. Signal Groups, Driver Types, and Loading Specifications . 22
11. The 32-Bit Connector and the Optional Extensions 25
12. The 16-Bit Connector and the Optional Extensions 26
13. Arbitration Block Diagram 29
14. Local Arbiter Example 31
15. Burst Transfer 32
16. Preempted Burst Transfer 33
17. Exiting from the Inactive State 35
18. Arbitration Cycle 36
19. Arbitration Cycle Timing Specifications 36
| 20. First Cycle After Grant 38
21. First-Cycle-After-Grant Timing Specifications 38
22. Basic Transfer Procedure - Overview 42
23. Default Cycle (200 ns Minimum) 44
24. Default Cycle (200 ns Minimum) Timing Specifications . . . 44
25. Synchronous-Extended Cycle (300 ns Minimum) 48
26. Synchronous-Extended Cycle Timing Specifications (300 ns Minimum) 48
27. Asynchronous-Extended Cycle Timing Specifications (≥300 ns) 50
28. Asynchronous Extended Cycle (≥ 300 ns) 51
29. Steering Control 53
30. Steering Responsibility (Basic Transfer, DMA, and Setup Procedures) 54
31. Steering Responsibility (Streaming Data Procedure) 54
32. Data Bus Steering Implementation 57
33. Streaming Data Transfer Rates 62
34. Data Block Example 64
35. Streaming Data Signal Combinations 65
36. 16- and 32-Bit Streaming Data Cycle Overview 66
| 37. 64-Bit Streaming Data Cycle Overview 67
| 38. Streaming Data Deferred Start Overview 70
| 39. CD CHRDY and Data Pacing 72

© Copyright IBM Corp. 1989, 1991

40. Master-Terminated Streaming Data Cycle 73
| 41. Master-Terminated Streaming Data Cycle (Slave Not Ready) 74
| 42. Slave-Terminated Streaming Data Cycle 75
43. Streaming Data Cycle 76
44. Streaming Data Cycle - Deferred Start 77
45. Streaming Data Transfer Timing Specifications 77
| 46. Streaming Data Clocking (16- and 32-Bit) 79
47. Streaming Data Transfer Timing Specifications (16- and 32-Bit) 79
| 48. Streaming Data Clocking (64-Bit) 81
49. Streaming Data Transfer Timing Specifications (64-Bit) . . . 82
50. Single DMA Transfer (DMA-Controller Controlled) 88
51. Single DMA Transfer (DMA-Controller Controlled) Timing Specifications 89
52. Burst DMA Transfer (DMA-Controller Terminated) 90
53. Burst DMA Transfer (DMA-Controller Terminated) Timing Specifications 90
54. Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns) 92
55. Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns) Timing Specifications 92
56. Burst DMA Transfer (DMA-Slave Terminated - Synchronous-Extended Cycle) 94
57. Burst DMA Transfer (DMA-Slave Terminated - Synchronous-Extended Cycle, 300 ns) Timing Specifications 95
58. Burst DMA Transfer (DMA-Slave Terminated - Asynchronous-Extended Cycle, ≥300 ns) 96
59. Burst DMA Transfer (DMA-Slave Terminated - Asynchronous-Extended Cycle, ≥300 ns) Timing Specifications 97
60. Typical Adapter Interrupt Sharing Implementation 98
61. POS I/O Address Space 103
62. Subaddressing Model 110
63. POS I/O Address Decode 112
64. Setup Cycle Overview 114
65. System Configuration Timing 115
66. System Configuration Timing Specifications 116
| 67. Data Parity Checking Read Cycle - Basic Transfer 121
68. Data Parity Checking Read Timing Specifications 121
69. Data Parity Read Cycle - Streaming Data Transfer 122
70. Data Parity Checking Write Cycle - Basic Transfer 123
71. Data Parity Checking Write Timing Specifications 123
72. Data Parity Write Cycle - Streaming Data Transfer 125
73. Data Parity Checking Write Timing Specifications 125
74. Address Parity 127
75. Address Parity Timing Specifications 127

76. Channel-Check - Default Cycle 133
77. Channel-Check Timing Specifications - Default Cycle . . . 133
78. Channel-Check - Extended Cycle 134
79. Channel-Check Timing Specifications - Extended Cycle . 134
80. Channel Check - Streaming Data Read Cycle 135
81. Channel-Check Timing Specifications - Streaming Data Read Cycle 135
82. Channel Check - Streaming Data Write Cycle 136
83. Channel-Check Timing Specifications - Streaming Data Write Cycle 136
84. Micro Channel Bus and Signal Status after a CHRESET . 141
85. Micro Channel Connectors 143
86. Micro Channel Connector 145
87. Micro Channel Connector - continued 146
88. Micro Channel Connector - continued 147
89. Channel Connector Voltage and Signal Assignments (Matched-Memory Extension) 148
90. Auxiliary Video Extension 149
91. Base Video Extension 150
92. Minimum current available for the Type 3 adapter. 152
93. Minimum current available for the Type 5 adapter 152
94. Channel Voltage Regulation 153

Notes:

Description

The Micro Channel* architecture consists of an address bus, a data bus, an arbitration bus, a set of interrupt signals, and support signals. It uses synchronous and asynchronous procedures for data transfer between memory, I/O devices, and a controlling master. The controlling master can be a DMA controller, the system master (system processor), or a bus master. The features of the Micro Channel architecture are:

- I/O data transfers of 8-, 16-, 24-, or 32-bits within a 64KB[1] address space (16-bit address width).
- Memory data transfers of 8-, 16-, 24-, or 32-bits within a 16MB[1] (24-bit address width) or 4GB[1] (32-bit address width) address space.
- An arbitration procedure that enables up to 15 devices and the system master to bid for control of the channel.
- A basic transfer procedure that allows data transfers between masters and slaves.
- A direct memory access (DMA) procedure that supports multiple DMA channels. Additionally, this procedure allows a device to transfer data in bursts.
- An optional streaming data procedure that provides a faster data-transfer rate than the basic transfer procedure and allows 64-bit data transfers.
- Address- and data-parity enable and detect procedures.
- Interrupt sharing on all levels.
- A flexible system-configuration procedure that uses programmable registers.

---

* Trademark of the IBM Corporation

1 KB equals 1024 bytes; MB equals 1 048 576 bytes; GB equals 1 073 741 824 bytes.

© Copyright IBM Corp. 1989, 1991

- An adapter interface to the channel using:
  - A 16-bit connector with a 24-bit address bus and a 16-bit data bus
  - A 32-bit connector with a 32-bit address bus and a 32-bit data bus
  - An optional matched-memory extension
  - An optional video extension.
- Support for audio signal transfer (audio voltage-sum node).
- Support for both synchronous and asynchronous data transfer.
- An exception condition reporting procedure.
- Improved electromagnetic characteristics.

Micro Channel Participants

All Micro Channel participants are either masters or slaves. There are three types of masters and three types of slaves. (See Figure 1 on page 4).

An adapter can incorporate either a master function, a slave function, or a combination of both. For example, an adapter might be designed to operate primarily as a DMA slave. However, it would also respond to certain I/O read and I/O write operations from the system master, making it an I/O slave. If the adapter contains RAM or ROM that is in the memory address space, it would be a memory slave when that memory was accessed.

Masters

A master is a participant that drives the address bus and data transfer control signals that cause data transfer to or from a slave.

The channel supports up to 16 masters. The central arbitration control point and the Micro Channel arbitration procedure control ownership of the channel. The central arbitration control point grants ownership at the end of the arbitration procedure to highest-priority requester.

The three types of masters are:

- System master

  A system master controls and manages the system configuration. It arbitrates for use of the channel. A system master can also be a default master. The default master is the master that owns the channel when no other master requires the channel.

  The system master supports data transfers with an I/O slave or a memory slave.

- Bus master

  A bus master arbitrates for use of the channel. A bus master supports data transfers with an I/O slave or a memory slave.

- DMA controller

  A DMA controller does not initiate arbitration for the channel; a DMA slave initiates and completes the arbitration procedure. The DMA controller monitors the arbitration bus to detect the DMA slave's arbitration level.

  A DMA controller supports data transfers with DMA slaves and memory slaves.

Slaves

A slave is a participant that sends and receives data under the control of a master. The slave responds to signals that are driven by the master. A slave is selected by the controlling master using Micro Channel procedures.

The channel supports 8-, 16- and 32-bit data bus size in a slave. The data port size (8-, 16-, or 32-bits) describes the maximum width of the data transfer. For example, a 32-bit slave can support 8-, 16- 24-, or 32-bit data transfers. During a data transfer cycle, an 8-bit data port on a 32-bit slave acts like an 8-bit slave.

The three types of slaves are:

- I/O slave

  I/O slaves are selected by their address within the I/O address space.

- Memory slave

  Memory slaves are selected by their address within the memory address space.

- DMA slave

  A DMA slave is selected by arbitration or, optionally, by its address within the I/O address space. A DMA slave initiates a request for the channel and arbitrates for the channel with arbitrating masters and other DMA slaves. The DMA controller provides the DMA slave with the address and data transfer signals required for data transfers. During a DMA write operation, the DMA slave provides the data, and the memory slave stores it. During a DMA read operation, the memory slave provides the data, and the DMA slave stores it.

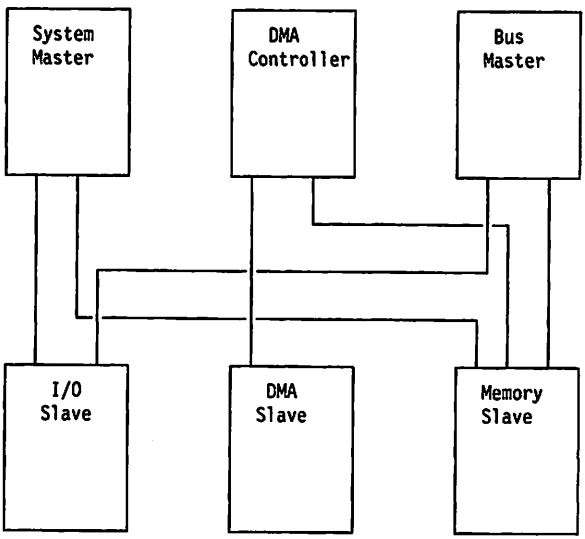
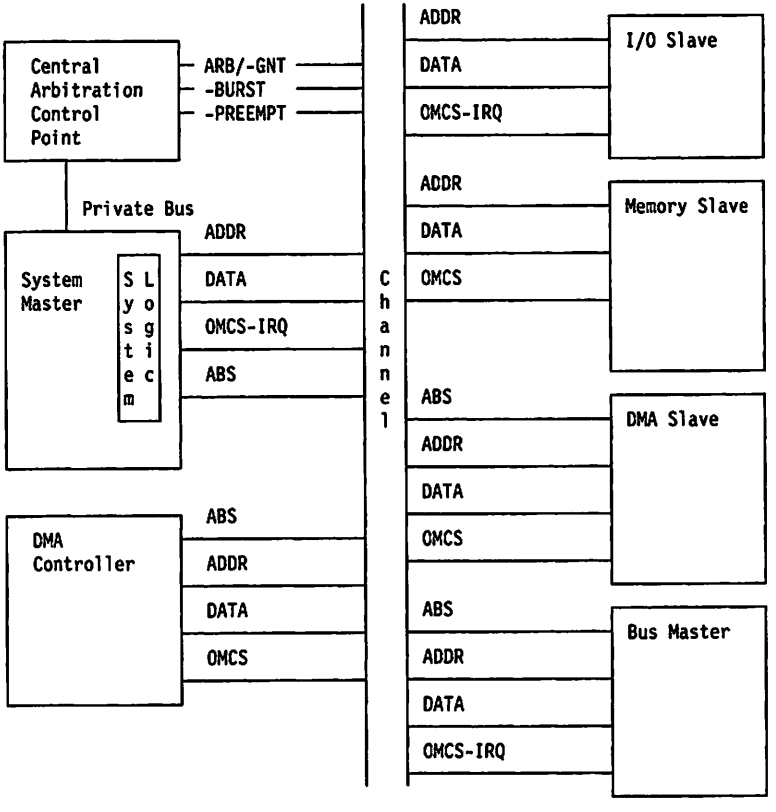


Figure 1. Master-Slave Relationships

Micro Channel Structure

The following shows the participants, the central arbitration control point, and the Micro Channel buses and signals that they drive or receive. "Micro Channel Buses and Signals" on page 7 describes the buses and signals of the Micro Channel architecture.



ABS - Arbitration Bus and Associated Signals
ADDR - Address Bus and Associated Signals
DATA - Data Bus and Associated Signals
IRQ - Interrupt Request Signals
OMCS - Other Micro Channel Signals

Figure 2. Micro Channel Structure

Figure 9 on page 21 shows which Micro Channel participant drives and receives each bus and signal, and specifies the requirements for each driver and receiver.

Addressing Model

The Micro Channel addressing model consists of a memory address space and an I/O address space. During an I/O cycle, the 64KB I/O address space is addressed by the low-order 16 bits of the address bus. During a memory cycle, the memory address space is addressed by the address bus.

The signals used for an address of up to 4GB are MADE 24, M/-IO, and A0 through A31 of the address bus. An address of up to 16MB uses A0 through A23 plus MADE 24 and M/-IO.

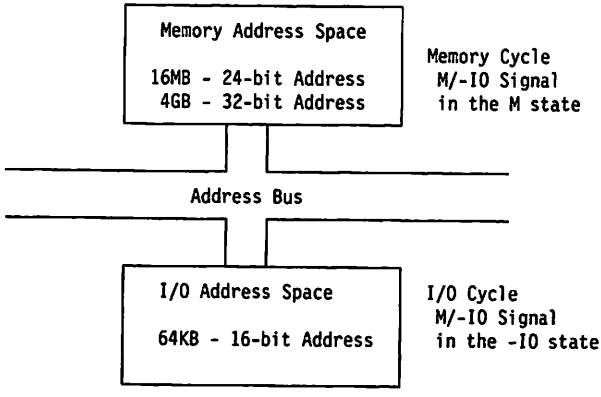


Figure 3. I/O and Memory Address Space

Micro Channel Buses and Signals

The Micro Channel architecture consists of:

- An arbitration bus and associated signals
- An address bus and associated signals
- A data bus and associated signals
- Interrupt signals
- Other Micro Channel signals
- Optional extensions for:
  - Matched memory extension signals
  - Video extension signals.

Throughout this document, a minus sign (-) in front of a signal name indicates that the signal is active when it is at a low-voltage level. When no minus sign appears, the signal is active when it is at a high-voltage level. For example, -CMD specifies the signal is active low. Also, ARB/-GNT is in the ARB state when it is at a high level and is in the -GNT state when it is at a low level.

Arbitration Bus and Associated Signals

The arbitration bus and associated signals allow arbitrating participants (the system master, bus masters, and DMA slaves) to request and gain ownership of the channel. The resolution of multiple arbitration requests results in granting ownership of the channel to the highest-priority requester. See "Arbitration" on page 27 for more information.

ARB0 – ARB3: Arbitration Bus: These signals make up the arbitration bus. They are driven by the system master, bus masters, and DMA slaves to present their arbitration level when requesting ownership of the channel. ARB0 (least significant) through ARB3 (most-significant) support up to 16 arbitration levels.

The highest value of the arbitration bus (hex F) has the lowest priority, and the lowest value (hex 0) has the highest priority. To participate in the arbitration procedure, an arbitrating participant must present its arbitration level immediately after the rising edge of ARB/-GNT. All arbitrating participants monitor the arbitration bus, and

those with lower priority arbitration levels withdraw them by not driving less-significant arbitration bits.

The arbitration level of the highest-priority requester is valid on the arbitration bus after a settling time. After the channel is granted to the highest-priority requester, that requester continues to drive its arbitration level on the bus.

ARB/-GNT: Arbitrate/-Grant: Only the central arbitration control point drives this signal. The negative-to-positive transition of ARB/-GNT initiates an arbitration cycle. When in the ARB state, this signal indicates an arbitration cycle is in progress. When in the -GNT state, this signal indicates the acknowledgment from the central arbitration control point to the arbitrating participants and the DMA controller that channel ownership has been granted. This signal is driven to the ARB state by the central arbitration control point following the end of transfer (EOT). (See page 28 for a description of the end of transfer.)

Note: The system master can perform data transfers during arbitration (ARB/-GNT in the ARB state).

-BURST: -Burst: This signal is optionally driven by the winning arbitrating participant or the DMA controller after ARB/-GNT is driven to the -GNT state. This signal indicates to the central arbitration control point that the controlling master will use the channel for one or more consecutive data transfer cycles. This type of data transfer is called *burst transfer.*

-PREEMPT: -Preempt: This signal is used by arbitrating participants to request use of the channel through arbitration. Any arbitrating participant that requires ownership of the channel drives -PREEMPT active, causing an arbitration cycle to occur. When a participant is granted control of the channel, it stops driving -PREEMPT. All arbitrating participants that have not been granted ownership keep their requests pending by continuing to drive -PREEMPT active. All masters and DMA slaves that use burst transfer must receive -PREEMPT.

Address Bus and Associated Signals

The address bus and the associated signals are used by the controlling master to assert the memory address or the I/O address (M/-IO), to enable a slave to latch the address and status signals (-S0, -S1), and to indicate that the memory address is greater than 16MB.

A0 – A23: Address Bits 0 through 23: These lines, along with A24 through A31, make up the address bus. These lines are driven by the controlling master to address memory, I/O slaves, and, optionally, DMA slaves. A0 is the least-significant bit and A23 is the most-significant bit. These 24 address lines allow access of up to 16MB of memory. Only the lower 16 address lines (A0 through A15) are for I/O operations, and all 16 lines must be decoded by the I/O slave. A0 through A23 are generated by the controlling master. Valid addresses, generated by the controlling master, are unlatched on the channel and, if required, must be latched by the slaves using either the leading or trailing edge of -ADL or the leading edge of -CMD.

A24 – A31: Address Bits 24 through 31: These lines, along with A0 through A23 are driven by the controlling master to address memory attached to the channel. A0 is the least-significant bit and A31 is the most-significant bit. These additional address lines allow access of up to 4GB of memory. A24 through A31 are generated by the controlling master. Valid addresses, generated by the controlling master, are unlatched on the channel and, if required, must be latched by the slaves using either the leading or trailing edge of -ADL or the leading edge of -CMD.

Note: A0 - A31 are used to transfer data during a 64-bit streaming data cycle.

-ADL: -Address Decode Latch: This signal, driven by the controlling master, is provided as a convenient way for the slave to latch valid address decodes and status bits.

-APAREN: -Address Parity Enable: This optional signal is driven active by a master when the master places an address on the bus. This signal indicates to a slave that the address parity signals are valid.

APAR0 – APAR3: Address Parity Bits 0 through 3: These optional signals are driven by a master when an address is placed on the address bus. These signals represent the odd parity of the address bits on the address bus during both read and write operations. (Odd parity is the condition where the total number of 1s in a byte of data, including the parity bit, is odd.)

APAR(0) represents the odd parity of A(0-7).
APAR(1) represents the odd parity of A(8-15).
APAR(2) represents the odd parity of A(16-23).
APAR(3) represents the odd parity of A(24-31).

During both read and write operations, a master generates a parity bit for each valid address byte, and the receiving slave optionally performs the parity checking to ensure the integrity of the address.

Note: APAR0 - APAR3 represent data parity during 64-bit streaming data cycle when -DPAREN is active.

APAR(0) represents the odd parity of D(32-39).
APAR(1) represents the odd parity of D(40-47).
APAR(2) represents the odd parity of D(48-55).
APAR(3) represents the odd parity of D(56-63).

-CD SFDBK (n): -Card Selected Feedback: This signal is driven active by the selected slave as a positive acknowledgement of the slave's selection. The (n) indicates this signal is unique to each channel connector (one independent signal per connector). This signal is unlatched and driven by the selected slave. The slave does not drive -CD SFDBK during the configuration procedure (-CD SETUP active).

Note: Memory that contains diagnostic code must not drive -CD SFDBK during the diagnostic operation.

MADE 24: Memory Address Enable 24: This signal is driven by the controlling master and decoded by all memory slaves, regardless of the size of their address-space. When this signal is active, A24 - A31 are undefined.

- A master driving only A0 - A23 drives MADE 24 active.
- A master driving A0 - A31 drives MADE 24:
  - Active when all bits in A24 - A31 are 0
  - Inactive when any bit in A24 - A31 is 1.

M/-IO: Memory/-Input Output: This signal is driven by the controlling master and decoded by all slaves. This signal selects a memory cycle or an I/O cycle. When this signal is in the M state, a memory cycle is selected. When this signal is in the -IO state, an I/O cycle is selected.

-SFDBKRTN: Selected Feedback Return: This optional signal is generated by the system logic from the AND of the -CD SFDBK(n) signals being driven by slaves. This signal is a positive acknowledgement to the master from the slave that the slave is at the address specified by the master. Masters that support address parity must receive this signal.

Address Signal Combinations

The following tables show the valid addressing signal combinations for:

- the address bus (A0 - A23 and A24 - A31)
- M/-IO
- MADE 24
- -APAR(0-3).

Addressing Signals	16-Bit Connector	32-Bit Connector	
	24-Bit Address	< 16MB Address	≥ 16MB Address
M/-IO	1	1	1
MADE 24	1	1	0
A0 - A23	Valid	Valid	Valid
APAR0 - APAR2 (Optional)	N/A	Valid	Valid
A24 - A31	N/A	X	Valid
APAR3 (Optional)	N/A	X	Valid
X - Don't Care	N/A - Not applicable		

Figure 4. Address Bus Support - Memory Cycle

Addressing Signals	16-Bit Connector	32-Bit Connector
M/-IO	0	0
MADE 24	X	X
A0 - A15	Valid	Valid
APAR0 - APAR1 (Optional)	N/A	Valid
X - Don't Care	N/A - Not applicable	

Figure 5. Address Bus Support - I/O Cycle

Data Bus and Associated Signals

The data bus is used to transfer either 8, 16, 24, or 32 bits of data. The associated signals indicate the amount of data transferred by the master in a single transfer cycle, the size of the slave's data port, and the type (read or write) of the data transfer.

D0 – D15: Data Bits 0 through 15: These lines, along with D16 - D31, make up the data bus. The data bus is driven by any master or slave that is transferring data. These lines (D0 - D15) provide data bits 0 through 15. D0 is the least-significant bit; D15 is the most-significant bit. The 16-bit transfers from the controlling master to an 8-bit slave are converted by the controlling master to two 8-bit transfers, and are transmitted on lines D0 through D7. An 8-bit slave must use D0 through D7 to communicate with the controlling master.

D16 – D31: Data Bits 16 through 31: These lines, along with D0 - D15, make up the data bus. The data bus is driven by any master or slave that is transferring data. These lines (D16 - D31) provide data bits 16 through 31. D0 is the least-significant bit; D31 is the most-significant bit. The 32-bit transfers from the controlling master to an 8-bit slave are converted to four 8-bit transfers by the controlling master, and are transmitted on lines D0 through D7. The 32-bit transfers from the controlling master to a 16-bit slave are converted to two 16-bit transfers by the controlling master, and are transmitted on lines D0 through D15.

-BE0 – -BE3: -Byte Enable 0 through 3: These signals are used during data transfers with 32-bit slaves to indicate which data bytes are valid on the data bus. Data transfers of 8, 16, 24, or 32 contiguous bits are controlled by -BE0 through -BE3 during transfers involving 32-bit slaves only. These signals are driven by the controlling master when TR 32 is inactive, and by the central translator logic (for those operations involving a 16-bit master with a 32-bit slave) when TR 32 is active. These signals are not latched on the bus and, if required, are latched by 32-bit slaves. (See "Channel Support" on page 55 for additional information.)

-CD DS 16 (n): -Card Data Size 16: This signal is driven by 16-bit and 32-bit slaves to indicate a 16-bit or 32-bit data port at the location addressed. The (n) indicates this signal is unique to each channel connector (one independent signal per connector). This signal is derived from a valid unlatched address decode. -CD DS 16 is not driven by 8-bit slaves and is inactive for an 8-bit data port.

-CD DS 32 (n): -Card Data Size 32: This signal, along with -CD DS 16, is driven by 32-bit slaves to indicate a 32-bit data port at the location addressed. The (n) indicates this signal is unique to a channel connector position (one independent signal per connector). -CD DS 32 is derived from a valid unlatched address decode. -CD DS 32 is inactive for an 8- or 16-bit data port.

CD CHRDY (n): Channel Ready: This signal is normally active (ready) and is driven inactive (not ready) by a slave to allow additional time to complete a channel cycle. The (n) indicates this signal is unique to each channel connector (one independent signal per connector).

During a read cycle, a slave ensures that data is valid within the time specified after releasing the signal to a ready state. The slave also holds the data long enough for the controlling master to sample the data. A slave can also use this signal during a write cycle if more time is needed to store the data. This signal is initially driven from a valid unlatched address decode and status active.

CHRDYRTN: Channel Ready Return: This signal is the AND of CD CHRDY (n). It is driven by the system logic. If all slaves drive CD CHRDY active, this signal is active. CHRDYRTN allows the controlling master to monitor the ready information.

-CMD: -Command: This signal is driven by the controlling master and is used to define when data on the data bus is valid. The trailing edge of this signal indicates the end of the bus cycle. This signal indicates to the slave how long data is valid. During write cycles, the data is valid as long as -CMD is active. During read cycles, the data is valid after the leading edge, but before the trailing edge, of -CMD and is held on the bus until after -CMD goes inactive. Slaves can latch address and status information with the leading edge of -CMD.

-DPAREN: -Data Parity Enable: This optional signal is driven active by the participant when data is placed on the data bus. This signal indicates that the data parity signals are valid.

Note: APAR(0) - APAR(3) represent data parity during 64-bit streaming data cycles when -DPAREN is active.

DPAR0 – DPAR1: Data Parity Bits 0 and 1: These optional signals are driven by the participant when data is placed on the data bus. These signals represent the odd parity of the data bits on the data bus during both read and write operations. (Odd parity is the condition where the total number of 1s in a byte of data, including the parity bit, is odd.)

DPAR(0) represents the odd parity of D(0-7).
DPAR(1) represents the odd parity of D(8-15).

During write operations, a master generates a parity bit for each data byte being transferred, and the receiving slave optionally performs the parity checking to ensure the integrity of the data. During read operations, a slave generates a parity bit for each valid data byte, and the receiving master performs the parity checking to ensure the integrity of the data.

DPAR2 – DPAR3: Data Parity Bits 2 and 3: These optional signals are driven by the participant when data is placed on the data bus. These signals represent the odd parity of the data bits on the data bus during both read and write operations.

DPAR(2) represents the odd parity of D(16-23).
DPAR(3) represents the odd parity of D(24-31).

During write operations, a master generates a parity bit for each data byte being transferred, and the receiving slave optionally performs the parity checking to ensure the integrity of the data. During read operations, a slave generates a parity bit for each valid data byte, and the receiving master performs the parity checking to ensure the integrity of the data.

-DS 16 RTN: -Data Size 16 Return: This signal is driven by the system logic. This signal is the AND of -CD DS 16 (n) from each channel connector. If any slave drives its -CD DS 16 active, this signal is active. This signal allows the controlling master to monitor the information about the selected slave's data port size (see Figure 6 on page 16).

-DS 32 RTN: -Data Size 32 Return: This signal is driven by the system logic. This signal is the AND of -CD DS 32 (n) from each channel connector. If any slave drives its -CD DS 32 active, this signal is active. This signal allows the controlling master to monitor the information about the selected slave's data port size.

-DS 16 RTN	-DS 32 RTN	Data Port Size
1	1	8-Bit Data Port
1	0	Not Valid
0	1	16-Bit Data Port
0	0	32-Bit Data Port

Figure 6. Port Size of Selected Slave

-MSDR: -Multiplexed Streaming Data Request: This signal is driven by a slave to indicate to the controlling master that the slave is capable of 64-bit streaming data transfers.

-SBHE: -System Byte High Enable: This signal is driven by the controlling master to indicate and enable transfers of data on D8 - D15. It is used with A0 to distinguish between high-byte transfers (D8 – D15) and low-byte transfers (D0 – D7) and double-byte (16-bit) transfers to 16-bit data ports. All 16-bit slaves receive this signal.

-SD STROBE: -Streaming Data Strobe: The controlling master and the slave use this signal to clock data on and off the data bus. This optional signal is driven by the controlling master. This signal also indicates to the slave that the controlling master supports streaming data.

-SDR(0): -Streaming Data Request 0: This optional signal is driven by a slave to indicate to the controlling master that the slave is capable of streaming data, and also indicates the maximum clocking rate the slave supports.

-SDR(1): -Streaming Data Request 1: This optional signal is driven by a slave to indicate to the controlling master that the slave is capable of streaming data, and also indicates the maximum clocking rate the slave supports.

-SDR(0)	-SDR(1)	Decoded Streaming Rate
1	1	Basic Transfer Cycle
0	1	10 MHz maximum (100 ns minimum cycle)
1	0	Reserved
0	0	Reserved

Figure 7. Streaming Data Rates

-S0, -S1: -Status 0, -Status 1: These status signals are driven by the controlling master to indicate the start of a data transfer cycle and also define the type of data transfer. When used with M/-IO, memory read or write cycles are distinguished from I/O read or write cycles. These signals are latched by the slave, as required, using the leading edge of -CMD, or the leading or trailing edge of -ADL.

Data is transferred to or from the data bus based on -CMD and a latched decode of the address, the status lines (-S0 exclusive-OR -S1), and M/-IO.

Slaves must support a full decode of -S0 and -S1. The following table shows the states of M/-IO, -S0, and -S1 in decoding I/O and memory read and write commands.

M/-IO	-S0	-S1	Function
0	0	0	Reserved
0	0	1	I/O Write Command
0	1	0	I/O Read Command
0	1	1	Inactive
1	0	0	Reserved
1	0	1	Memory Write Command
1	1	0	Memory Read Command
1	1	1	Inactive

Figure 8. I/O and Memory Transfer Controls

An I/O write command instructs an I/O slave to receive the data from the data bus. An I/O read command instructs an I/O slave to drive its data onto the data bus.

A memory write command instructs a memory slave to receive the data from the data bus. A memory read command instructs a memory slave to drive its data onto the data bus.

-TC: -Terminal Count: This signal is driven by the DMA controller and provides a pulse during a read or write command to the DMA slave to indicate that the terminal count of the current DMA channel has been reached. This indicates to the DMA slave that this is the last cycle to be performed. -TC is driven active on the channel during DMA operations only.

TR 32: Translate 32: This signal is driven inactive by 32-bit controlling masters and received by the central translator logic. The signal indicates to the central translator logic that the controlling master is performing data steering. TR 32 can also be received by any 32-bit slave. See "Data Steering" on page 53 for information about the central translator logic.

Interrupt Signals

-IRQ 3 – 7, -IRQ 9 – 12, and -IRQ 14 – 15: -Interrupt Request: An interrupt request is generated when an I/O slave drives one of the 'interrupt request' signals low. These signals make up the set of interrupt signals. The polarity of these signals makes it possible for multiple slaves to concurrently share the same interrupt level.

Other Micro Channel Signals

AUDIO: Audio Sum Node: This analog signal is the sum of all the audio signals being driven. It is used to drive audio signals from an adapter to the system audio output and between adapters. The frequency response of the audio signal is 50 Hz to 10 kHz ± 3 dB. The maximum signal amplitude is 2.5 Vac peak-to-peak, at a dc offset of 0.0 Vdc ± 50 millivolts. The noise level is limited to a maximum of 50 millivolts peak-to-peak.

AUDIO GND: Audio Ground: This is the analog ground return signal for the audio subsystem.

-CD SETUP (n): -Card Setup: This signal is driven by system logic to individually select channel connectors. The (n) indicates this signal is unique to each channel connector (one independent signal per connector). When this signal is activated, a specific channel connector is selected, and, if an adapter is present, access is gained to configuration data at that connector. The adapter ID and configuration data is obtained by an I/O read during a setup cycle; the configuration data is stored by an I/O write during a setup cycle.

-CHCK: -Channel Check: This signal is driven active by a slave to report an exception condition, and optionally, it can be driven by a master.

CHRESET: Channel Reset: This signal is generated by the system logic to reset or initialize all adapters at power-on or when a low voltage condition is detected. The system can also activate this signal under program control.

OSC: Oscillator: This signal is a high-speed clock driven by the system logic, with a frequency of 14.31818 MHz ± 0.01%. The high-level pulse width (more than 2.3 Vdc) and the low-level pulse width (less than 0.8 Vdc) must not be less than 20 nanoseconds each.

-REFRESH: -Refresh: This signal is driven by the system logic and is used to indicate that a memory refresh operation is in progress. Memory slaves that do not need to perform refresh operations do not need to receive this signal.

Channel Signal Groups (Digital)

The following figure lists the digital Micro Channel signals and shows what type of driver or receiver is required. The 'audio' and 'audio ground' signals are analog signals. (For further information about these signals, see page 19.)

When a Micro Channel adapter combines the functions of bus masters and slaves, the signals driven and received by the adapter are a superset of the selected functions.

For example:

- An adapter that combines a slave function with a bus master function must drive and receive all signals required by a slave and all signals required by a bus master.
- An adapter that combines an I/O slave function with a memory slave or DMA slave function, must drive and receive all signals required by both slaves.

  Note: All adapters must support setup.

The central arbitration control point, central steering logic, and central translator logic are packaged within the system logic. The system logic also includes circuitry for memory refresh, the oscillator, and channel reset.

The signals for the 16-bit adapter are a subset of the signals for the 32-bit adapter.

Signal Name	System Master	DMA Cntlr	Bus Master	DMA Slave	Memory Slave	I/O Slave	Driver Type
	D/R	D/R	D/R	D/R	D/R	D/R	Signal Group
A(0-15)	D/-	D/-	D/-	-/0	-/R	-/R	TS (1)
A(16-23)	D/-	D/-	D/-	-/-	-/R	-/-	TS (1)
A(24-31)	*/-	*/-	*/-	-/-	-/*	-/-	TS (1)
-ADL	D/-	D/-	D/-	-/0	-/0	-/0	TS (1)
APAR(0-3)	0/-	0/-	0/-	-/0	-/0	-/0	TS (2)
-APAREN	0/-	0/-	0/-	-/0	-/0	-/0	TS (2)
ARB(0-3)	0/0	-/R	D/R	D/R	-/-	-/-	OC (5)
ARB/-GNT	D/-	-/R	-/R	-/R	-/-	-/-	BD (4)
-BE(0-3)	*/-	*/-	*/-	-/*	-/*	-/*	TS (1)
-BURST	0/R	D/R	0/-	0/-	-/0	-/0	OC (5)
CD CHRDY (n)	-/R	-/-	-/-	0/-	0/-	0/-	TP (3)
-CD DS 16 (n)	-/R	-/-	-/-	$/-	$/-	$/-	TP (3)
-CD DS 32 (n)	-/*	-/-	-/-	*/-	*/-	*/-	TP (3)
-CD SETUP (n)	D/-	-/@	-/@	-/@	-/@	-/@	TP (8)
-CD SFDBK (n)	-/R	-/-	-/-	D/-	D/-	D/-	TP (3)
-CHCK	-/R	0/R	0/R	0/-	0/-	0/-	OC (6)
CHRDYRTN	D/-	-/R	-/R	-/-	-/-	-/-	BD (4)
CHRESET	D/-	-/R	-/R	-/R	-/R	-/R	BD (4)
-CMD	D/-	D/-	D/-	-/R	-/R	-/R	TS (1)
D(0-7)	D/R	D/R	D/R	D/R	D/R	D/R	TS (2)
D(8-15)	D/R	D/R	D/R	$/$	$/$	$/$	TS (2)
D(16-31)	*/*	*/*	*/*	*/*	*/*	*/*	TS (2)
DPAR(0)	0/0	0/0	0/0	0/0	0/0	0/0	TS (2)
DPAR(1)	0/0	0/0	0/0	0/0	0/0	0/0	TS (2)
DPAR(2-3)	0/0	0/0	0/0	0/0	0/0	0/0	TS (2)
-DPAREN	0/0	0/0	0/0	0/0	0/0	0/0	TS (2)
-DS 16 RTN	D/-	-/R	-/R	-/-	-/-	-/-	BD (4)
-DS 32 RTN	*/-	-/*	-/*	-/-	-/-	-/-	BD (4)
-IRQ (N)	0/R	0/-	0/0	0/-	0/-	0/-	OC (6)
M/-IO	D/-	D/-	D/-	-/R	-/R	-/R	TS (1)
MADE 24	D/-	D/-	D/-	-/-	-/R	-/-	TS (1)
-MSDR	-/0	-/0	-/0	0/-	0/-	0/-	TS (1)
OSC	D/-	-/0	-/0	-/0	-/0	-/0	CD (7)
-PREEMPT	D/R	-/-	D/0	D/0	-/-	-/-	OC (5)
-REFRESH	D/-	-/-	-/-	-/-	-/0	-/-	BD (4)
-S0,-S1	D/*	D/-	D/-	-/R	-/R	-/R	TS (1)
-SBHE	D/*	D/-	D/-	-/#	-/#	-/#	TS (1)
-SFDBKRTN	0/-	-/0	-/0	-/-	-/-	-/-	BD (4)
-SD STROBE	0/-	0/-	0/-	-/0	-/0	-/0	TS (1)
-SDR(0)	-/0	-/0	-/0	0/-	0/-	0/-	TS (1)
-SDR(1)	-/0	-/0	-/0	0/-	0/-	0/-	TS (1)
-TC	-/-	D/-	-/-	-/R	-/-	-/-	TS (1)
TR 32	*/R	*/-	*/-	-/0	-/0	-/0	TS (1)

- KEY -

D = Drive Enabled
O = Optional
R = Receive Enabled
- = Not Implemented
CD = Clock Driver

OC = Open Collector
TS = Tri-State
TP = Totem Pole
BD = Bus Driver
N = IRQ(9-12,14,15,3-7)

= Required for 16-bit devices
$ = Required for 16- or 32-bit devices
* = Required for 32-bit devices
@ = -CD SETUP is received by all adapters

Figure 9. Driver and Receiver Requirements and Options

The following table and notes apply to the driver and receiver options listed on page 21.

Signal Group	Driver Type	Sinking Capacity	Capacitive Loading	Total Capacitance
1	Tri-State	24 mA	20 pF	240 pF
2	Tri-State	24 mA	20 pF	240 pF
3	Totem Pole	6 mA	20 pF	40 pF
4	Bus Driver	24 mA	20 pF	240 pF
5	Open Collector	24 mA	15 pF	200 pF
6	Open Collector	24 mA	20 pF	240 pF
7	Clock Driver	24 mA	15 pF	200 pF
8	Totem Pole or Tri-State	6 mA	50 pF	Not Applicable

Figure 10. Signal Groups, Driver Types, and Loading Specifications

Notes:

1. Pull-up resistors are provided by system logic for all signals other than -SDR0, -SDR1, -MSDR, and -SD STROBE. Adapters that receive these signals must provide a pull-up resistor for each signal received. (A minimum of 20 kΩ pulled to 5 Vdc is recommended.)
2. The maximum loading current is 1.6 mA per channel connector, except signal group 5. The maximum loading current of group 5 is 1.0 mA per channel connector.
3. An open collector can be either an open-collector device or a tri-state device wired with the input grounded and using the enable line of the tri-state device to control the output.
4. For good signal quality, the net length on the adapter for any control signal driven or received on the adapter should not exceed 2 inches.
5. A Tri-state driver can be used on -REFRESH as long as all timing requirements are met for this signal.

Matched-Memory Extension Signal Descriptions

The matched-memory extension can, in some cases, improve the data transfer capabilities between the system master and channel-resident memory. The matched-memory signal-sequence and signal-timing specifications are system dependent. For more information, see the technical manuals for the systems that support the matched-memory signals.

Auxiliary Video Extension Signal Descriptions

The auxiliary video extension allows optional video adapters to share video resources with the base video subsystem. The base video can be on the system board or on an adapter with the base video extension. When the base video subsystem is on the system board, support of the auxiliary video extension is direct. When the base video subsystem is on an adapter, support of the auxiliary video extension is through the base video extension.

The signals and signal timings for the auxiliary and base video extensions are system dependent. For information on these signals, see the technical references for your specific system.

The auxiliary video extension is shown in Figure 90 on page 149. The base video extension is shown in Figure 91 on page 150.

Micro Channel Architecture Base and Extensions

Figure 11 on page 25 shows the buses and signals available for use with the 32-bit connector. The optional extensions signals are shown at the top of the figure and the optional connector pins at the bottom of the figure.

Figure 12 on page 26 shows the buses and signals available for use with the 16-bit connector. The optional extensions signals are shown at the top of the figure and the optional connector pins at the bottom of the figure. The 16-bit connector is a subset of the 32-bit connector.

"Connector Definition" on page 142 contains the physical specifications for the connectors.

32-Bit Connector and Extensions

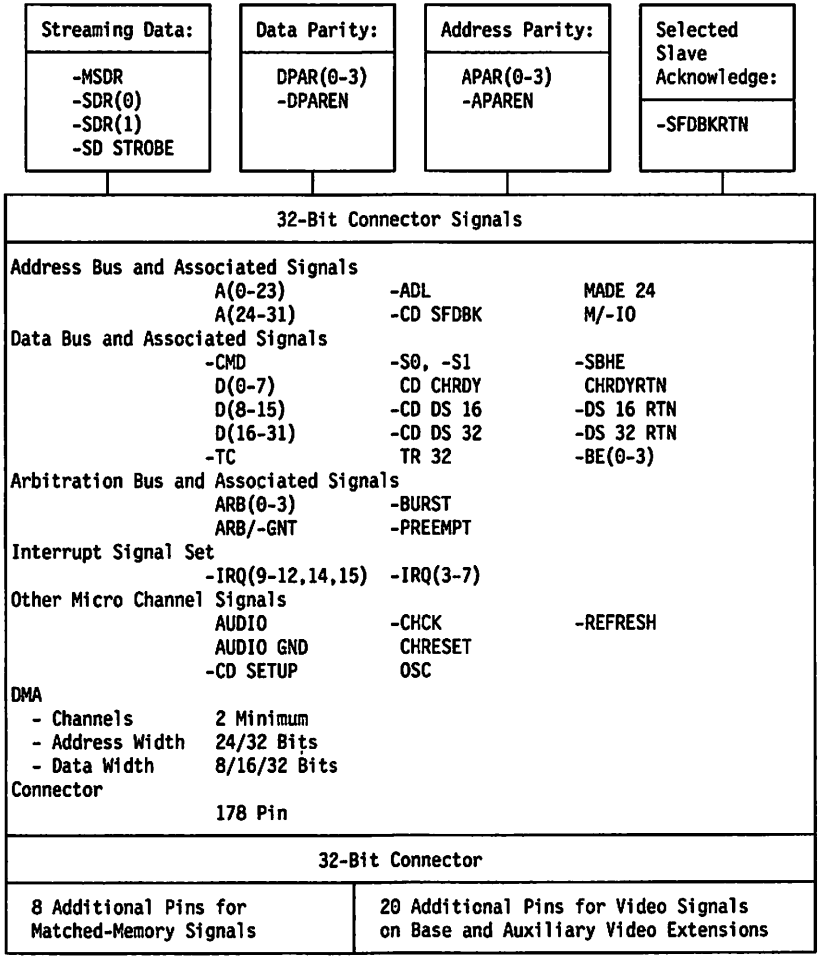


Figure 11. The 32-Bit Connector and the Optional Extensions

16-Bit Connector and Extensions

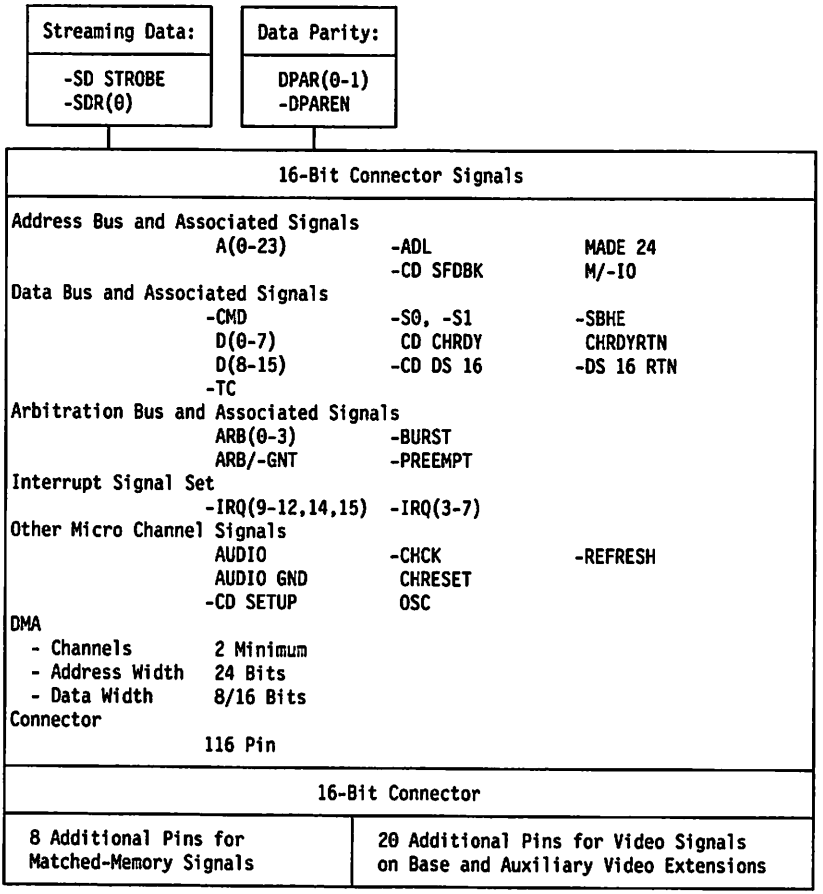


Figure 12. The 16-Bit Connector and the Optional Extensions

Micro Channel Procedures

The Micro Channel procedures, as well as data steering, are described in the following pages.

Arbitration

The arbitration procedure provides the functions to resolve multiple requests for control of the channel. Arbitration consists of local arbiters, the arbitration bus and the associated signals, and a central arbitration control point.

The arbitration bus is a parallel bus that consists of four lines ARB(0-3), supporting up to 16 arbitration levels. These arbitration levels are hex 0 through hex F; level hex 0 is the highest priority, level hex F is the lowest priority. The master with level hex F is always the default master. No two active arbiters will be assigned the same arbitration level at the same time.

If no -PREEMPT is driven, the arbitration level defaults to hex F and the default master gains control of the channel.

Channel Ownership

Channel ownership is controlled by the central arbitration control point based on prioritized arbitration of up to 16 devices. These arbitrating devices can be the system master, bus masters, or DMA slaves. If either a bus master or the system master wins the arbitration, it controls the bus and becomes the controlling master. If a DMA slave wins the arbitration, the supporting DMA controller becomes the controlling master.

If a system master is also the default master, the system master controls the Micro Channel when the channel is not controlled by any other master.

Central Arbitration Control Point

The central arbitration control point controls arbitration cycles using three arbitration signals: -PREEMPT, ARB/-GNT, and -BURST.

Arbitrating participants (local arbiters) drive -PREEMPT active to request use of the channel. An arbiter can activate -PREEMPT asynchronously. However, an arbiter must have -PREEMPT activated *before* the beginning of an arbitration cycle to participate in that cycle. If an arbiter drives -PREEMPT active when an arbitration cycle is in progress, it must wait for the *next* arbitration cycle before driving its arbitration level onto the arbitration bus.

The central arbitration control point initiates an arbitration cycle by driving ARB/-GNT to the ARB state. The requesting local arbiters then drive their 4-bit arbitration level onto the arbitration bus. When a local arbiter sees a more-significant bit low on the arbitration bus (other than those driven low by itself), it stops driving its less-significant bits onto the arbitration bus. The local arbiter driving the lowest arbitration level (highest priority) thereby wins control of the channel when ARB/-GNT goes to the -GNT state. When a master is granted control of the channel, it is required to initiate a transfer cycle or an abort cycle (see "Data Transfer" on page 40).

Arbitrating participants with multiple data transfers to perform signal the central arbitration control point by driving -BURST active. -BURST is held active until all transfers have been completed or until another participant drives -PREEMPT active. When preempted, the controlling master or DMA slave must release the channel and signal the end of transfer within 7.8 microseconds.

The end of transfer (EOT) is indicated by the trailing edge of -S0, -S1, -BURST, or -CMD, whichever occurs last. When the central arbitration control point detects an EOT condition, it drives ARB/-GNT to the ARB state, initiating the next arbitration cycle.

When ARB/-GNT is in the ARB state, bus masters and DMA controllers disable all of their channel drivers except -PREEMPT and ARB(0-3). Slaves, including bus masters and DMA controllers responding as slaves, must be capable of driving the channel when addressed.

During exception conditions, the central arbitration control point can gain control of the channel by driving ARB/-GNT to the ARB state independently of an EOT condition.

Note: While ARB/-GNT is in the ARB state, the system master can perform data transfers and system logic can perform refresh

operations. Selected slaves must respond independently of the state of ARB/-GNT

The following is a block diagram of the interaction between the central arbitration control point and the local arbiters.

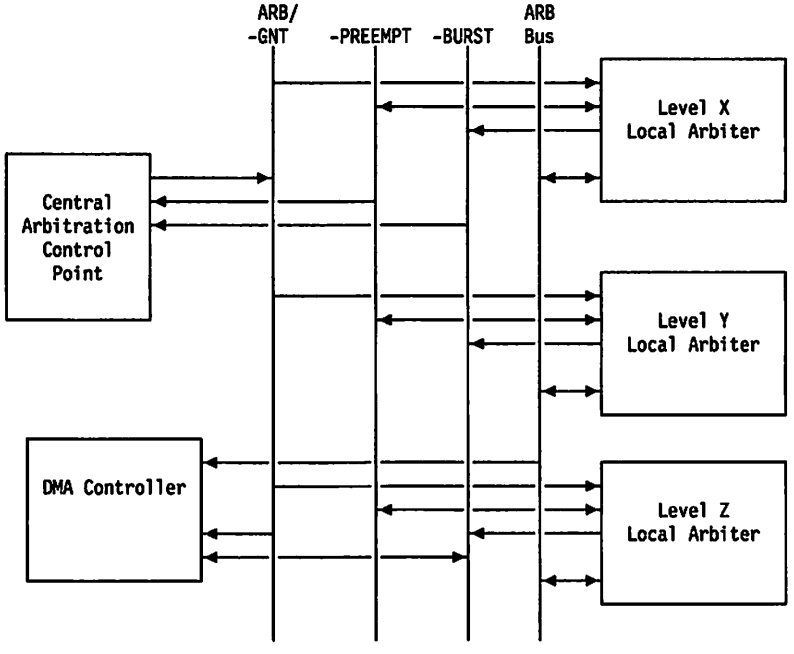


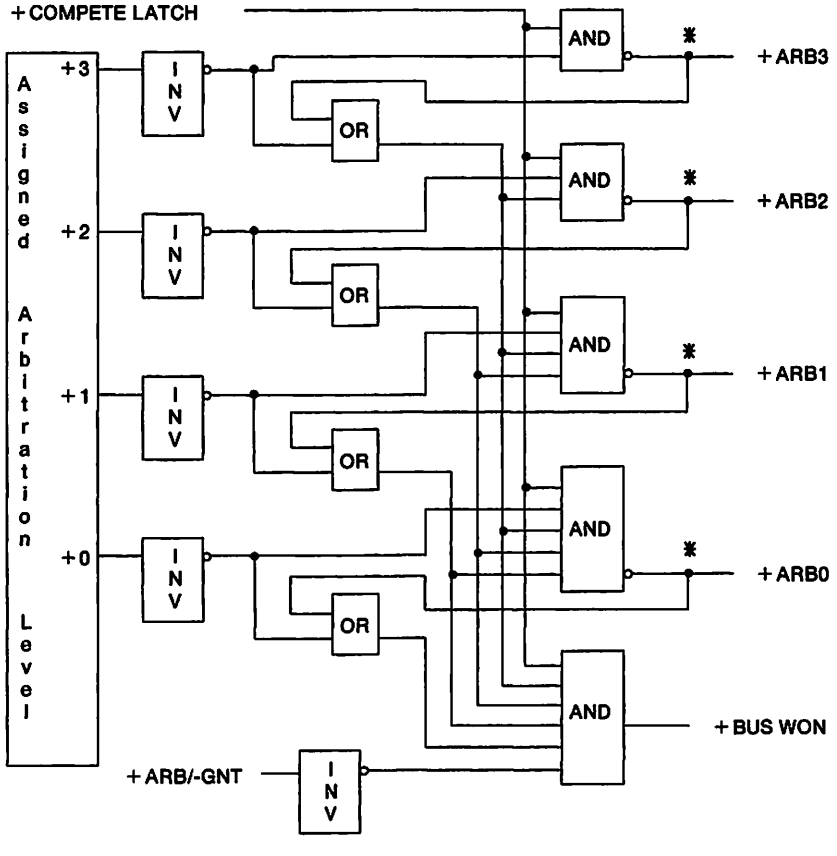
Figure 13. Arbitration Block Diagram

Local Arbiters

Participants requesting the use of the channel implement logic that allows all competing participants to recognize the winner. This logic is known as a local arbiter. An arbitrating participant competes for control of the channel only if it has driven -PREEMPT active before ARB/-GNT goes to the ARB state. A competing local arbiter drives its arbitration level onto the arbitration bus, then compares its arbitration level, on a bit-by-bit basis, with the value appearing on the arbitration bus beginning with the most significant bit, ARB3. If the competing local arbiter detects a mismatch on one of the bits, it immediately stops driving its less-significant bits. If the local arbiter subsequently recognizes a match on that bit, it resumes driving its less-significant bits until another mismatch is detected. The following is an example of an arbitration cycle.

1. Two local arbiters with arbitration levels 1010 and 0101 (hex A and 5) compete for the channel. Both local arbiters sense ARB/-GNT being driven to the ARB state and drive their arbitration levels on the bus, which now appears as 0000.
2. The first local arbiter (1010) detects a mismatch on ARB3 and stops driving all less-significant arbitration bits (ARB2 and ARB0 in this case).
3. The second local arbiter (0101) detects a mismatch on ARB2 and stops driving its less-significant arbitration bit (ARB1, in this case). The arbitration bus now shows 0111.
4. The second local arbiter now sees a match on ARB2 and resumes driving ARB1 of the arbitration bus.
5. The arbitration bus now shows a value of 0101, and the second local arbiter wins the arbitration cycle when ARB/-GNT is driven to the -GNT state.
6. The winning local arbiter continues to drive its arbitration level on the arbitration bus throughout the grant period until the channel is released and ARB/-GNT is driven to the ARB state.

The following is a simplified example of a local arbiter using combinational logic. If ARB/-GNT goes to the ARB state, +BUS WON is not true.



* Open-Collector Driven

Figure 14. Local Arbiter Example

Burst Transfer

To use burst transfer, a controlling master or a DMA slave activates -BURST after being granted the channel. -BURST is held active until after the start of the last cycle of the burst transfer. The following diagram shows a burst transfer.

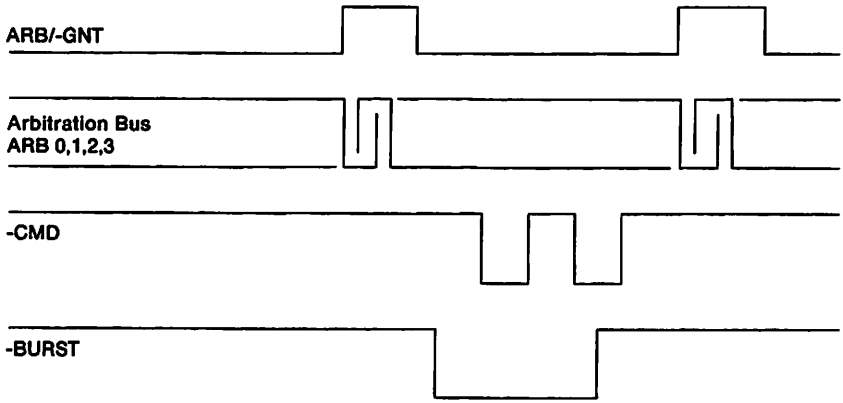


Figure 15. Burst Transfer

Preemption

When an arbitrating participant requires the channel, it activates -PREEMPT. The following diagram shows -PREEMPT occurring during a burst transfer.

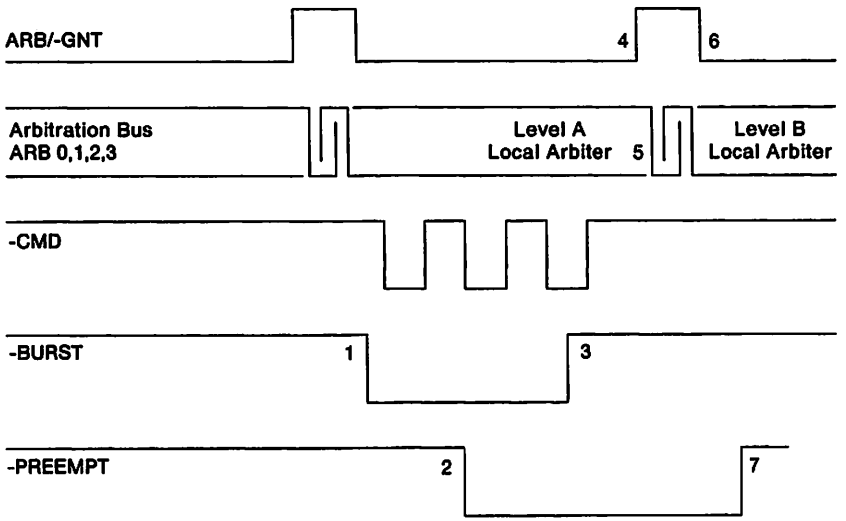


Figure 16. Preempted Burst Transfer

The preempted burst transfer sequence is as follows:

1. Participant A has control of the channel and has driven -BURST active.
2. Participant B drives -PREEMPT active.
3. Participant A, still in control of the channel, recognizes -PREEMPT active and signals the EOT condition within 7.8 microseconds. Participant A does not participate in the next arbitration cycle if its fairness feature is enabled. (See "Fairness Feature and the Inactive State" on page 34.)
4. When the central arbitration control point recognizes the EOT, it drives ARB/-GNT to the ARB state.
5. The arbitration cycle begins.
6. When ARB/-GNT is in the -GNT state, Participant B gains control of the channel.
7. Participant B, the preempting participant, removes -PREEMPT in response to gaining control of the channel.

Arbitrating masters should maximize use of the channel within the 7.8 microsecond limit from -PREEMPT going active. If a master or a DMA slave *does not* release the channel by presenting EOT within 7.8 microseconds after -PREEMPT is driven active, a channel time-out can occur, creating an exception condition.

Fairness Feature and the Inactive State

All bursting participants must support the fairness feature. Fairness forces the bursting participant to equitably share the channel time with the other participants. An optional configuration field in programmable option select (POS) can be used to disable the fairness feature.

When a bursting participant with control of the channel senses an active -PREEMPT, it must release the channel within the specified time (even if the fairness feature is disabled). If its fairness feature is enabled, the participant must enter the *inactive state* unless -PREEMPT was inactive while that participant owned the channel. If -PREEMPT was inactive while that participant had control, it is not required to enter the inactive state but must release the channel. In the inactive state, the bursting participant will not drive -PREEMPT active or participate in arbitration.

To exit the inactive state, the arbitrating participant must wait for -PREEMPT to go inactive. The participant then waits for a specified time (T42B) after the trailing edge of status (status active to inactive) before driving -PREEMPT active. It is possible for multiple arbitrating participants to be in the inactive state concurrently.

The following figure shows exiting from inactive state.

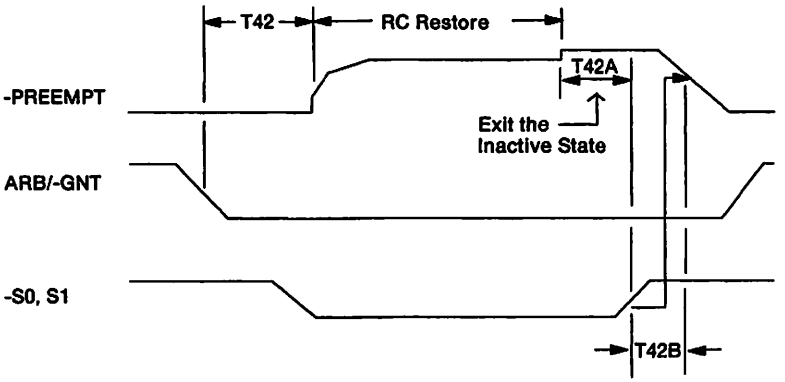


Figure 17. Exiting from the Inactive State

Note: The participant exits the inactive state independently of the state of ARB/-GNT.

Arbitration Timing

The following figures show the signal sequences and the timing specifications of the arbitration procedure.

After ARB/-GNT goes to the -GNT state and until EOT, the data transfer procedures are used to transfer data. See "Basic Transfer" on page 40, "Streaming Data Transfer" on page 58, and "DMA Procedure" on page 84 for the signal sequences and timing specifications.

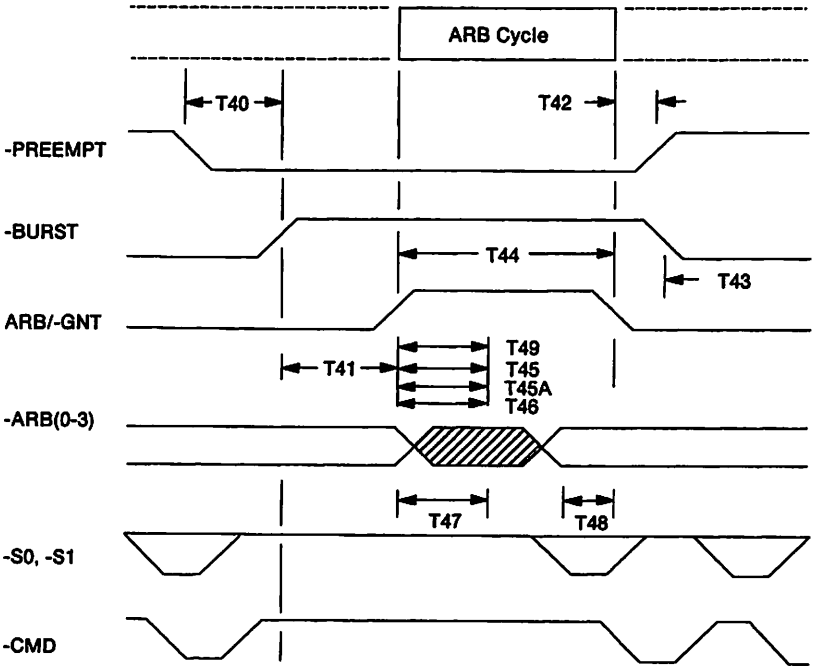


Figure 18. Arbitration Cycle

	Timing Parameter	Min/Max	Note
T40	-PREEMPT active to EOT (channel release)	0 / 7.8 μs	1
T41	ARB/-GNT in the ARB state from EOT (channel release)	30 / - ns	6,9
T42	-PREEMPT inactive from ARB/-GNT in -GNT state	0 / 50 ns	8
T42A	-PREEMPT inactive to status inactive	20 / - ns	5
T42B	-PREEMPT active from status inactive (Compete for the channel after the inactive state)	10 / - ns	
T43	-BURST active from ARB/-GNT in -GNT state	- / 50 ns	4
T44	ARB/-GNT in ARB state	100 / - ns	2
T45	Driver turn-on delay from ARB/-GNT in ARB state	0 / 50 ns	3
T45A	Driver turn-on delay from lower priority line	0 / 50 ns	3
T46	Driver turn-off delay from ARB/-GNT in ARB state	0 / 50 ns	3
T47	Driver turn-off delay from higher priority line	0 / 50 ns	3
T48	Arbitration bus stable before ARB/-GNT in -GNT state	10 / - ns	
T49	Tri-state drivers from ARB/-GNT in ARB state	- / 50 ns	7

Figure 19. Arbitration Cycle Timing Specifications

Notes:

1. This parameter applies to all masters and DMA slaves driving -BURST active. This parameter ensures that time-critical processes (such as refresh) can gain control of the channel resources. The value of the parameter is a requirement for the adapter designer and not a channel timeout requirement.
2. The value shown applies only to special case implementation involving the central arbitration control point and is provided for pulse width consideration only.
3. T45, T45A, T46 and T47 must be satisfied by the ARB(0-3) drivers of all arbitrating participants.
4. This parameter applies to all DMA slaves and bus masters when they are granted control of the channel.
5. This parameter is met indirectly by meeting T43D and T42 and includes the RC restore time on the bus, assuming no other participant is driving -PREEMPT active during this time. This delay allows participants to detect inactive -PREEMPT and subsequently exit the inactive state.
6. Because no maximum is specified, each non-default master must maintain the EOT on the channel until ARB/-GNT is in the ARB state.
7. This timing applies only to the controlling master as it is releasing control of the channel.
8. The RC restore of -PREEMPT will only occur at this time if no other participant is driving -PREEMPT active. If the system master performs cycles when ARB/-GNT is in the ARB state, -PREEMPT might be released while ARB/-GNT is in the ARB state.
9. System masters may follow this timing parameter, but are also allowed to follow procedures described under "System Master and Arbitration" on page 39.

The following are the timing parameters for the first cycle after ARB/-GNT goes to the -GNT state.

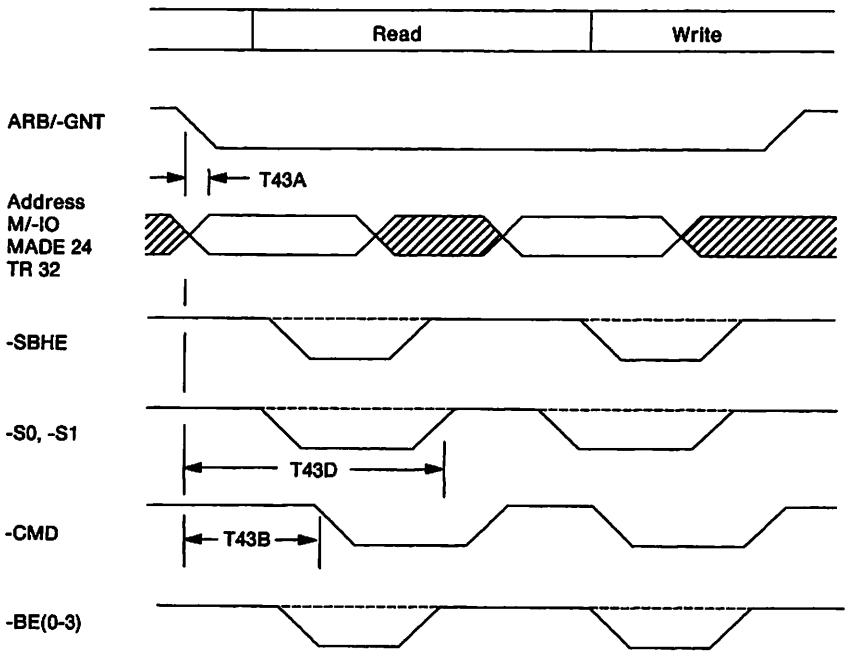


Figure 20. First Cycle After Grant

	Timing Parameter	Min/Max	Note
T43A	Address bus valid from ARB/-GNT in the -GNT state	0 / - ns	2
T43B	-CMD active from ARB/-GNT in the -GNT state	115 / - ns	2
T43C	DMA controller delay	30 / - ns	1
T43D	Status inactive from ARB/-GNT in the -GNT state	145 / - ns	2

Figure 21. First-Cycle-After-Grant Timing Specifications

Notes:

1. A DMA controller must allow a minimum time (T43C) after the grant for a slave to generate an internal acknowledgment that it has been selected. During the first cycle, the DMA controller must allow this additional time before sampling -DS 16/32 RTN and CHRDYRTN if it places an address on the address bus within 30 nanoseconds after grant. However, if the DMA controller drives the address 30 nanoseconds after grant, the additional delay is not needed.
2. System masters may follow this timing parameter, but are also allowed to follow procedures described under "System Master and Arbitration" on page 39.

System Master and Arbitration

The system master and the central arbitration control point can work together to improve use of the channel by overlapping a system master data transfer with arbitration as follows.

When the system master is the controlling master and -PREEMPT is active, ARB/-GNT can be driven to the ARB state. After the arbitration cycle is completed and the system master has signalled EOT, ARB/-GNT is driven to the -GNT state.

The system master can execute cycles during arbitration. When the arbitration cycle is completed, ARB0 - ARB3 can be checked to see if the winning arbitration level is the system master's arbitration level. If the winning arbitration level is the system master's arbitration level, ARB/-GNT can be driven to the -GNT state while the system master is executing a bus operation.

Data Transfer

The data transfer procedures are used to transfer data between a controlling master and the selected slave. The three types of data transfer procedures are:

- Basic Transfer
- Streaming Data Transfer
- DMA Transfer.

Each procedure defines the signal sequences and the signal timing specifications used in the procedure. Data transfer occurs only during the data transfer cycle defined by the procedure. A cycle is one or more data transfers, occurring during the time between the leading edge of -CMD and the trailing edge of -CMD. The data transfer cycle address selects either an I/O-address-space address or a memory-address-space address, under the control of M/-IO.

A controlling master or a DMA slave can drive -BURST, to indicate to the central arbitration control point that one or more data transfer cycles will be used before the EOT occurs. Refer to "Burst Transfer" on page 32 for more information.

This section describes each procedure, the signal sequence for each procedure, and the signal timing specifications.

The system-specific technical manuals contain the signal sequence and timing specifications for matched-memory signals and auxiliary-video signals.

Exception condition handling and reporting descriptions are contained in "Exception Condition Reporting and Handling" on page 118.

Basic Transfer

The three types of basic transfer cycles are:

- Default Cycle
- Synchronous Extended Cycle
- Asynchronous Extended Cycle.

Basic Transfer Procedure

All masters and DMA controllers transfer data with the same control sequence. The following covers the case of a write cycle. The signals appear in the following sequence:

1. The master drives the address bus, MADE 24, M/-IO, -APAREN, APAR(0-3) and -REFRESH (if applicable) valid, beginning the cycle.
2. In response to an unlatched decode of address, MADE 24, and M/-IO, the selected slave returns:
   - -CD SFDBK
   - -CD DS 16 (if the addressed data port is capable of 16-bit operations)
   - -CD DS 16 and -CD DS 32 (if the addressed data port is capable of 32-bit operations)
3. The master drives -S0 active (-S1 inactive).
4. In response to an unlatched decode of address, MADE 24, M/-IO, and status, the selected slave drives CD CHRDY inactive if the cycle is to be extended.
5. The master drives -ADL active. A slave can latch decodes of address, status, and M/-IO at the leading or trailing edge of -ADL. A DMA slave can also be selected by a latched decode of the same signals, using the arbitration level in place of the address.
6. The master drives the write data onto the data bus and, if appropriate, drives -DPAREN active and DPAR(0-3) valid.
7. The master drives -CMD active and -ADL inactive. A slave must latch the decodes of address, status, and M/-IO with the leading edge of -CMD if they were not latched with -ADL.
8. If CD CHRDY has been driven inactive, the cycle is extended until CD CHRDY is driven active. The signal must not be held inactive longer than specified.
9. The status signals become inactive and the address bus can change in preparation for the next cycle.
10. In response to an address change, the slave updates the state of:
    - -CD SFDBK
    - -CD DS 16
    - -CD DS 32.
11. The status signals and M/-IO can become valid in preparation for the next cycle.

12. -CMD goes inactive and the selected slave receives the data from the data bus. The data transfer cycle is complete.
13. The selected slave degates the data bus and data transfer control signals when it is no longer selected (status and -CMD are inactive).

Note: The changes to the address bus and status (-S0, -S1) can be overlapped with the preceding cycle to minimize the impact of the memory access time.

The sequence for the basic transfer procedure is as follows.

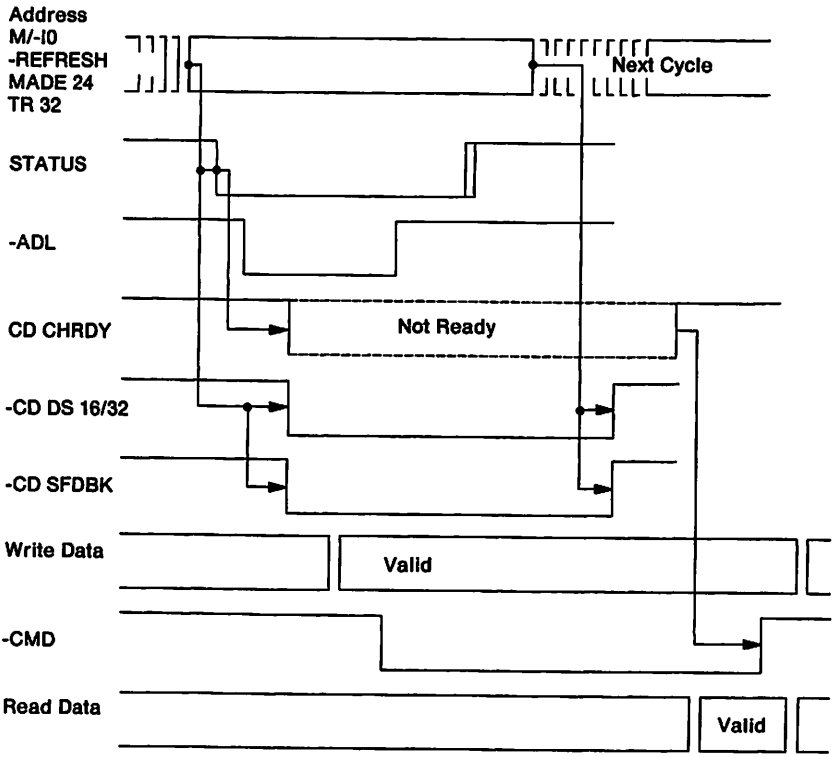


Figure 22. Basic Transfer Procedure - Overview

Whether a default, a synchronous-extended, or an asynchronous-extended cycle is performed depends on how a slave uses CD CHRDY.

- A default cycle occurs when a slave does not drive CD CHRDY inactive.

- A synchronous-extended cycle occurs when a slave drives CD CHRDY inactive, then releases CD CHRDY synchronously within the specified time after the leading edge of -CMD. The slave provides the read data within the specified time from -CMD active.
- An asynchronous-extended cycle occurs when a slave drives the CD CHRDY inactive, then releases CD CHRDY asynchronously.

Abort Cycle

If a master begins a transfer cycle but must abort that cycle, -ADL and -CMD must *not* be activated, and -S0, -S1 must be activated with a minimum pulse width T2A (see Figure 24 on page 45 for the T2A timing specification). The T2A time allows the central arbitration control point to recognize EOT. -BURST, if active, is deactivated with status deactivation. Slaves must not latch CD CHRDY inactive until after the appearance of an active -ADL signal, and if the cycle is aborted, must release CD CHRDY with the deactivation of status.

Default Cycle Timing (200 ns Minimum)

This section provides the specifications for the timing parameters for the I/O and memory default transfer cycle.

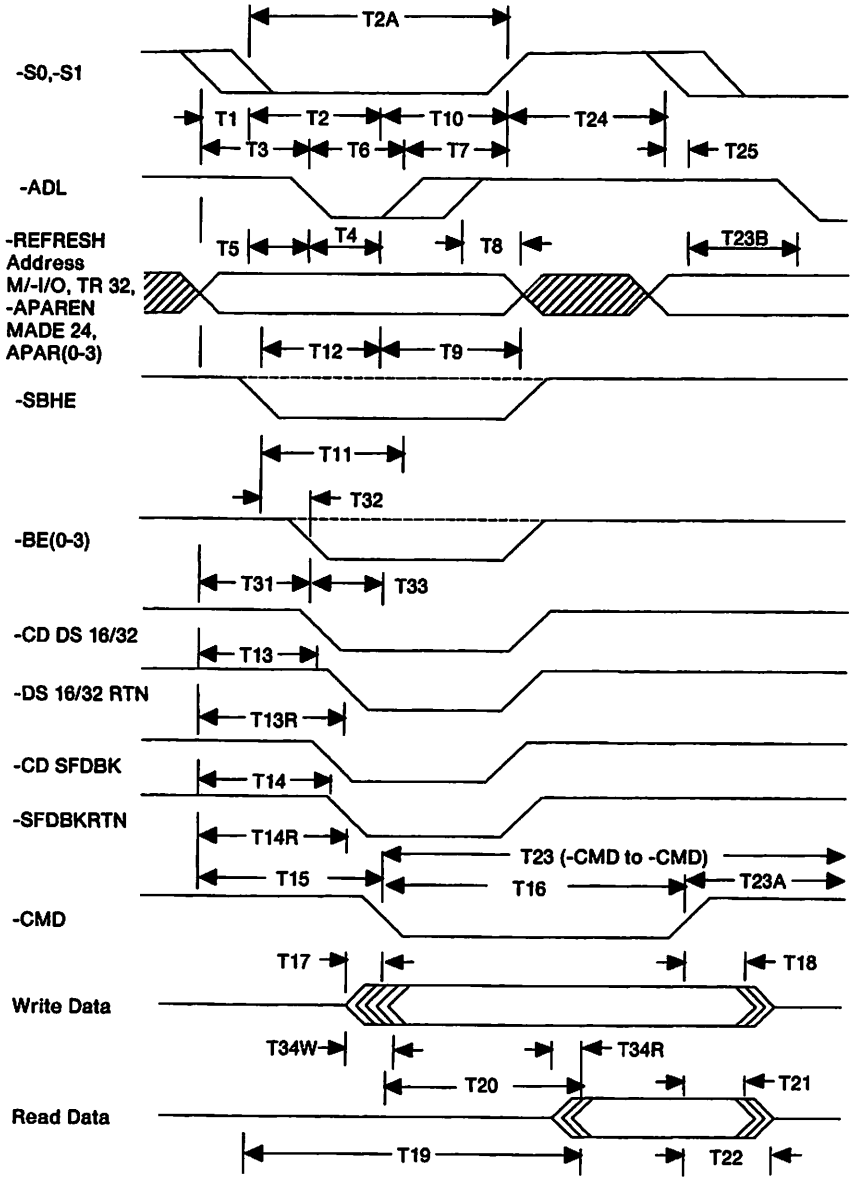


Figure 23. Default Cycle (200 ns Minimum)

	Timing Parameter	Min/Max	Note
T1	Status active from Address, M/-IO, -APAREN, -REFRESH, MADE 24, TR 32, and APAR(0-3) valid	10 / - ns	13
T2	-CMD active from status active	55 / - ns	4
T2A	Status pulse width of an aborted cycle	85 / - ns	10
T3	-ADL active from Address, M/-IO, -APAREN, -REFRESH, MADE 24, TR 32, and APAR(0-3) valid	45 / - ns	
T4	-ADL active to -CMD active	40 / - ns	
T5	-ADL active from status active	12 / - ns	
T6	-ADL pulse width	40 / - ns	
T7	Status hold from -ADL inactive	25 / - ns	4,11
T8	Address, M/-IO, -REFRESH, -SBHE, -BE(0-3), MADE 24, TR 32, APAR(0-3), -APAREN hold from -ADL inactive	25 / - ns	4,13
T9	Address, M/-IO, -REFRESH, -SBHE, -BE(0-3), MADE 24, TR 32, APAR(0-3), -APAREN hold from -CMD active	30 / - ns	2,3,4, 13
T10	Status hold from -CMD active	30 / - ns	4,11
T11	-SBHE setup to -ADL inactive	40 / - ns	4
T12	-SBHE setup to -CMD active	40 / - ns	4
T13	-CD DS 16/32 active from Address Bus and M/-IO valid	- / 55 ns	2,6
T13R	-DS 16/32 RTN active from Address Bus and M/-IO valid	- / 75 ns	2
T14	-CD SFDBK active from Address Bus and M/-IO valid	- / 60 ns	1,2
T14R	-SFDBKRTN active from Address Bus and M/-IO valid	- / 80 ns	2
T15	-CMD active from Address valid	85 / - ns	4
T16	-CMD pulse width	90 / - ns	
T17	Write Data setup to -CMD active	0 / - ns	5,12
T18	Write Data hold from -CMD inactive	30 / - ns	5
T19	Status active to Read Data valid (Access Time)	- / 125 ns	5,8
T20	Read Data valid from -CMD active	- / 60 ns	5
T21	Read Data hold from -CMD inactive	0 / - ns	5
T22	Read data bus tri-state from -CMD inactive	- / 40 ns	5
T23	-CMD active to next -CMD active	190 / - ns	
T23A	-CMD inactive to next -CMD active	80 / - ns	
T23B	-CMD inactive to next -ADL active	40 / - ns	
T24	Next status active from status inactive	30 / - ns	
T25	Next status active to -CMD inactive	- / 20 ns	
T31	-BE(0-3) active from Address valid (32-bit masters only)	- / 40 ns	
T32	-BE(0-3) active from -SBHE, A0, A1 active	- / 30 ns	
T33	-BE(0-3) active to -CMD active	10 / - ns	
T34R	Centralized steering logic Read Data buffer delay	0 / 15 ns	9
T34W	Centralized steering logic Write Data buffer delay	0 / 15 ns	9

Figure 24. Default Cycle (200 ns Minimum) Timing Specifications

Notes:

1. Slaves drive -CD SFDBK when selected by either arbitration or address. The slave does not drive -CD SFDBK active when selected by -CD SETUP.
2. The master must hold the address bus and associated signals valid until -DS 16 RTN, -DS 32 RTN, and -SFDBKRTN are sampled, in addition to meeting T8 and T9. The slave must continue to drive -CD DS 16/32 and -CD SFDBK as long as its address is valid.
3. A master can deactivate status and address concurrently. This can result in changes on the address bus while status is still active at the slave. CD CHRDY can change states during changes on the address bus.
4. It is recommended that slaves use transparent latches to latch information with the leading edge of -CMD, or with the leading or trailing edge of -ADL.
5. When data parity is supported, the implemented data parity signals *must be* active and stable on the bus for the data to be considered valid.
6. -CD DS 16/32 and -CD SFDBK must be driven by unlatched address decodes because the next address can be driven valid in the current cycle.
7. When address parity is supported, the implemented address parity signals *must be* active and stable for the address to be considered valid.
8. T19 is a valid timing parameter *only* when T2 is less than 65 nanoseconds.
9. The centralized steering delay applies when a 16-bit master accesses a 32-bit data port. The slave must compensate for the centralized steering delay, as required.
10. Masters must provide this minimum timing for the central arbitration control point and for inactive state exit. The system master can provide a shorter T2A that is compatible with the central arbitration control point. See "Fairness Feature and the Inactive State" on page 34 for inactive state exit requirements (T42A). Other participants must not depend on a minimum T2A time during an abort cycle.
11. A bus master must deactivate -S(0,1) as soon as possible after the hold time specified by T7 and T10, and prior to the deactivation of -CMD. Additional rules apply for extended cycles and streaming data cycles.

12. If data parity is used, see T96 timing in Figure 71 on page 123.
13. TR 32 can be held inactive between cycles by a 32-bit master. TR 32 timings are provided to define when it must be valid on the first and last cycle.

Synchronous-Extended Cycle Timing (300 ns Minimum)

A synchronous-extended cycle occurs when a slave drives CD CHRDY inactive, then releases CD CHRDY synchronously within the specified time after the leading edge of -CMD.

The timing sequence is shown in the following figure.

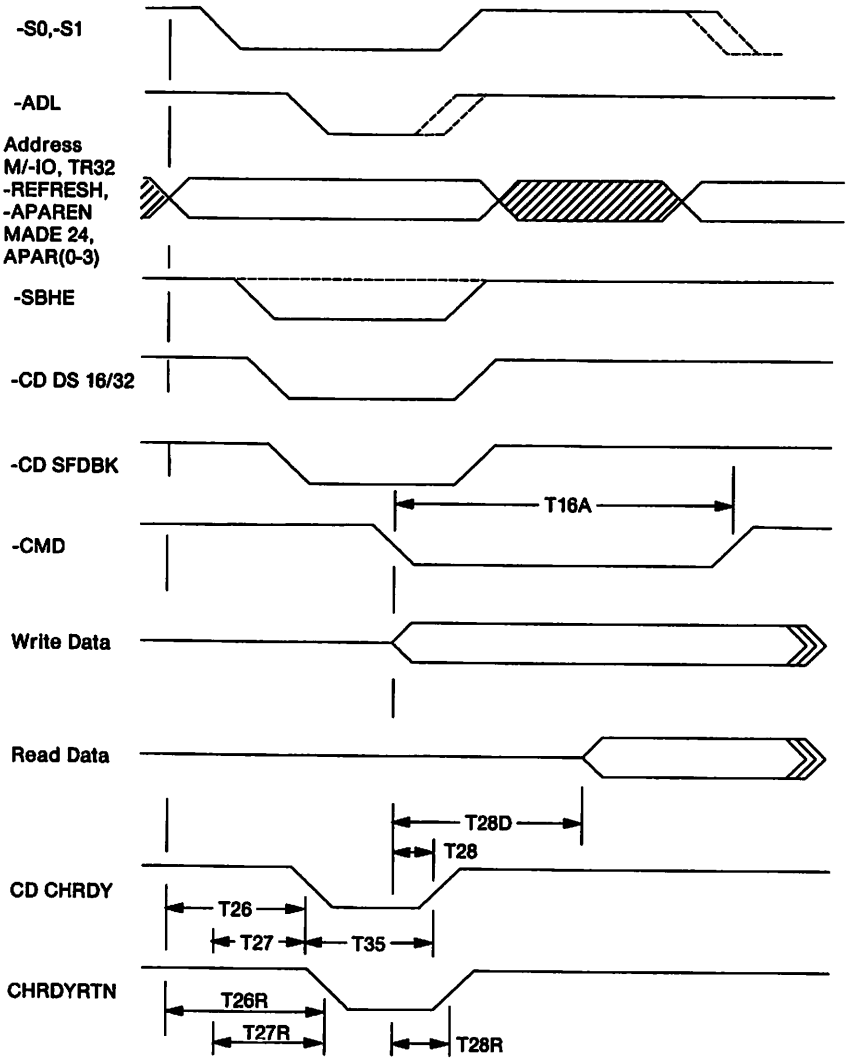


Figure 25. Synchronous-Extended Cycle (300 ns Minimum)

	Timing Parameter	Min/Max	Note
T16A	-CMD pulse width	190 / - ns	
T26	CD CHRDY inactive from Address Bus valid	- / 60 ns	3
T26R	CHRDYRTN inactive from Address Bus valid	- / 80 ns	3
T27	CD CHRDY inactive from status valid and previous -CMD inactive	0 / 30 ns	3,4
T27R	CHRDYRTN inactive from status valid and previous -CMD inactive	0 / 50 ns	3,4
T28	CD CHRDY active from -CMD active	0 / 30 ns	1
T28R	CHRDYRTN active from -CMD active	0 / 50 ns	
T28D	Read Data valid from -CMD active (used with T28)	0 / 160 ns	
T35	CD CHRDY active from CD CHRDY inactive	0 / 3.5μs	5

This table shows only the parameters that are additional to those of the default cycle. All other parameters are the same as the default cycle.

Figure 26. Synchronous-Extended Cycle Timing Specifications (300 ns Minimum)

Notes:

1. CD CHRDY is released by a slave performing a 300 nanoseconds extended cycle within the specified time after the leading edge of -CMD.
2. A master can deactivate status and address concurrently. This can result in changes on the address bus while status is still active at the slave. CD CHRDY can change states during changes on the address bus.
3. T27 and T27R apply *only* when T1 is greater than 30 nanoseconds. T26 and T26R applies *only* when T1 is less than or equal to 30 nanoseconds.
4. Masters are responsible for controlling their own bus activity. When a master uses overlapped cycles, it *must* be capable of handling (or ignoring) CD CHRDY from the next slave while terminating the cycle with the current slave. Once the current cycle is terminated, the master must act appropriately to an inactive CD CHRDY signal. Masters must also recognize and handle any added delay in a slaves deactivation of CD CHRDY caused by overlapped cycles.
5. A bus master must deactivate -S0,-S1 as specified in the default cycle, and this deactivation must be independent of CHRDYRTN. Slaves should not depend on the removal of -S0,-S1 for the deactivation of CD CHRDY.
6. When address parity is supported, the address parity signals (APAR 0-3) *must* be active and stable for the address to be considered valid.

Asynchronous-Extended Cycle Timing (≥300 ns)

An asynchronous-extended cycle occurs when a slave drives CD CHRDY inactive, then releases CD CHRDY asynchronously. The timing sequence is shown in the following figures.

	Timing Parameter	Min/Max	Note
T16A	-CMD pulse width	190 / - ns	
T26	CD CHRDY inactive from Address Bus valid	- / 60 ns	3
T26R	CHRDYRTN inactive from Address Bus valid	- / 80 ns	3
T27	CD CHRDY inactive from status valid and previous -CMD inactive	0 / 30 ns	3,4
T27R	CHRDYRTN inactive from status valid and previous -CMD inactive	0 / 50 ns	3,4
T29S	Read Data from slave valid from CD CHRDY active	- / 60 ns	1
T29M	Read Data valid to master from CHRDYRTN active	- / 60 ns	1
T29A	-CMD inactive from CHRDYRTN active	60 / - ns	
T35	CD CHRDY active from CD CHRDY inactive	0 /3.5μs	5

This table shows only the parameters that are additional to those of the default cycle. All other parameters are the same as the default cycle.

Figure 27. Asynchronous-Extended Cycle Timing Specifications (≥300 ns)

Notes:

1. CD CHRDY is released asynchronously by a slave performing a ≥300-nanosecond cycle. The slave must present the read data within the time specified after the release of CD CHRDY.
2. A master can deactivate status and address concurrently. This can result in changes on the address bus while status is still active at the slave. CD CHRDY can change states during changes on the address bus.
3. T27 and T27R apply *only* when T1 is greater than 30 nanoseconds. T26 and T26R apply *only* when T1 is less than or equal to 30 nanoseconds.

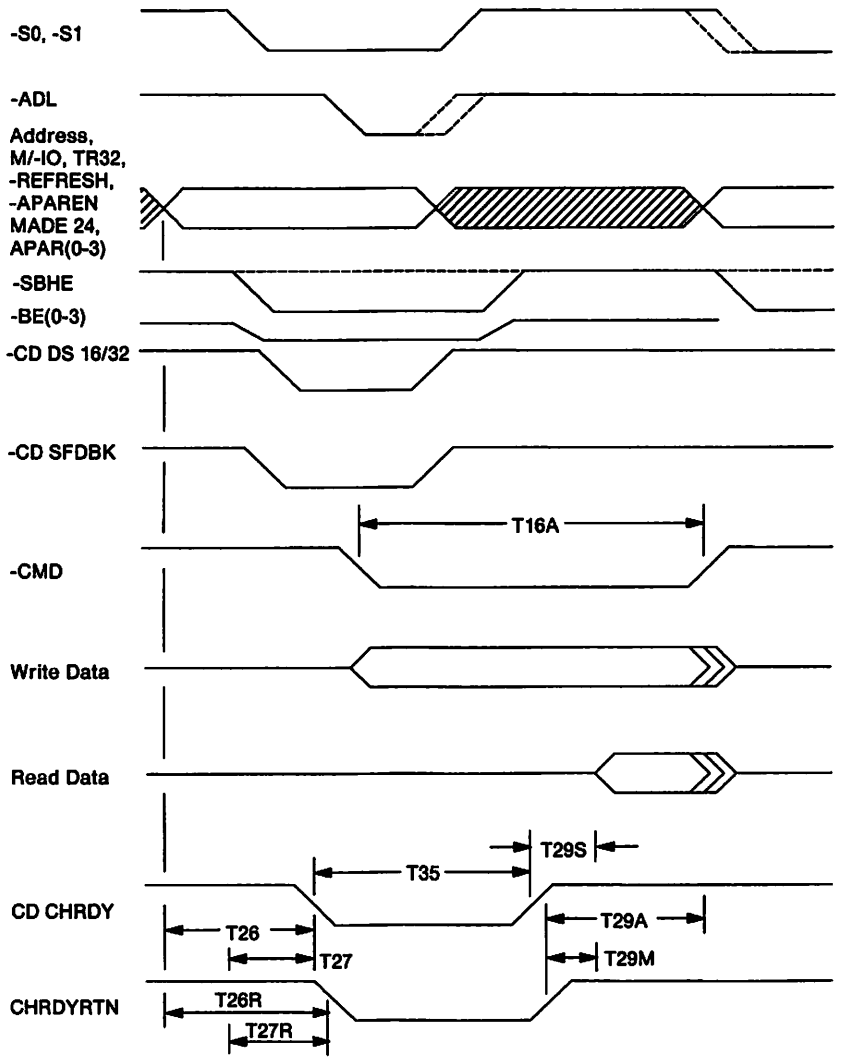


Figure 28. Asynchronous Extended Cycle (≥ 300 ns)

4. Masters are responsible for controlling their own bus activity. When a master uses overlapped cycles, it *must* be capable of handling (or ignoring) CD CHRDY from the next slave while terminating the cycle with the current slave. Once the current cycle is terminated, the master must act appropriately to an inactive CD CHRDY signal. Masters must also recognize and handle the potential added delay in a slave's deactivation of CD CHRDY due to overlapped cycles.
5. A bus master must deactivate -S0, -S1 as specified in the default cycle, and this deactivation must be independent of CHRDYRTN. Slaves should not have dependencies on removal of -S0, -S1 for the deactivation of CD CHRDY.
6. When address parity is supported, the implemented address parity signals (APAR 0-3) *must* be active and stable for the address to be considered valid.

Data Steering

During data transfer procedures, data can be transferred between a 16-bit or 32-bit master, and an 8-, 16-, or 32-bit slave. The following table shows the relationship between the data bus, associated signals, and the number of cycles required for the combinations of data transfer and steering.

		S32/32-bit Data Port -CD DS 32 Active -CD DS 16 Active								S16/16-bit Data Port -CD DS 32 Inactive -CD DS 16 Active							S8/8-bit Data Port -CD DS 32 Inactive -CD DS 16 Inactive					
	Data Trans Size	-BE3	-BE2	-BE1	-BE0	D3	D2	D1	D0	-SBHE	A1	A0	D3	D2	D1	D0	-SBHE	A1	A0	D3	D2	D1	D0
M32 TR 32 =0	4 Byte	0	0	0	0	D	C	B	A	0	0	0	-	-	B	A	-	0	0	-	-	-	A
										0	1	0	-	-	D	C	-	0	1	-	-	-	B
																	-	1	0	-	-	-	C
																	-	1	1	-	-	-	D
	3 Byte	1	0	0	0	-	C	B	A	0	0	0	-	-	B	A	-	0	0	-	-	-	A
										1	1	0	-	-	-	C	-	0	1	-	-	-	B
																	-	1	0	-	-	-	C
		0	0	0	1	D	C	B	-	0	0	1	-	-	B	-	-	0	1	-	-	-	B
										0	1	0	-	-	D	C	-	1	0	-	-	-	C
																	-	1	1	-	-	-	D
	2 Byte	1	1	0	0	-	-	B	A	0	0	0	-	-	B	A	-	0	0	-	-	-	A
																	-	0	1	-	-	-	B
		1	0	0	1	-	C	B	-	0	0	1	-	-	B	-	-	0	1	-	-	-	B
										1	1	0	-	-	-	C	-	1	0	-	-	-	C
		0	0	1	1	D	C	-	-	0	1	0	-	-	D	C	-	1	0	-	-	-	C
																	-	1	1	-	-	-	D
	1 Byte	1	1	1	0	-	-	-	A	1	0	0	-	-	-	A	-	0	0	-	-	-	A
		1	1	0	1	-	-	B	-	0	0	1	-	-	B	-	-	0	1	-	-	-	B
		1	0	1	1	-	C	-	-	1	1	0	-	-	-	C	-	1	0	-	-	-	C
		0	1	1	1	D	-	-	-	0	1	1	-	-	D	-	-	1	1	-	-	-	D
M16 TR 32 =1	2 Byte	1	1	0	0	-	-	B	A	0	0	0	-	-	B	A	-	0	0	-	-	-	A
																	-	0	1	-	-	-	B
		0	0	1	1	D	C	-	-	0	1	0	-	-	D	C	-	1	0	-	-	-	C
																	-	1	1	-	-	-	D
	1 Byte	1	1	1	0	-	-	-	A	1	0	0	-	-	-	A	-	0	0	-	-	-	A
		1	1	0	1	-	-	B	-	0	0	1	-	-	B	-	-	0	1	-	-	-	B
		1	0	1	1	-	C	-	-	1	1	0	-	-		C	-	1	0	-	-	-	C
		0	1	1	1	D	-	-	-	0	1	1	-	-	D	-	-	1	1	-	-	-	D

-Key-

A,B,C,D - Byte Transferred
- - Not Used by the Slave
M32 - 32-bit Master
M16 - 16-bit Master

S32 - 32-bit Slave
S16 - 16-bit Slave
S8 - 8-bit Slave

D0 - D(0-7)
D1 - D(8-15)
D2 - D(16-23)
D3 - D(24-31)

Figure 29. Steering Control

Notes:

1. Masters are responsible for generating and controlling the extra cycles needed to complete the transfer.
2. If TR 32 = 1, the central translator logic drives -BE(0-3) signals. The central steering logic steers data bus byte D(0-7) to or from data bus byte D(16-23) and data bus byte D(8-15) to or from data bus byte D(24-31).
3. If TR 32 = 0, the 32-bit master (M32) drives -BE(0-3) signals.
4. A -BE0, -BE1, -BE2, or -BE3 signal equal to 0 indicates a valid data byte in the corresponding data bus byte D(0-7), D(8-15), D(16-23), or D(24-31).
5. A 1 denotes the most-positive logic state of the signal. A 0 denotes the least-positive logic state of the signal.
6. Combinations of -SBHE, A1, A0 not shown in the table are not permitted.

Data Steering Control

The following table shows which participant is responsible for data-steering control during basic transfer, DMA, and setup procedures.

	S32	S16	S8
M16	Central Steering	No Steering	M16
M32	No Steering	M32	M32
	M32 - 32-bit Master M16 - 16-bit Master	S32 - 32-bit Slave S16 - 16-bit Slave S8 - 8-bit Slave	

Figure 30. Steering Responsibility (Basic Transfer, DMA, and Setup Procedures)

The following table shows which participant is responsible for data-steering control during streaming data procedures.

	S32	S16
M16	S32	No Steering
M32	No Steering	M32
	M32 - 32-bit Master M16 - 16-bit Master	S32 - 32-bit Slave S16 - 16-bit Slave

Figure 31. Steering Responsibility (Streaming Data Procedure)

Master Replication of Data Bytes

The following are the requirements for master replication of data bytes during a write operation:

- A 16 or 32-bit master transferring one byte of data on D(8-15) replicates D(8-15) on D(0-7).
- A 32-bit master transferring two bytes of data on D(8-15) and D(16-23) or three bytes of data on D(8-15), D(16-23), and D(24-31) replicates D(8-15) on D(0-7).
- A 32-bit master transferring one byte of data on D(16-23) replicates D(16-23) on D(0-7).
- A 32-bit master transferring one byte of data on D(24-31) replicates D(24-31) on D(0-7) and D(8-15).
- A 32-bit master transferring two bytes of data on D(16-23) and D(24-31) replicates D(16-23) on D(0-7) and replicates D(24-31) on D(8-15).

Channel Support

The 32-bit data bus requires unique system logic to permit 16-bit masters to communicate with slaves with 32-bit data ports.

Address Bus Translator

A 32-bit slave uses signals -BE(0-3) as part of its address instead of A0, A1, and -SBHE. A 16-bit master does not provide -BE(0-3); the central translation logic generates them when a 16-bit master has control of the channel.

Central Data Steering

A 32-bit slave writes data to and reads data from data bits 0 through 31. A 16-bit master does not use data bits 16 through 31; the central steering logic must cross data over from the low 16 data lines (D0 through D15) to the high data lines (D16 through D31) and back at the appropriate times. The 32-bit slave must compensate for the added delay.

TR 32

TR 32 is driven inactive by 32-bit masters only. When TR 32 is active, it is used by:

- The central translator logic to drive -BE(0-3)
- The central steering logic to perform data steering.

When TR 32 is active, 32-bit slaves can use it to detect that the controlling master is not a 32-bit master and compensate for additional delay attributable to the central steering logic.

Central Steering Logic

Central steering logic uses A1, -SBHE, -S(0,1), -CD DS 16, and -CD DS 32 to steer data in support of 16-bit masters communicating with 32-bit slaves.

Central Translator Logic

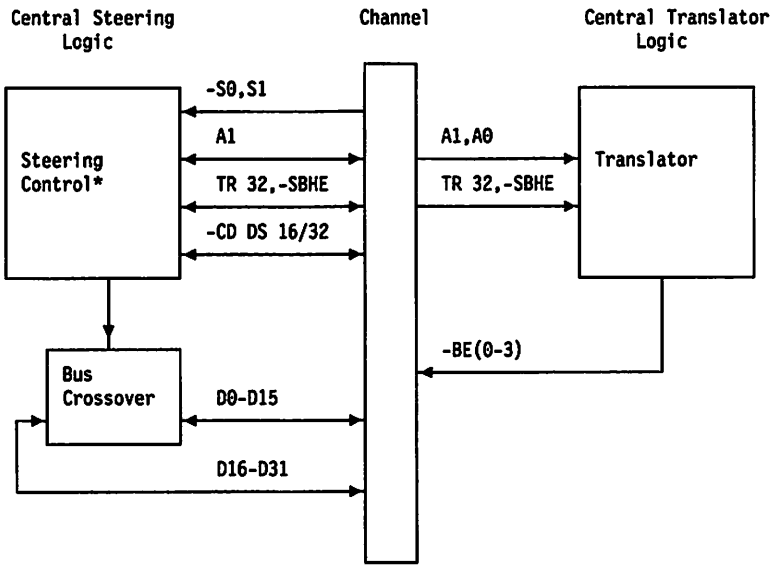
Central translator logic translates A0, A1, and -SBHE to -BE(0-3), when TR 32 is active to support transfers between 16-bit masters and 32-bit slaves.

-BE(0-3)

Signals -BE(0-3) are:

- Driven by a 32-bit controlling master.
- Created by the central translator logic when a 16-bit master is the controlling master.
- Used by 32-bit slaves only.

The following block diagram shows the implementation of data steering.



* For 16-bit devices to 32-bit devices

Figure 32. Data Bus Steering Implementation

Streaming Data Transfer

The streaming data procedure provides performance improvements over basic transfer procedures for block transfers, and supports data-transfer rates of up to 80MB per second. It provides for the transfer of a data block by using a single address followed by multiple 16-, 32-, or 64-bit data transfers within a single streaming data cycle. The data transfers are clock-synchronous and incorporate automatic speed matching between the controlling master and slave.

The streaming data procedure can be used for high speed data transfer between a controlling master and the selected slave. The streaming data procedure is transparent to devices that are not selected. Streaming data participants **must** support the basic transfer procedure to operate with nonstreaming-data participants.

Special rules apply to streaming data devices to ensure compatibility with nonstreaming-data participants. These include:

- A streaming-data-enable bit must be supported in the adapter POS space.

  Note: Streaming data participants must support a fully-functional operation with nonstreaming-data participants, independent of the state of the Streaming Data Enable field.

- After CHRESET, the default state of the streaming-data-enable bit is disabled.
- Both 16- and 32-bit streaming data transfers must start on four-byte address boundaries.
- A 64-bit streaming data transfer must be aligned on an eight-byte address boundary.

Data is validated by using CD CHRDY to indicate ready or not ready, so that the slave has the ability to pace the data transfer for 16- or 32-bit transfers. Either the master or slave can terminate the cycle at any time.

Implementing and using streaming data transfer is optional to masters. Implementing and indicating streaming data transfer capabilities is optional to slaves.

Streaming Data Procedure (16- or 32-Bit)

A 16- or 32-bit streaming data transfer cycle is initiated as either a 16- or 32-bit basic data transfer cycle, and all rules associated with those procedures apply. The following is an example of a 16- or 32-bit streaming data write cycle. The signals appear in the following sequence:

1. The master drives the address bus, MADE 24, M/-IO, -APAREN, and APAR(0-3) valid to begin the cycle.
2. In response to an unlatched decode of address, MADE 24, and M/-IO, the selected slave returns:
   - -CD SFDBK
   - -CD DS 16 (if the addressed data port is capable of 16-bit operations)
   - -CD DS 16 and -CD DS 32 (if the addressed data port is capable of 32-bit operations).
3. The master drives -S0 active (-S1 inactive).
4. In response to an unlatched decode of address, MADE 24, M/-IO, and status, the selected slave drives CD CHRDY inactive, if the cycle is to be extended.
5. A 32-bit master (or the central translator logic for a 16-bit master in a 32-bit system) drives the byte enable signals. -BE(0-3) is driven to binary 0011 by the central translator logic, or -BE(0-3) is driven to binary 0000 by a 32-bit master.

   Note: Because 16- and 32-bit streaming data operations are aligned on four-byte boundaries, A0, A1, and -SBHE equal a binary 000.

6. The master drives -ADL valid. A slave can latch decodes of address, status, and M/-IO at the leading or trailing edge of -ADL.
7. The slave drives -SDR(0,1) active.

   Note: When the slave **does not** drive -SDR(0,1) active (indicating that it is defaulting to basic transfer procedures), the controlling master will drive -S(0,1) inactive after the hold time specified by timing parameter T7 and T10 (see Figure 24 on page 45).

8. The master drives the write data onto the bus and, if supported, drives -DPAREN active and DPAR(0-3) valid.
9. The master drives -CMD active and -ADL inactive. A slave must latch the decodes of address, status, and M/-IO with the leading edge of -CMD if they were not latched with -ADL.

10. If CD CHRDY has been driven inactive, the cycle is extended until CD CHRDY is driven active. CD CHRDY must not be held inactive longer than specified.
11. The controlling master indicates the ability to perform streaming data by starting -SD STROBE.

    Note: If CHRDYRTN is active, -SD STROBE can be activated concurrently with -CMD. -SD STROBE is used by both the controlling master and slave to transfer data, with data being clocked on and off the bus on the falling clock edge and clocked off the bus on the next falling clock edge. The operation proceeds with new data being placed on the bus every time -SD STROBE makes a high-to-low transition.
12. The master drives status inactive.
13. The slave drives -SDR(0,1) inactive.
14. The address bus can change in preparation for the next cycle.
15. In response to an address change, the slave updates the state of:
    - -CD SFDBK
    - -CD DS 16
    - -CD DS 32.
16. The status signals and M/-IO can become valid in preparation for the next cycle.
17. The master stops driving -SD STROBE.
18. On the last transfer, the controlling master drives -CMD inactive, and the selected slave gates the data off the bus. The transfer cycle is complete.

Streaming Data Procedure (64-Bit)

A 64-bit streaming data transfer cycle is initiated as a 32-bit basic transfer cycle, and all the rules associated with this procedure apply. The following is an example of a 64-bit streaming data write cycle. The signals appear in the following sequence:

1. The master drives the address bus, MADE24, M/-IO, -APAREN, and APAR(0-3) valid to begin the cycle.
2. In response to an unlatched decode of address, MADE24, and M/-IO, the selected slave returns:
    - -CD SFDBK
    - -CD DS 16
    - -CD DS 32.

3. The master drives -S0 active (-S1 inactive).
4. In response to an unlatched decode of address, MADE24, M/-IO, and status, the selected slave drives CD CHRDY inactive, if the cycle is to be extended.
5. A 32-bit master drives -BE(0-3) to binary 0000 (64-bit streaming data transfers are not performed by 16-bit masters).

   Note: Because 64-bit operations are aligned on an eight-byte boundary, A0, A1, and -SBHE equal binary 000.

6. The master drives -ADL active. A slave can latch decodes of address, status, and M/-IO at the leading or trailing edge of -ADL.
7. The slave drives -SDR(0,1) to indicate its transfer speed capability and drives -MSDR active to indicate 64-bit capability.

   Note: When the slave does not drive -SDR(0,1), and -MSDR active (indicating that it is defaulting to a basic transfer procedure), the controlling master drives -S(0,1) inactive after the hold time specified by timing parameters T7 and T10, and prior to driving -CMD active (see Figure 24 on page 45.)

8. The master drives the write data onto the data bus and, if supported, drives -DPAREN active and DPAR(0-3) valid.
9. The master drives -CMD active and -ADL inactive. A slave must latch the decodes of address, status, and M/-IO with the leading edge of -CMD if they were not latched with -ADL.
10. If CD CHRDY has been driven inactive, the cycle is extended until CD CHRDY is driven active. CD CHRDY must not be held inactive longer than specified.
11. The controlling master indicates the ability to perform streaming data transfers by activating -SD STROBE. The 64-bit streaming data transfer capability is indicated by driving -BE(0-3) inactive, driving the four most-significant bytes of the eight byte transfer on the address bus, and, if -DPAREN is active, driving APAR(0-3) valid. -APAREN is driven inactive.

    Note: -SD STROBE is used by both the controlling master and the slave to transfer data; data is clocked on and off the address bus and the data bus on the falling clock edge and clocked off the address bus and the data bus on the next falling clock edge. The operation proceeds with the new data being placed on the address bus and the data bus each time -SD STROBE makes a high-to-low transition.

12. Both the address bus and data bus are used for transferring data for the remainder of the cycle.
13. The status signals are driven inactive on the last high-to-low transition of -SD STROBE. The slave drives -SDR(0,1) inactive.
14. In response to data on the address bus, other slaves can drive -CD SFDBK, -CD DS 16, -CD DS 32, and CD CHRDY valid during the data transfer. The master should not monitor the state of these signals during the data transfer.
15. On the last transfer, the controlling master drives -CMD inactive and the slave gates the data off the address bus and the data bus. The cycle is complete.

Streaming Data Transfer Rates

The amount of data transferred per -SD STROBE period is constant during streaming data transfer. The number of bytes transferred in one transfer cycle are listed below.

Streaming Data Master Size	16-Bit Slave	32-Bit Slave	64-Bit Slave
16-Bit	2	2	2
32-Bit	2	4	4
64-Bit	2	4	8

Figure 33. Streaming Data Transfer Rates

Note: When a slave requests a streaming data transfer, it must be prepared to support a minimum of two transfers. When a master initiates a streaming data transfer, it must be prepared to support a minimum of one transfer.

Address Bus Management

During 16 and 32 bit streaming data transfers, the address on the address bus and TR 32 are not changed until -SDR(0,1) is driven inactive. The controlling master and the slave must provide memory-address-space address management and I/O-address-space address management as follows:

- Memory-address-space addresses

  The controlling master must keep a count of the number of bytes transferred, so that if early termination of the block transfer occurs, the master can assert the correct address (original address plus the number of bytes transferred) at the start of the next data transfer procedure.

The slave must manage the address, to ensure that the data being clocked off the bus is gated into the correct memory addresses.

- I/O-address-space addresses

  If early termination of the block transfer occurs, the master must present the same (original) address on the address bus when restarting the streaming data procedure.

  The slave must accommodate the streaming data transfer to the same I/O address.

During 64-bit streaming data transfers (after the address is removed from the address bus), the controlling master must ignore CHRDYRTN, -DS 16 RTN, -DS 32 RTN, and -SFDBKRTN. Slaves (except the selected slave) are decoding the address bus and responding as if selected.

Address Boundary Alignment

If the starting address is not a four-byte address boundary and a 16- or 32-bit streaming data procedure is going to be used, then the basic transfer procedure must be used to transfer data until the address has the correct boundary alignment. The controlling master must execute the basic transfer procedure to obtain address boundary alignment.

A slave can indicate streaming data capability, by driving -SDR(0,1) active, without checking address boundary alignment. Then the controlling master and the slave can use the streaming data procedure. The basic transfer procedure might also be required to complete the data transfer, even if the initial data address was a four-byte address boundary.

If the starting address is not on an eight-byte address boundary and a 64-bit streaming transfer procedure is going to be used basic transfer cycles are used to transfer data until the address has a correct boundary alignment. The basic transfer procedure might also be needed to complete the data transfer, even if the initial data address was a eight-byte address boundary.

The following is an example of a basic transfer procedure occurring before and after a 16- or 32-bit streaming data procedure. The controlling master should drive -BURST active to indicate that one or more consecutive data transfer cycles will occur. Figure 34 on page 64 shows the data block to be transferred.

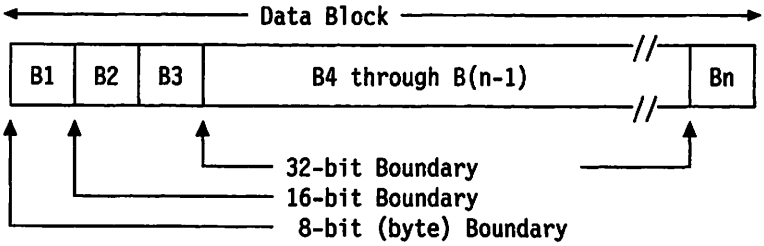


Figure 34. Data Block Example

In the above example, the data transfer procedures used to achieve address boundary alignment between a controlling master and the selected slave are:

- Byte 1 (B1) is transferred using a basic transfer procedure.
- Bytes 2 (B2) and 3 (B3) are transferred using a basic transfer procedure.

  The data address is now on a 32-bit boundary.
- Bytes B4 through B(n-1) are transferred using the streaming data procedure. Each period of -SD STROBE transfers two or four bytes of data. The streaming data procedure continues until all data, except byte Bn, is transferred.
- The streaming data procedure is terminated.
- Byte Bn, the last byte is transferred using a basic transfer procedure.
- The session between the controlling master and the slave is terminated.

Streaming Data Cycles

A master can perform streaming data transfers at 10 MHz when -SDR(0,1) is a binary 00, 01, or 10, and can perform the transfer at less than the requested width. -SDR(1) is reserved and slaves should not activate it. The valid streaming data signal combinations from the selected slave are shown in the following table.

-SDR 0 1	-CD DS 16/32	-MSDR	Type of Cycle Requested by Slave
0 0	X X	X	Reserved
0 1	0 0	0	64-bit Streaming Data Transfer
0 1	0 0	1	32-bit Streaming Data Transfer
0 1	0 1	1	16-bit Streaming Data Transfer
1 0	X X	X	Reserved
1 1	X X	X	Basic Transfer

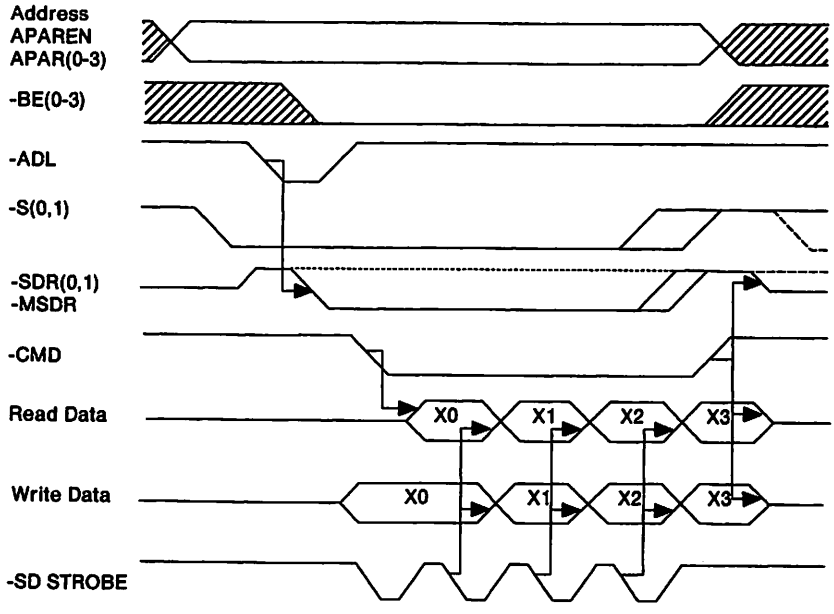
Figure 35. Streaming Data Signal Combinations

Both 16- and 32-bit masters can use streaming data procedures with either 16- or 32-bit slaves. However, because the central translator logic does not perform data steering during a streaming data transfer, a 32-bit slave must perform the steering when streaming with a 16-bit master. (The 32-bit master does the steering for all data transfers with slaves having a smaller data port.)

"Streaming Data Procedure (16- or 32-Bit)" on page 59 and "Streaming Data Procedure (64-Bit)" on page 60 contain the general signal sequence overview.

The following describe how:

- Masters and slaves select streaming data or basic transfer procedures, and the number of bits in each transfer.
- Deferred cycles are performed.
- Data is paced during streaming data procedures.
- Streaming data can be terminated by either the master or the slave.



X0, X1, X2, and X3 represent a data transfer of 16 or 32 bits.

Figure 36. 16- and 32-Bit Streaming Data Cycle Overview

64-Bit Streaming Data Cycles

The 64-bit streaming data transfer can only be used between 64-bit streaming masters and 64-bit streaming slaves. An example of a 64-bit cycle is shown in Figure 37. The 64-bit streaming transfer is similar to the streaming data transfer shown in Figure 36 on page 66, but uses both the data bus and the address bus to achieve the 64-bit data transfer width.

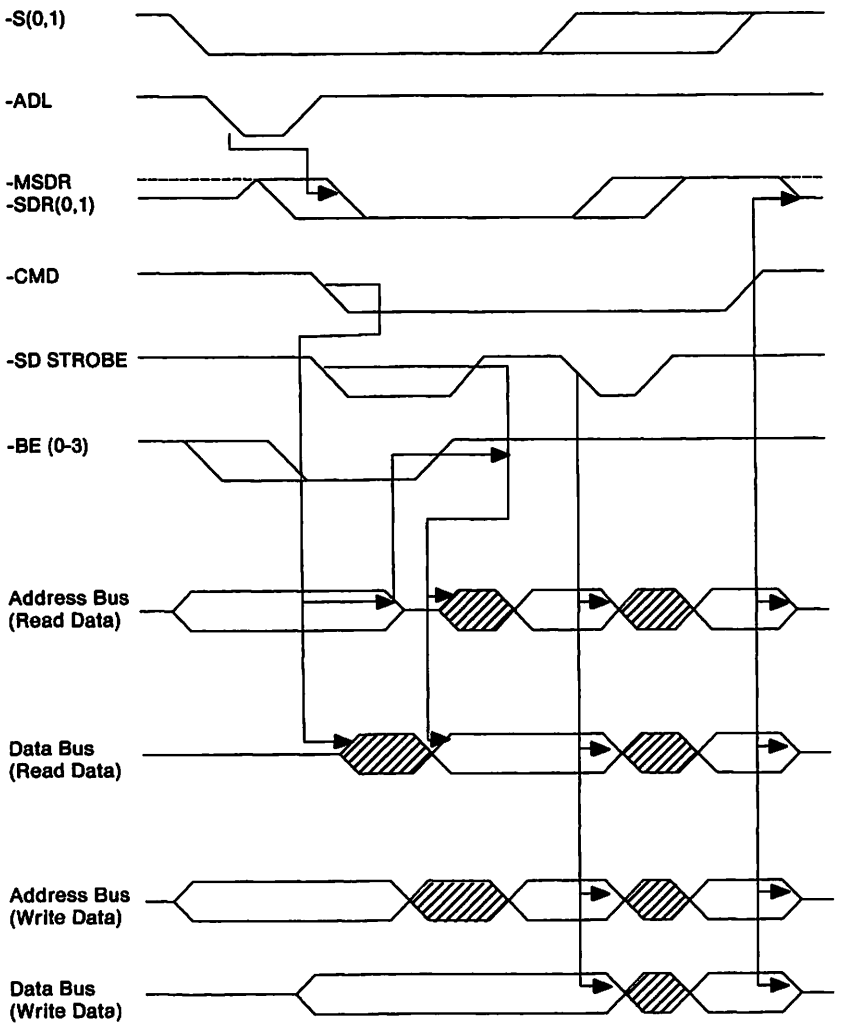


Figure 37. 64-Bit Streaming Data Cycle Overview

The 64-bit cycle begins as a 32-bit basic transfer cycle. The selected slave responds by driving -CD DS 16, -CD DS 32, -SDR(0,1) and -MSDR valid to indicate that it supports 64-bit streaming operations. -BE(0-3) are driven inactive by the master to indicate a 64-bit streaming-data procedure will be used. During a transfer to the slave, the master starts -SD STROBE and gates the data onto the data and address buses. During a transfer to the master, the master tri-states the address bus after driving -CMD active, and, after the trailing edge of -BE(0-3), the slave gates the data onto the data and address buses. The least significant byte is D(0-7) and the most significant byte is A(24-31).

The 64-bit streaming transfer then proceeds like a 32-bit streaming data transfer.

If -DPAREN has been driven active, the parity bits for data bytes zero through three are contained in DPAR(0-3), the parity bits for data bytes four through seven are contained in APAR(0-3).

Deferred start of the 64-bit streaming transfer is supported, but data pacing is not available during 64-bit data streaming transfers.

Starting the Cycle

Streaming masters can drive -SD STROBE active at the same time as -CMD to minimize the overhead in starting the streaming data cycle. The T70 time plus propagation delay for the signals to and from the slave can delay the sample point of -SDR(0,1) and -MSDR until after -CMD and -SD STROBE have been activated. Therefore, if a slave indicates it is capable of performing a streaming data transfer, the master must perform at least one streaming data transfer. Requiring one streaming data transfer prevents the slave from confusing a 32-bit basic transfer with a 64-bit streaming data transfer.

Deferring the Start of the Cycle

During a streaming data transfer, the slave may need more time at the start of the cycle to prepare for the transfer. The slave can defer the start of the transfer by driving CD CHRDY inactive. The controlling master delays the start of -SD STROBE until after CHRDYRTN goes active.

If -SDR(0,1) is not active within the time specified (T70A), the controlling master continues the cycle using basic transfer procedures. All specifications for the basic transfer procedure apply.

If -SDR(0,1) is active, after CHRDYRTN becomes active, the streaming data procedure can be invoked as shown in Figure 38 on page 70.

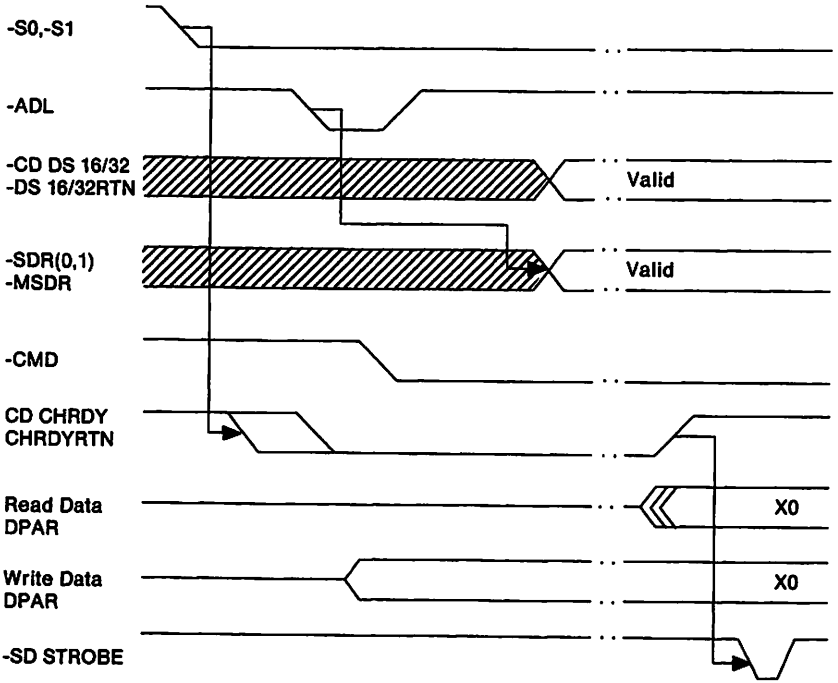


Figure 38. Streaming Data Deferred Start Overview

Pacing the Data

After a 16- or 32-bit streaming data cycle is started, CD CHRDY and CHRDYRTN pace the data transfer, allowing slaves to introduce momentary pauses in the data transfer. CD CHRDY is clocked simultaneously with data and makes state transitions following the high-to-low transition of -SD STROBE On each falling edge, the controlling master must inspect the state of CHRDYRTN. When CHRDYRTN is inactive during a read operation, it indicates that the slave did not have valid data in this clock period and that the transfer must be repeated. When CHRDYRTN is inactive during a write operation, it indicates the slave did not accept the data and that the transfer must be repeated.

Note: Data pacing by the slave is not supported during 64-bit streaming data cycles.

An example of data pacing using CD CHRDY is shown in Figure 39 on page 72. The example shows two idle (CD CHRDY inactive) clock periods, one in the middle of the data transfer, and one at the end. If the slave has driven CD CHRDY inactive, it must not deactivate -SDR(0,1).

When -MSDR is active, the slave cannot pace the transfer. The slave can use CD CHRDY only to delay the initiation of 64-bit transfers. Slaves that cannot meet the timing specifications for the 64-bit transfer must terminate the streaming data cycle. After -SD STROBE is being driven, the master ignores CHRDYRTN.

In this example, as the controlling source of CD CHRDY, the slave internally clocks its bus interface registers with the inversion of -SD STROBE. At the end of the cycle, the bus interface registers are clocked when -CMD goes high. The internal interface clock of the controlling master is conditional on CHRDYRTN being active.

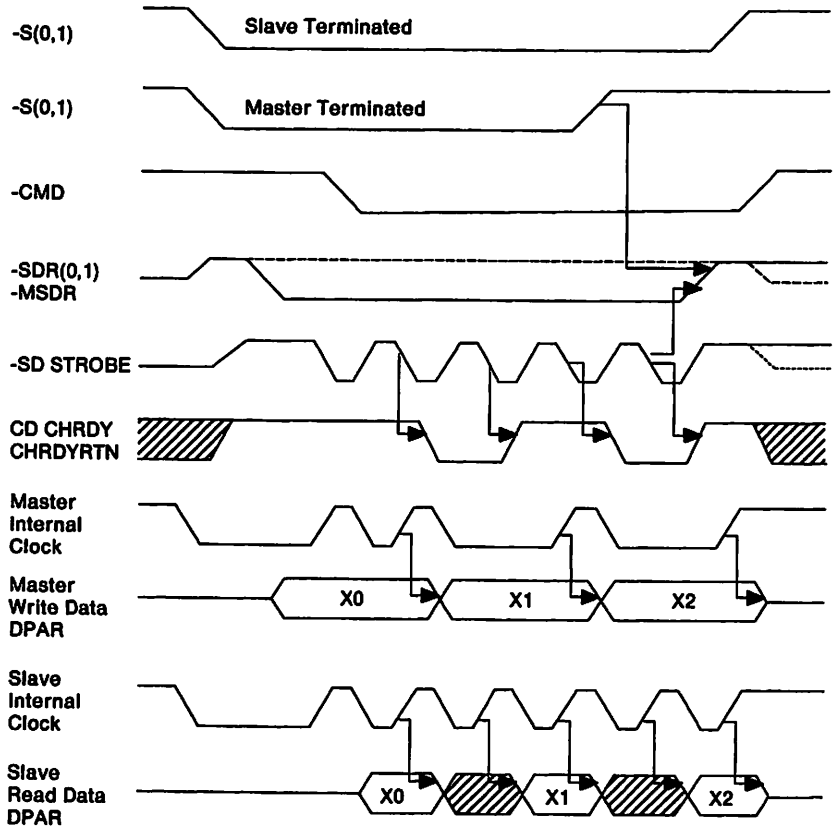


Figure 39. CD CHRDY and Data Pacing

To perform data transfer pacing, streaming masters can introduce momentary pauses in the data transfer by extending -SD STROBE. Data is transferred on the falling edge of -SD STROBE. The strobe period can be extended at either the high or low level, or both. This is done asynchronously to the edges of -SD STROBE.

Whenever the slave is not-ready (CD CHRDY inactive), the master must continue to pulse -SD STROBE. The actual data transfer will occur on the falling edge of -SD STROBE once the slave becomes ready by driving CD CHRDY active.

Terminating the Cycle

Either the controlling master or the slave can terminate the streaming data procedure. If -PREEMPT has become active, the controlling master terminates the streaming data procedure within 7.8 microseconds and releases the channel at EOT. If the fairness feature is active, the preempted master enters the inactive state. See "Fairness Feature and the Inactive State" on page 34 for more information.

Figure 40 shows streaming data cycle termination by the controlling master. Figure 41 on page 74 shows streaming data cycle termination by the controlling master when the slave is not ready. Figure 42 on page 75 shows streaming data cycle termination by a streaming slave.

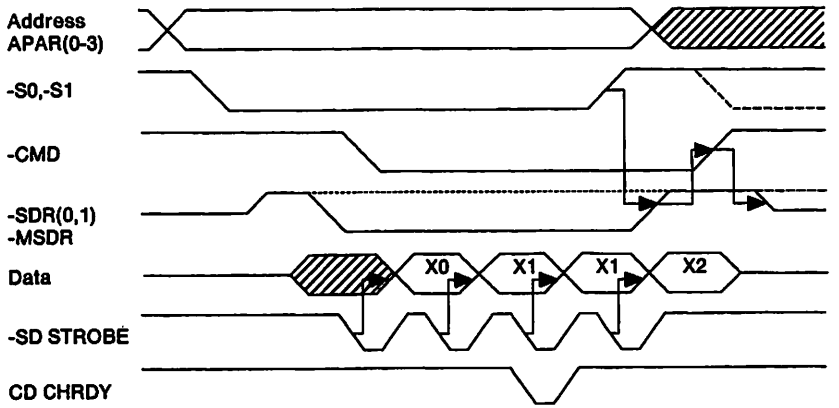


Figure 40. Master-Terminated Streaming Data Cycle

The controlling master can terminate a streaming data cycle by deactivating -S(0,1) on the data transfer before the desired stopping point. This can be done concurrently with the high-to-low transition of -SD STROBE. If -SDR(0,1) is inactive, the controlling master drives -CMD inactive at the point where the next -SD STROBE would have been. The slave **must not** drive -SDR(0,1) inactive while driving CD CHRDY inactive during 16- or 32-bit transfers. If -SDR(0,1) is active and CD CHRDY is inactive, it indicates that the slave is not ready, and the controlling master cannot terminate the cycle.

Figure 41 shows streaming data cycle termination by the master, where status has been deactivated, but the slave is not ready. The master continues to pulse -SD STROBE until it detects the deactivation of -SDR(0,1), which implies that CD CHRDY has been activated. Following the slaves deactivation of -SDR(0,1), the master proceeds to terminate the streaming data cycle. The slave must drive -SDR(0,1), according to T71, on the last -SD STROBE falling edge, similar to a slave-terminated cycle.

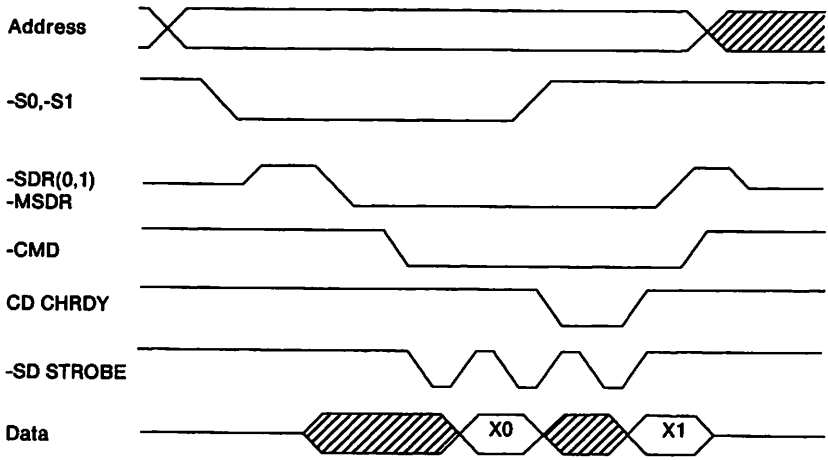


Figure 41. Master-Terminated Streaming Data Cycle (Slave Not Ready)

Figure 42 shows how a streaming data cycle can be terminated by a slave. However, if the master has not transferred all of its data, the master can immediately initiate an additional data transfer procedure.

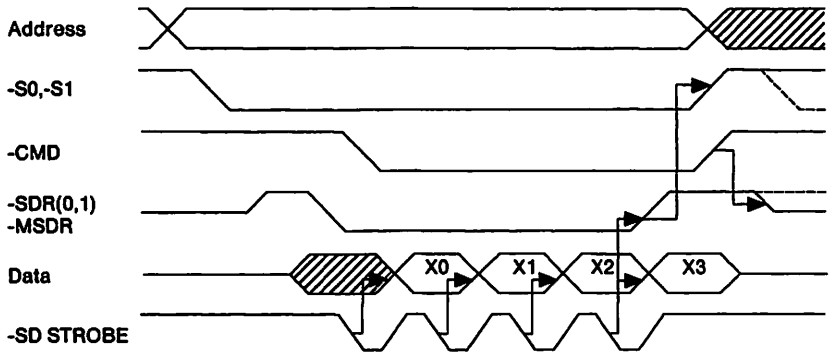


Figure 42. Slave-Terminated Streaming Data Cycle

The slave can terminate a streaming data cycle by driving -SDR(0,1) inactive following the last desired high-to-low transition of the -SD STROBE. The slave must not drive -SDR(0,1) inactive while CD CHRDY is inactive. The controlling master stops -SD STROBE and deactivates -S0, -S1 and -CMD at the point where the next -SD STROBE would have been.

A slave that does not drive CD CHRDY inactive (in response to slave selection) must not drive -SDR(0,1) or -MSDR inactive with the intention of executing a slave-terminated single-cycle operation. This ensures that the master has adequate time to sample the state of -SDR(0,1) before it is deactivated. The slave must be able to execute **two or more** data transfers with the master. The slave can sample TR32 to determine if two or four bytes will be transferred with each -SD STROBE. A master can perform a single streaming-data cycle operation. A master that performs a single streaming-data cycle must make sure that it meets the basic timing requirements (T10) as well as all master-terminated streaming-data requirements, including T71A.

When -CMD is driven inactive, the slave tri-states -SDR(0,1), -MSDR and, if performing a read operation, the data bus. The termination sequence results in restoring the streaming data signals to their inactive state prior to their being tri-stated, eliminating uncertainties about the state of these signals in succeeding channel cycles.

Streaming Data Timing

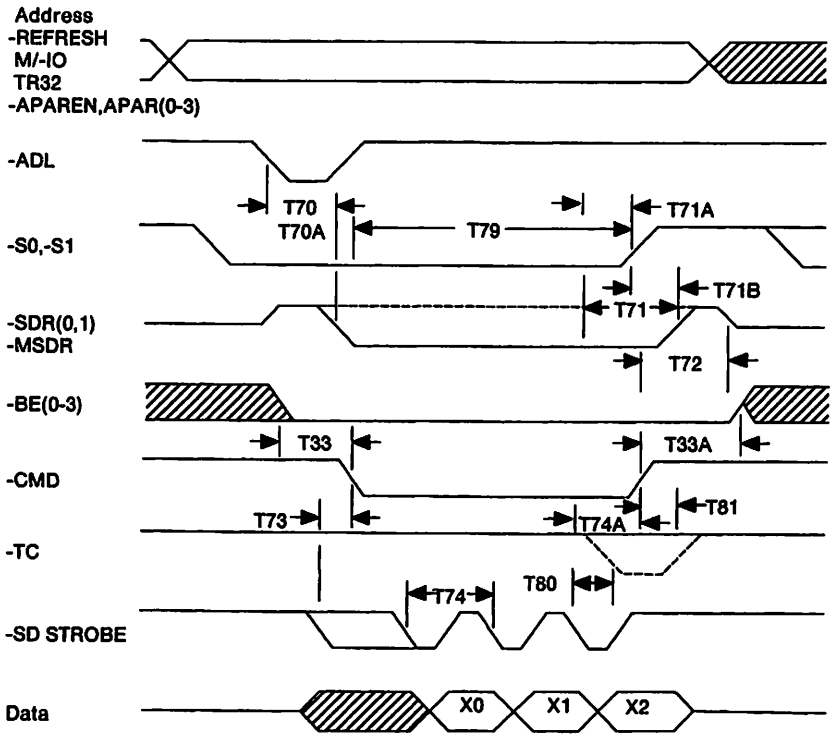


Figure 43. Streaming Data Cycle

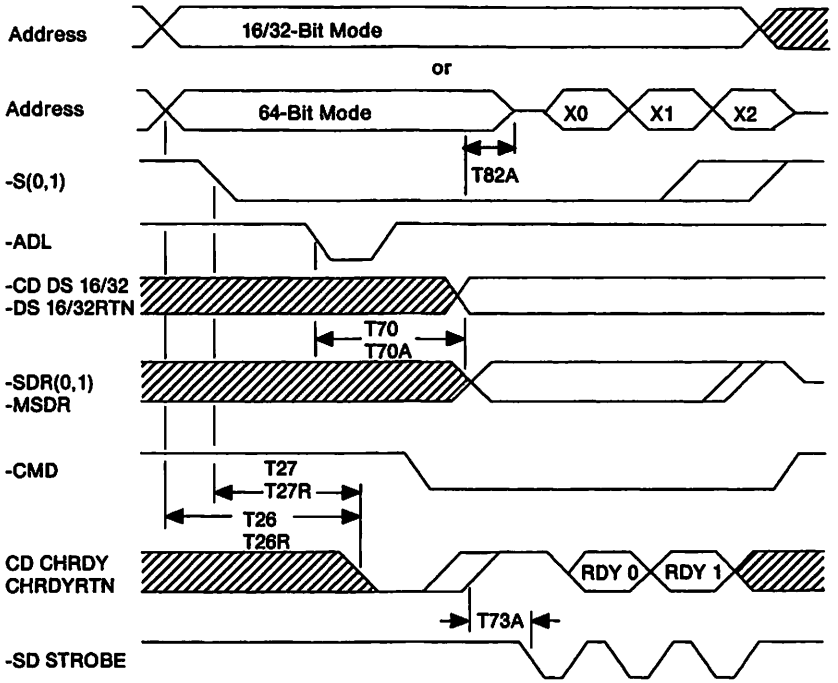


Figure 44. Streaming Data Cycle - Deferred Start

Note: CD CHRDY is not supported during 64-bit streaming data transfers after the address has been removed.

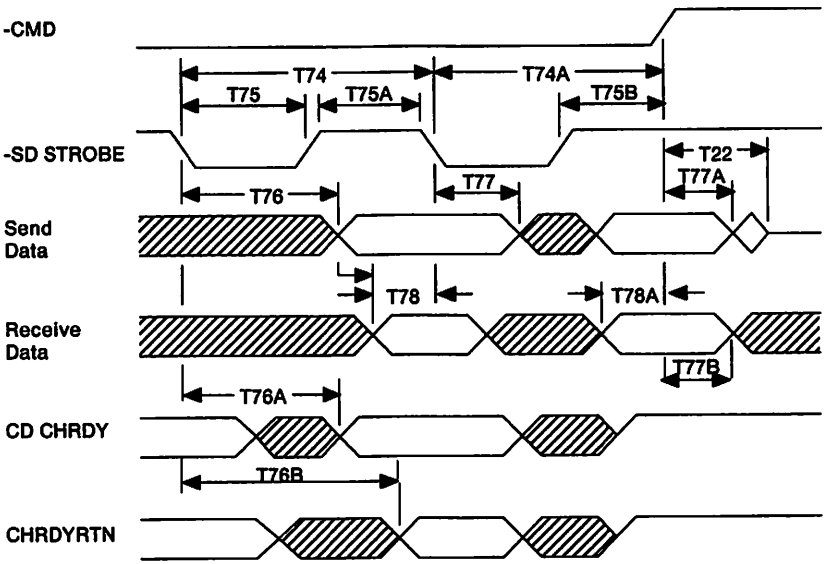
	Timing Parameter	Min/Max	Note
T22	Data bus tri-state from -CMD inactive	0 / 40 ns	
T33	-BE(0-3) active to -CMD active	10 / - ns	
T33A	-BE(0-3) hold from -CMD inactive	10 / - ns	
T70	-SDR(0,1) and -MSDR valid from -ADL active (slave only)	0 / 40 ns	7
T70A	-SDR(0,1) and -MSDR valid from -ADL active (masters only, for deferred cycle use)	0 /115 ns	
T71	-SDR(0,1) and -MSDR inactive from last -SD STROBE fall (slave terminated cycle)	0 / 40 ns	
T71A	-S0, -S1 inactive from last -SD STROBE fall (master terminated cycle)	- / 10 ns	4,1
T71B	-SDR(0,1) and -MSDR inactive from -S0, -S1 inactive (master terminated; slave ready)	0 / 40 ns	
T72	-SDR(0,1) and -MSDR tri-state from -CMD inactive (master-terminated cycle)	0 / 40 ns	
T73	-SD STROBE active to -CMD active	- / 10 ns	3
T73A	-SD STROBE active from CHRDYRTN active	0 / - ns	
T74	-SD STROBE period	100/ - ns	
T74A	-CMD inactive from last -SD STROBE fall	100 / - ns	
T79	Status inactive from -CMD active (master-terminated cycle)	- / 7.8 μs	6
T80	-TC active from last -SD STROBE fall	- / 60 ns	1
T81	-TC hold from -CMD inactive	10 / - ns	
T82A	A(0-31) tri-stated from CD CHRDY active	0 / - ns	5

Figure 45. Streaming Data Transfer Timing Specifications

Notes:

1. Streaming masters can increase T71A and T80 by the amount that the -SD STROBE clock period exceeds T74 minimum.
2. For streaming data cycles that default to the basic transfer cycles, the rules for deactivation of -S0, -S1 apply, as specified by the basic transfer procedure.
3. -SD STROBE can be driven active concurrently or after -CMD active, and can become active just before -CMD active due to driver skew and loading.
4. -S0, -S1 are deactivated concurrently with the negative transition of -SD STROBE. Normal skew from circuits and net loading can result in -S0, -S1 being deactivated prior to the negative transition of -SD STROBE in order to meet the T71A timing requirement.

5. This timing applies only to the 64-bit streaming data transfer.
6. The master must signal termination of the streaming data transfer within 7.8 microseconds from -CMD active. However, if -PREEMPT goes active, the master can continue to hold the channel for 7.8 microseconds from -PREEMPT.
7. A valid address decode must be completed before driving these signals.



See notes for definition of Send and Receive Data

Figure 46. Streaming Data Clocking (16- and 32-Bit)

	Timing Parameter	Min/Max	Note
T74	-SD STROBE period	100 / - ns	
T74A	-CMD inactive from last -SD STROBE fall	100 / - ns	
T75	-SD STROBE active	35 / - ns	
T75A	-SD STROBE inactive	35 / - ns	
T75B	-SD STROBE inactive to -CMD inactive	35 / - ns	
T76	Send Data valid from -SD STROBE fall	- / 60 ns	1
T76A	CD CHRDY valid from -SD STROBE fall	3 / 25 ns	
T76B	CHRDYRTN valid from -SD STROBE fall	3 / 45 ns	
T77	Send Data hold from -SD STROBE fall	10 / - ns	
T77A	Write data hold from -CMD inactive	11 / - ns	
T77B	Read data hold from -CMD inactive	7 / - ns	
T78	Receive Data valid before -SD STROBE fall	25 / - ns	
T78A	Receive Data valid before -CMD inactive	25 / - ns	

Figure 47. Streaming Data Transfer Timing Specifications (16- and 32-Bit)

Notes:

1. Streaming masters can increase T76 by the amount that the -SD STROBE clock period exceeds T74 minimum.
2. To perform data transfer pacing, streaming masters can introduce momentary pauses in the data transfer by extending the active or inactive levels of the streaming data strobe.
3. Send Data refers to write data at the connector for the master and read data at the connector for the slave. Receive Data refers to read data at the connector for the master and write data at the connector for the slave.
4. The value of (T74 minus T76 minus T78) represents system timing margins to accommodate bus skew, propagation delays, and other signal transmission delays.

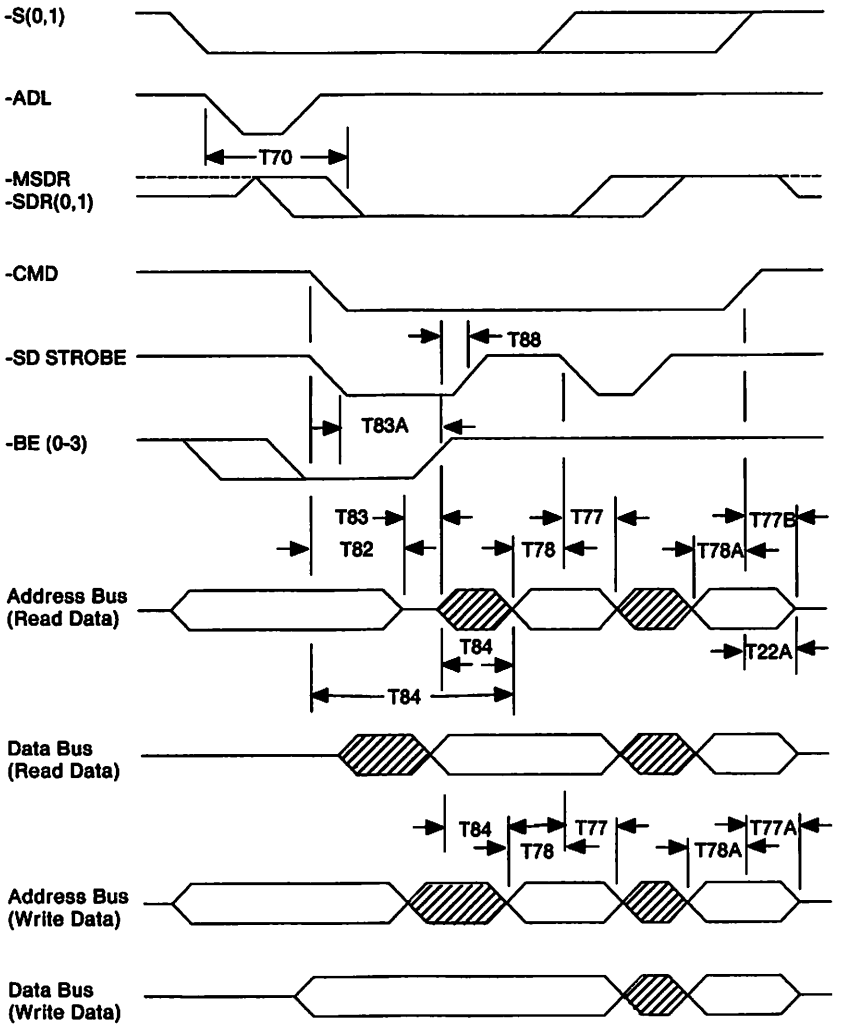


Figure 48. Streaming Data Clocking (64-Bit)

	Timing Parameter	Min/Max	Note
T22A	A(0-31) as D(32-63) tri-state from -CMD inactive	0 / 40 ns	2
T82	A(0-31) tri-stated from -CMD active	30 / - ns	1
T83	-BE(0-3) inactive from A(0-31) tri-stated	0 / - ns	1
T83A	-BE(0-3) inactive from -SD STROBE active	20 / - ns	1
T84	A(0-31) valid as D(32-63) from -BE(0-3) inactive and -SD STROBE active	0 / 60 ns	4
T88	-SD STROBE inactive from -BE(0-3) inactive	40 / - ns	5

Figure 49. Streaming Data Transfer Timing Specifications (64-Bit)

Notes:

1. This timing applies only to a master during a read operation.
2. This timing applies only to a slave during a read operation.
3. During 64-bit streaming data transfers the master and slave meet the setup and hold times shown in Figure 48 on page 81, for the data being transferred.
4. Streaming masters can increase T84 by the amount that the -SD STROBE clock period exceeds T74 minimum.
5. This allows the slave to steer the address path outward when the master signals a streaming read and to steer it inward when -SD STROBE is deactivated with -BE(0-3) still active.

Direct Memory Access Procedure

A direct memory access (DMA) operation is a specialized sequence of basic transfer procedure. It consists of a single transfer or a burst transfer between a DMA slave and a memory slave. The following lists only those definitions, signal sequences, and timing specifications that are different from or additional to those of a basic transfer cycle.

Read and Write Operations

During a DMA write operation, the DMA slave provides the data and the memory slave stores it. During a DMA read operation, the memory slave provides the data and the DMA slave stores it. If the slave is unable to provide or store data within the time specified for a normal operation, it extends the cycle by driving CD CHRDY inactive. To ensure the read data is valid, the slaves storing the data sample the data lines with reference to the trailing edge of the -CMD.

DMA Slave Selection

A DMA slave is selected by either its arbitration level or optionally by its I/O address, but not by both. The default selection is by decode of the arbitration level, status (-S0 exclusive-OR -S1), and an I/O cycle (M/-IO in the -IO state). Selection by I/O address is by decode of the I/O address (A0-A15), status (-S0, -S1), and an I/O cycle (M/-IO in the -IO state). When selection by I/O address is implemented for a DMA slave, a bit located in either configuration-data space or I/O address space is used to control the method of selection. All DMA slaves drive -CD SFDBK during the I/O portion of the DMA cycle for both selection methods.

Address During DMA Operation

During a memory cycle, the DMA controller drives the memory address on the address bus. During an I/O cycle, the DMA controller must drive a known I/O address on the bus. The I/O address must be that of the DMA slave, or hex FFFC or 0000.

Transfer Data Length

Normally, the DMA transfer length is an integral number of transfers of a width determined by the DMA slave's data port width. The data port address for a 16-bit DMA slave must be on an even address boundary; the address for a 32-bit DMA slave must be on a four-byte address boundary. The burden of supporting odd-length transfers is the responsibility of the DMA slave.

DMA Procedure

The two types of DMA procedures are:

- Single Transfer
- Burst Transfer.

Single Transfer

During a data transfer, the requesting DMA slave initiates the DMA operation by having its arbiter drive -PREEMPT active and arbitrate for the channel. On the low-to-high transition edge of ARB/-GNT (start of arbitration cycle), the DMA slave's priority level is placed on the arbitration bus. The DMA slave competes for use of the channel as defined in the arbitration procedure.

On the high-to-low transition edge of ARB/-GNT the DMA controller monitors the arbitration bus to determine if its DMA slave gained control of the channel. A match initiates a serial DMA operation in which the DMA controller drives the address lines during both cycles, as well as driving -S0, -S1, -CMD, and other control signals. The DMA controller also monitors -BURST, finds it inactive, and then activates -BURST (either on the channel, or through the system logic) to notify the central arbitration control point that consecutive cycles are required.

The DMA controller also manages the storage of data during the DMA read or DMA write operations for the consecutive cycles required by serial DMA transfers. Data count for data gated on or off the bus by the DMA slave is also maintained by the DMA controller.

Notes:

1. The DMA controller might be required to perform multiple cycles for the data transfer to the memory slave, if the data port width of the memory slave is less than that of the DMA slave.
2. DMA slaves using the single-transfer-type DMA procedure *will not* receive more data than is required to support their data port width (for example, 2 bytes to a 16-bit port).
3. Consecutive transfers can be made with the DMA slave (as opposed to the serial sequences shown in the remainder of this section). The memory read or write cycles might not appear on the channel.

Burst Transfer (Multiple Transfer Cycles)

The method for transferring a block of data is similar to a single transfer DMA cycle, except that the local arbiter for the requesting DMA slave activates -BURST to request continued channel ownership for a burst operation. The central arbitration control point will not drive ARB/-GNT to the ARB state while -BURST is active. As in single-transfer DMA operations, the DMA controller drives the address, -S0, -S1, -CMD, and data transfer signals, as well as manages addresses and data transfer counts. The DMA controller also drives -BURST (either on the channel or through the system logic) jointly with the DMA slave for DMA write cycles. It does not drive -BURST active for aurst DMA read cycles.

DMA Terminating Conditions

The DMA transfer operation is terminated by the EOT. At this time, a DMA slave can again request DMA service by arbitrating for use of the channel.

The terminating conditions for DMA transfers are:

- During a single transfer, the DMA controller can terminate the DMA transfer by releasing -BURST.
- During burst transfers, either the DMA slave or the DMA controller can terminate the DMA operation.
  - The DMA slave terminates the transfer by driving -BURST inactive. The DMA slave will terminate the transfer when it reaches its buffer limit, or after -PREEMPT is driven active by another arbitrating participant. Arbitrating slaves should maximize use of the channel within the 7.8 microsecond time limit.
  - The DMA controller terminates the transfer by driving -TC active when the transfer count goes to zero. When the DMA slave receives -TC, it drives -BURST inactive.

Use of -BURST

When a bursting DMA slave wins the arbitration for the channel, it drives -BURST active. During default cycles, the DMA slave drives -BURST inactive when it detects status active for the last I/O cycle of this burst transfer. However, on sensing -SBHE and/or -BE(0-3), the DMA slave might discover that it has one more cycle to perform. If so, it redrives -BURST active to continue the transfer. See "Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns)" on page 92.

During extended cycles, the DMA slave drives -BURST inactive within the specified time after the leading edge of -CMD.

If a burst transfer is terminated by the DMA controller, the DMA slave drives -BURST inactive within the specified time after the leading edge of -TC.

For DMA write cycles using burst transfers, the DMA controller drives -BURST active at the leading edge of -CMD for each I/O read and drives it inactive when it detects status active for the next memory write.

DMA Timings

The following are the timing parameters for the DMA procedure.

Single DMA Transfer (DMA-Controller Controlled)

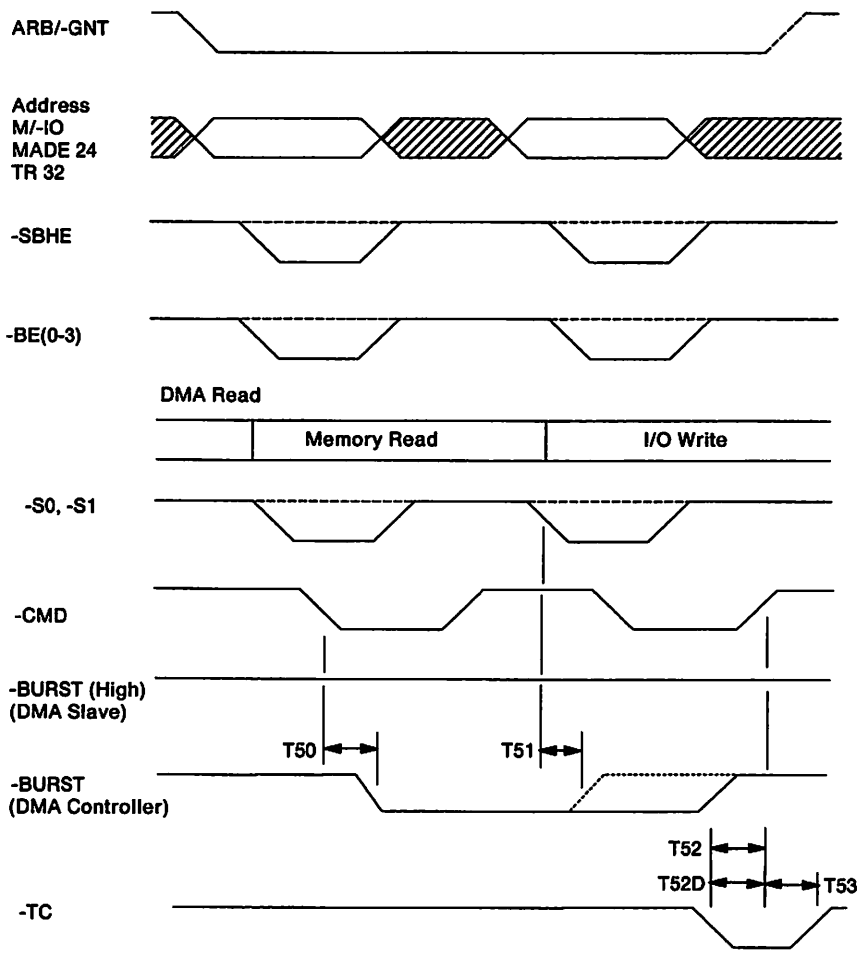


Figure 50 (Part 1 of 2). Single DMA Transfer (DMA-Controller Controlled)

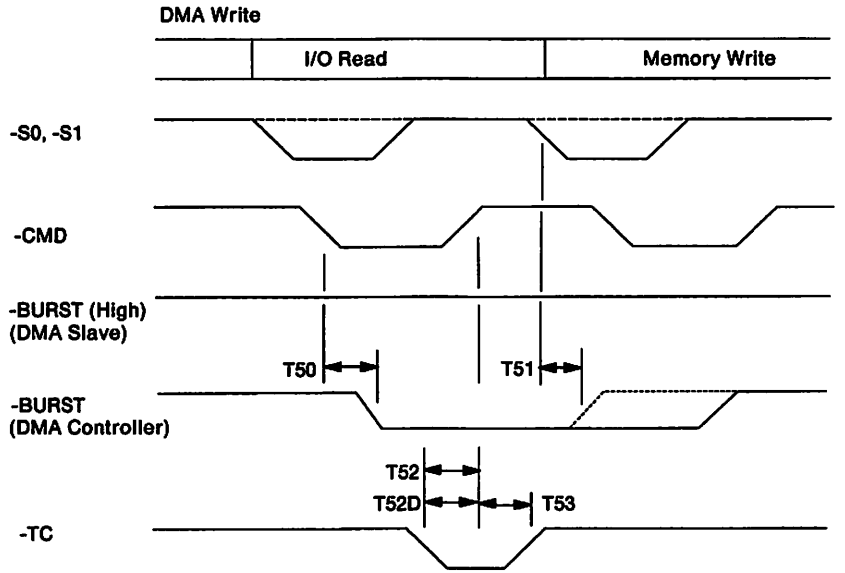


Figure 50 (Part 2 of 2). Single DMA Transfer (DMA-Controller Controlled)

	Timing Parameter	Min/Max	Note
T50	-BURST activated by the DMA controller from the first -CMD active	- / 30 ns	
T51	-BURST released by the DMA controller from the last status active	- / 40 ns	
T52	-TC setup to -CMD inactive	30 / - ns	2
T52D	-TC setup to -CMD inactive	15 / - ns	2
T53	-TC hold from -CMD inactive	10 / - ns	3

Figure 51. Single DMA Transfer (DMA-Controller Controlled) Timing Specifications

Notes:

1. Only the timing parameters that are additional to those specified for the basic transfer cycle are included here.
2. T52D applies to participants operating at 200-nanosecond cycles *only*. Devices operating at cycle rates slower than 200 nanoseconds *must* meet T52.
3. For streaming data procedures, T80 and T81 also apply. (See "Streaming Data Timing" on page 76).

Burst DMA Transfer (DMA-Controller Terminated)

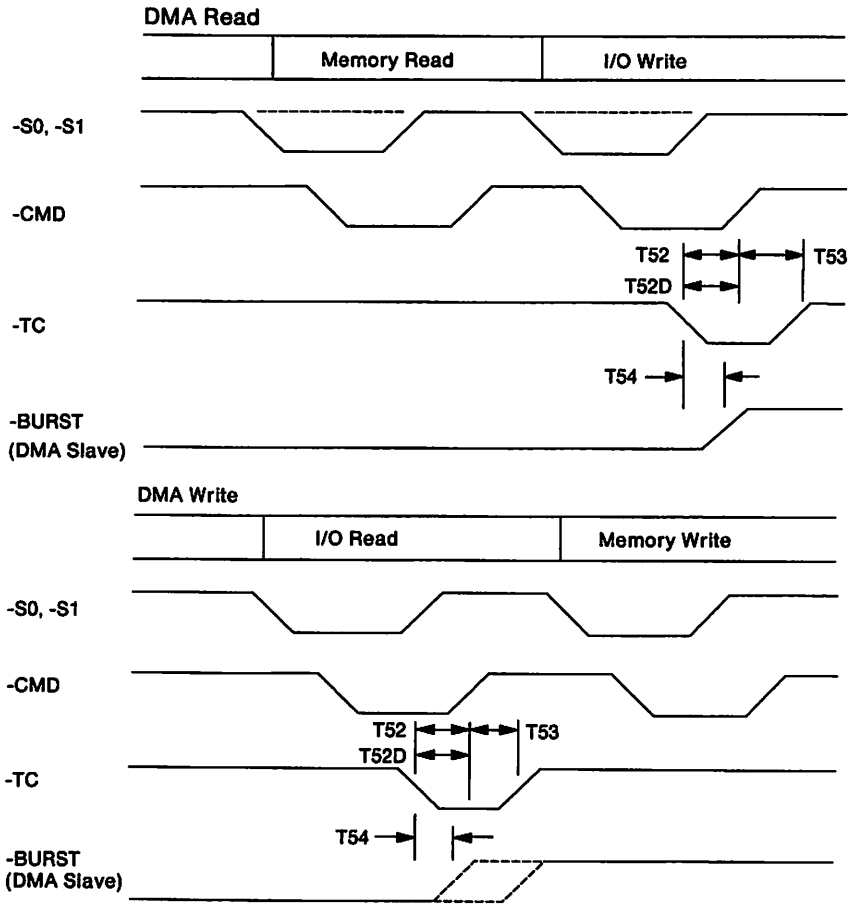


Figure 52. Burst DMA Transfer (DMA-Controller Terminated)

	Timing Parameter	Min/Max	Note
T52	-TC setup to -CMD inactive	30 / - ns	1
T52D	-TC setup to -CMD inactive	15 / - ns	1
T53	-TC hold from -CMD inactive	10 / - ns	3
T54	-BURST released by the DMA slave from -TC active	- / 30 ns	

Figure 53. Burst DMA Transfer (DMA-Controller Terminated) Timing Specifications

Notes:

1. T52D applies to devices operating at 200-nanosecond cycles *only*. Devices operating at cycle rates slower than 200 nanoseconds *must* meet T52.
2. During a DMA read, the DMA controller must not initiate another cycle after issuing -TC, even if the DMA slave is not fast enough to drive -BURST inactive before the end of -CMD.
3. For the streaming data procedure, T80 and T81 also apply. (See "Streaming Data Timing" on page 76).

Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns)

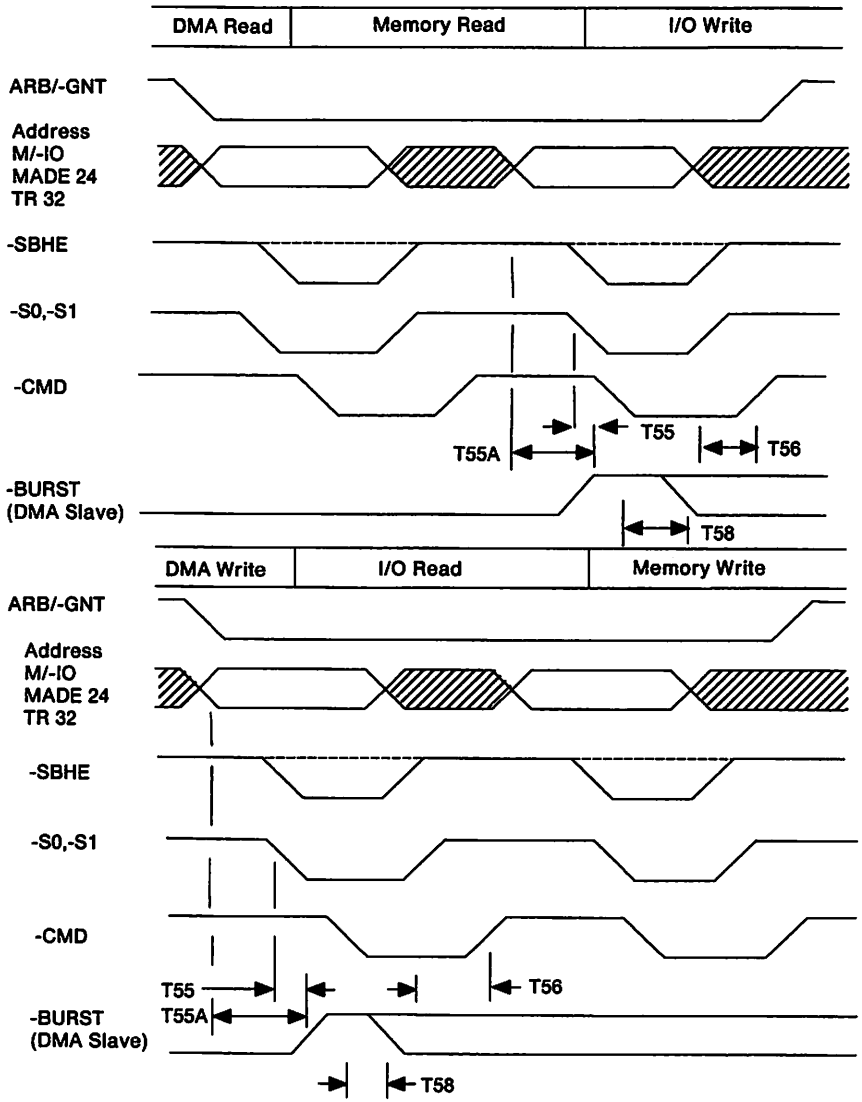


Figure 54. Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns)

	Timing Parameter	Min/Max	Note
T55	-BURST released by the DMA slave from the last I/O cycle status active (default cycle only)	0 / 40 ns	1
T55A	-BURST released by the DMA slave from I/O address valid for the last I/O cycle (default cycle only)	- / 70 ns	1
T56	-BURST inactive to -CMD inactive	35/ - ns	2
T58	-BURST redrive from -CMD active	- / 30 ns	1

Figure 55. Burst DMA Transfer (DMA-Slave Terminated - Default Cycle, 200 ns) Timing Specifications

Notes:

1. After releasing -BURST and on receiving -SBHE and/or -BE(0-3), if the DMA slave determines that it requires another cycle, it can redrive -BURST active.
2. The setup time for -BURST inactive to the end of -CMD (T56) must be guaranteed during the last I/O write cycle to prevent the DMA controller from starting the next cycle. This setup time (T56) is guaranteed by the sum of -BURST release time (T55/T55A controlled by the DMA slave) and the -BURST RC restoration time. The RC restoration time must not exceed 70 nanoseconds. T56 is the same for default and extended cycles.

Burst DMA Transfer (DMA-Slave Terminated - Synchronous-Extended Cycle, 300 ns)

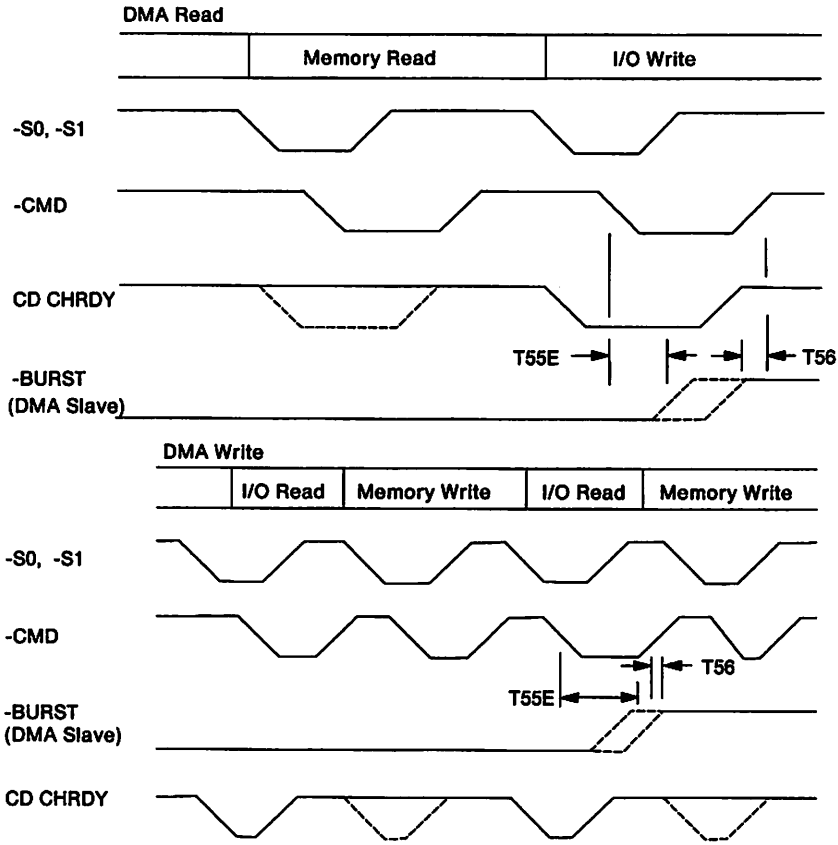


Figure 56. Burst DMA Transfer (DMA-Slave Terminated - Synchronous-Extended Cycle)

	Timing Parameter	Min/Max	Note
T55E	-BURST released by the DMA slave from the last I/O -CMD active (extended cycles only)	- / 80 ns	2
T56	-BURST inactive to -CMD inactive	35 / - ns	1

Figure 57. Burst DMA Transfer (DMA-Slave Terminated - Synchronous-Extended Cycle, 300 ns) Timing Specifications

Notes:

1. The setup time for -BURST inactive to the end of -CMD (T56) must be guaranteed during the last I/O write cycle to prevent the DMA controller from starting the next cycle. This setup time (T56) is guaranteed by the sum of -BURST release time (T55E controlled by the DMA slave) and the -BURST RC restoration time. The RC restoration time must not exceed 70 nanoseconds. T56 is the same for default and extended cycles.
2. -BURST cannot be redriven.

Burst DMA Transfer (DMA Slave Terminated - Asynchronous-Extended Cycle, ≥300 ns)

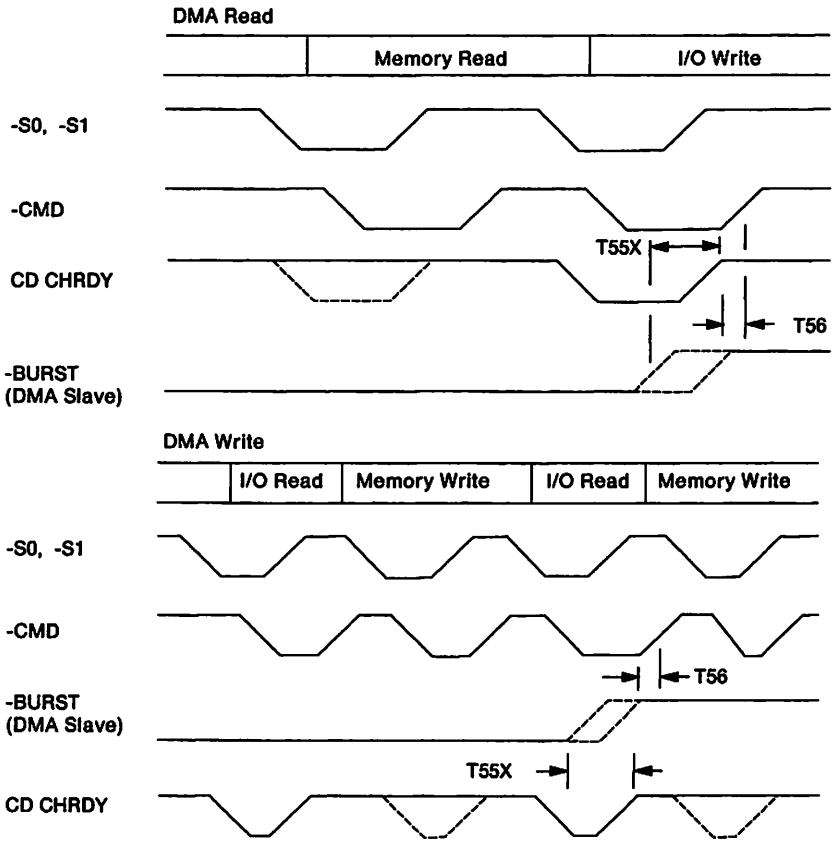


Figure 58. Burst DMA Transfer (DMA-Slave Terminated - Asynchronous-Extended Cycle, ≥300 ns)

	Timing Parameter	Min/Max	Note
T55X	-BURST released by the DMA slave before CD CHRDY active (asynchronous-extended cycles only)	50 / - ns	2
T56	-BURST inactive to -CMD inactive	35 / - ns	1

Figure 59. Burst DMA Transfer (DMA-Slave Terminated - Asynchronous-Extended Cycle, ≥300 ns) Timing Specifications

Notes:

1. The setup time for -BURST inactive to the end of -CMD (T56) must be guaranteed during the last I/O write cycle to prevent the DMA controller from starting the next cycle. This setup time (T56) is guaranteed by the sum of -BURST release time (T55X controlled by the DMA slave) and the -BURST RC restoration time. The RC restoration time must not exceed 70 nanoseconds. T56 is the same for default and extended cycles.
2. -BURST cannot be redriven.

Interrupt Procedures

This section describes how all devices use interrupts. The procedure for sharing interrupts uses a request for interrupt services that is detected by the level of the interrupt request signal (level sensitive). These procedures involve the interaction between the hardware and an interrupt service routine.

To initiate an interrupt request, a device drives its -IRQ(N) active (N represents the assigned interrupt level for the device) and holds the signal active until it is reset by the interrupt service routine.

Each device provides an interrupt-pending bit within its address space. This bit is set by the device when it has an interrupt request pending and is reset by the interrupt handler when the interrupt request is serviced.

Note: The drivers for the interrupt signal must be disabled when the device is disabled.

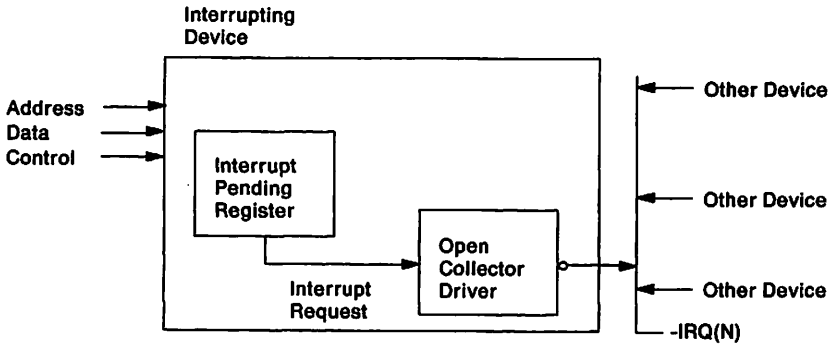


Figure 60. Typical Adapter Interrupt Sharing Implementation

An example of the sequence of the hardware and interrupt service routine interaction is shown in the following.

Hardware Operation

1. An interrupt condition causes the hardware to drive -IRQ(N) active and sets an interrupt-pending latch, which can be read by software.

Software Operation

2. A master begins executing code at the beginning of the appropriate chain of interrupt handlers.
3. The interrupt handler reads the interrupt-pending latch of the first device in the chain. If the latch is not set, the next device in the chain is tested. When the reporting device is detected, the handler executes the service routine.
4. The interrupt service routine operates the device hardware.

Hardware Operation

5. The device hardware resets the interrupt-pending latch and the interrupt request because of interrupt-service-routine actions.

Software Operation

6. The interrupt service routine completes the interrupt processing.

Hardware Operation

7. If another interrupt is pending (-IRQ(N) driven active by another device), the sequence starts again at 1.

Refresh

The timing of the refresh operation is performed by system logic. -REFRESH is driven active during a memory-read operation to indicate that a refresh cycle is in progress. The refresh cycle is a basic transfer default cycle with -REFRESH active. The timing for -REFRESH is the same as that of the address bus.

Note: Adapters with registers that have destructive read-outs and are mapped into the memory address space should include -REFRESH in their address decode.

During a refresh operation, the nine low-order address bits are incremented for each refresh cycle to address the next byte to be refreshed. (A0 changes after each refresh cycle; A8 changes after 256 cycles.)

The address bits, A31 through A9, might not change, but these bits will be stable during the refresh cycle. The signals -BE(0-3) and -SBHE are irrelevant (but stable) during the refresh cycle.

The refresh operation can occur in periodic intervals or in short bursts of multiple operations. The refresh rate is a minimum of 128 refresh cycles every 2 milliseconds.

Note: Time periods between refresh cycles can vary and should not be used as a timing mechanism.

Regardless of whether it decodes -REFRESH or not, a memory slave can perform a memory read operation during a refresh cycle and drive its data onto the data bus, provided its memory address is on the address bus.

Memory slaves that do not need to perform refresh operations do not need to receive this signal.

System Configuration

The adapter configuration procedure consists of:

- Adapter ID and adapter configuration data
- Programmable option select (POS) registers
- Nonvolatile memory (for system configuration data)
- Signal timing specifications.

The system master configures a system using setup cycles. The system configuration procedure supports the identification of the adapters that reside within a system (by reading the adapter IDs) and the configuration of those adapters (by writing configuration data). Configuration data consists of the following programmable fields:

- Required Fields
  - Adapter ID - the two byte adapter ID
  - Card Enable - a bit that enables or disables the output drivers on the adapter
  - Channel Check - a bit that indicates if the adapter has driven -CHCK active
- Conditionally Required Fields
  - Adapter I/O Address Select - if two or more of the same type of adapter can be installed in a system
  - Adapter ROM/RAM Address Select - if the adapter uses part of the memory address space at hex 000C0000 to 000DFFFF
  - Arbitration level - if the adapter is an arbitrating participant
  - Auto-Incrementing Enable - if the adapter uses POS Registers 6 and 7 as an automatically-incremented register
  - Channel Check Exception Reporting - if the adapter supports synchronous -CHCK for channel non-parity or slave-dependent exceptions
  - Channel Check Status Indicator - if the adapter supports -CHCK
  - Data and Address Parity Enable - if the adapter supports data or address parity
  - Device I/O Address Assignment - if the total I/O-address-space required for a type of adapter is greater than 32 bytes

- Matched Memory Enable - if the adapter supports the matched-memory extension
- Selected Feedback Return Exception Enable - if the bus master receives -SFDBKRTN
- Streaming Data Enable - if the adapter supports streaming data procedures

- Optional Fields
  - Option Select Data - customized, adapter-unique configuration data
  - Fairness Enable - optional if the adapter drives -BURST
  - POS Subaddress Extension
  - Status Information - contains the channel-check status or a pointer to the channel-check status.

See the specific-system technical manuals for system configuration information.

Multiple Card Adapters

Multiple-card adapters are adapters that consist of a primary card and one or more secondary cards. The secondary cards are installed in the channel connectors adjacent to the primary card and can be on either, or both, sides of the primary card. The number of secondary cards is limited only by the number of connectors available.

The multiple-card adapter provides only one set of POS registers. Only the primary card responds to -CD SETUP.

The multiple-card adapter operates as a single adapter, and is completely enabled or disabled by the single set of POS registers.

Adapter Configuration Data

After initialization, the system must be able to access the configuration data written into adapter POS registers.

Adapters must always respond to setup cycles. After initialization, the ability to change the configuration data in the POS registers for an adapter is adapter dependent.

Figure 61 shows both the address and field definition. When its -CD SETUP is active, the adapter must provide access to its configuration data. This data is contained in its POS registers and is divided into multiple device-unique, read/write fields.

The following table shows the organization of the I/O address space used for POS operations. Setup functions respond to these addresses only when -CD SETUP is active.

Address (Hex)	Function
XXX0	POS Register 0 - Adapter Identification Byte (Low Byte)
XXX1	POS Register 1 - Adapter Identification Byte (High Byte)
XXX2	POS Register 2 - Option Select Data Byte 1 Bit 0 is designated as Card Enable.
XXX3	POS Register 3 - Option Select Data Byte 2
XXX4	POS Register 4 - Option Select Data Byte 3
XXX5	POS Register 5 - Option Select Data Byte 4 Bit 7 is designated as channel check. Bit 6 is designated as channel-check-status indicator.
XXX6	POS Register 6 - Subaddress Extension (Low Byte)
XXX7	POS Register 7 - Subaddress Extension (High Byte)

Figure 61. POS I/O Address Space

Bits 6 and 7 of POS Register 5 and bit 0 of POS Register 2 are defined by the architecture; all other bits within POS Registers 2 through 5 are available for adapter use.

Required Configuration Fields

The following configuration fields are required for all adapters.

Note: When CHRESET is driven active, adapters must set all fields to the default state.

Adapter ID Field

The Adapter ID field is a two-byte, read-only field contained in POS Register 0 and 1. Each adapter is assigned a unique two-byte adapter ID that is used for adapter identification. An adapter ID of hex 0000 indicates that the adapter is not ready; if an adapter is not present, the adapter ID will be read as hex FFFF.

Card Enable Field

The card enable field is a one-bit, read/write field; it is bit 0 in POS Register 2. When this field is set to 0, the adapter disables the Micro Channel interface except when responding to its setup cycles. When this field is set to 1, the adapter enables all of its Micro Channel interface. The default state of this field is 0.

Channel Check Field

The Channel Check field is a one-bit, read/write field that indicates the adapter has driven -CHCK active; it is bit 7 of POS Register 5. The adapter sets this field to 0 to indicate it has an exception condition, and the system master resets this field to 1 to reset the adapter after handling the error condition. If the adapter has driven -CHCK active, it must set this field to 0 except for address parity exceptions. If the system master resets this field to 1 and the adapter is still driving -CHCK active, the adapter must drive -CHCK inactive. If the adapter does not support driving the -CHCK signal active (does not support channel check) this field is always set to 1.

Note: If this field is set to 0 during a setup cycle, the adapter should not drive -CHCK active.

Conditionally-Required Configuration Fields

The following are the configuration fields that are required when the adapter supports a specified function.

The Channel Check Status Indicator field has a specific location and state because the system master must handle the channel-check condition. Other conditionally-required fields do not have a specific location within POS address space or a specific state.

Note: When CHRESET is driven active, adapters must set all fields to the default state.

Adapter I/O Address Select Field

The Adapter I/O Address Select field is a one- to three-bit field that selects the adapter address decode from the possible address ranges. These bits allow I/O address resolution for up to eight devices of one type in a system. However, if the total I/O address space needed to support the maximum number of devices exceeds 32 bytes, use the Device I/O Address Assignment field (see page 106).

Adapter ROM/RAM Address Select Field

The Adapter ROM/RAM Address Select field is a four-bit, read/write field that determines the starting address of the adapter ROM/RAM address space for the adapter. The adapter address space is defined as the memory address space from hex 000C0000 to 000DFFFF. The four bits represent address bits A16 through A13, which corresponds to 16 consecutive 8KB blocks.

Note: Adapters that require more than 16KB in this memory space can cause addressing conflicts that cannot be resolved during system configuration.

Arbitration Level Field

The Arbitration Level field is a four-bit, read/write field that selects the arbitration level. All adapters that use the arbitration procedure (bursting and nonbursting) must support a selectable arbitration level.

Auto-Incrementing Enable Field

The Auto Incrementing Enable field is a one-bit, read/write field that enables or disables the automatic incrementing of POS Registers 6 and 7. The default state for this field is disable, which causes the adapter to not perform auto-incrementing. When enabled, it causes POS Registers 6 and 7 to automatically increment for each access to POS Register 3 or 4 (see "POS Subaddress Extension Field" on page 108).

Channel Check Exception Reporting Field

The Channel Check Exception Reporting field is a one-bit, read/write field that selects either synchronous or asynchronous -CHCK exception reporting for channel non-parity or slave-dependent exception conditions. When enabled, these exceptions are reported with synchronous -CHCK. When disabled, these exceptions are reported by asynchronous -CHCK. The default state for this field is disabled (see "Exception Reporting" on page 131 for more information on exception reporting).

Channel Check Status Indicator Field

The Channel Check Status Indicator field is a one-bit, read-only field that indicates if channel-check status is available; it is bit 6 of POS Register 5. The default state of this field is 1. This field is required if the Channel Check field is supported by the adapter.

When this field is a 1, it indicates that status for the channel-check condition is not available. When the field is 0 and the Channel Check field is 0, it indicates that POS Registers 6 and 7 contain the channel-check status or a pointer to the status. (If an adapter does not support -CHCK, bit 6 of POS register 5 can be used for other configuration information.)

Data and Address Parity Enable Field

The Data and Address Parity Enable field is a one-bit, read/write field that is required if the adapter supports address or data parity. This field enables address or data parity on the channel. The default state for this field is disabled. While disabled, the device does not drive either -DPAREN or -APAREN active, and does not check parity from the channel. This field can be a two-bit field to individually enable data and address parity.

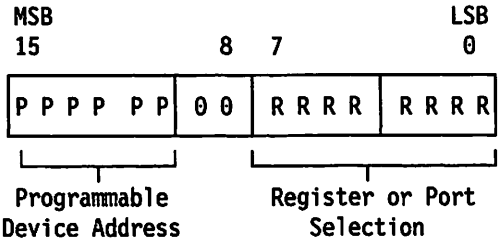
Device I/O Address Assignment

The Device I/O Address Assignment field is a six-bit field that selects the I/O address of the adapter. The six bits represent bits 15 through 10 of the I/O address as shown in the following figure. All I/O devices with a total I/O-address-space requirement greater than 32 bytes must support this field.

The total I/O-address-space requirement for a particular type of device is:

The number of bytes of I/O address space required by one device multiplied by the maximum number of devices of the same type supported in one system.

The I/O address assigned to the device is:



Address bits 15, 14, and 13 must be in POS registers 2 through 5. Address bits 12, 11, and 10 can be in POS registers 2 through 5 or in the POS extension. The address bits in the POS extension must default to 1 at power on.

Matched Memory Enable Field

The Matched Memory Enable field is a one-bit, read/write field that enables or disables support for the matched-memory signals. The default state of this field is disable, which causes the memory adapter to use the default cycle of the Basic Transfer procedures. When the field is in the enable state, the adapter will support the matched-memory signals.

Selected Feedback Return Exception Enable Field

The Selected Feedback Return Exception Enable field is a one-bit, read/write field that selects whether or not a master reports select-feedback-return exceptions. When the field is in the enable state, the adapter reports select-feedback-return exceptions. When the field is in the disable state, the adapter does not report a select-feedback-return exception. The default state of this field is disable.

Streaming Data Enable Field

The Streaming Data Enable field is a one-bit, read/write field that enables support for streaming data procedures. The default state of this field is disable, which causes the adapter to use the basic data transfer procedures. Adapters that support streaming data procedures must support this bit. When disabled, a slave must tri-state its streaming data signals, and a master must tri-state -SD STROBE. When enabled, the streaming data procedure is supported.

Optional Configuration Fields

The following configuration fields are optional.

Note: When CHRESET is driven active, adapters must set all fields to the default state.

Option Select Data Field

The Option Select Data field is four bytes of adapter unique information in POS Registers 2 through 5 that allow developers to customize configuration data for the specific adapter. Bit 0 of POS Register 2 and bits 7 and 6 of POS Register 5 are defined by the architecture; all other bits within the Option Select Data field are available for adapter configuration.

Fairness Feature Enable Field

The Fairness Feature Enable field is a one-bit, read/write field that enables or disables the fairness feature. The default state for this bit is enabled (see "Fairness Feature and the Inactive State" on page 34).

POS Subaddress Extension Field

The POS Subaddress Extension field is an optional, two-byte field that provides a method to extend the POS register space. It is contained in POS Registers 6 and 7 and is used as an index for all setup-cycle accesses to POS Registers 3 and 4. This field allows the subaddressing of up to 128KB of additional information for each adapter.

The POS extension is an extension to POS Registers 3 and 4 and is made up of two one-byte registers. An index of hex 0000 results in setup cycles accessing POS Register 3 and 4. All adapters

supporting the POS extension must set the index to hex 0000 when CHRESET is driven active. The default value for this field is hex 0000.

Adapters supporting POS extension can use POS Registers 6 and 7 as an index register that is automatically incremented for each access to POS Register 3 or 4. Adapters supporting this feature use the Auto-Incrementing field to enable the feature.

When bits 6 and 7 in POS Register 5 are set to 0, reading POS Registers 6 and 7 will return the channel-check-status information. This information can be status or a pointer to the status.

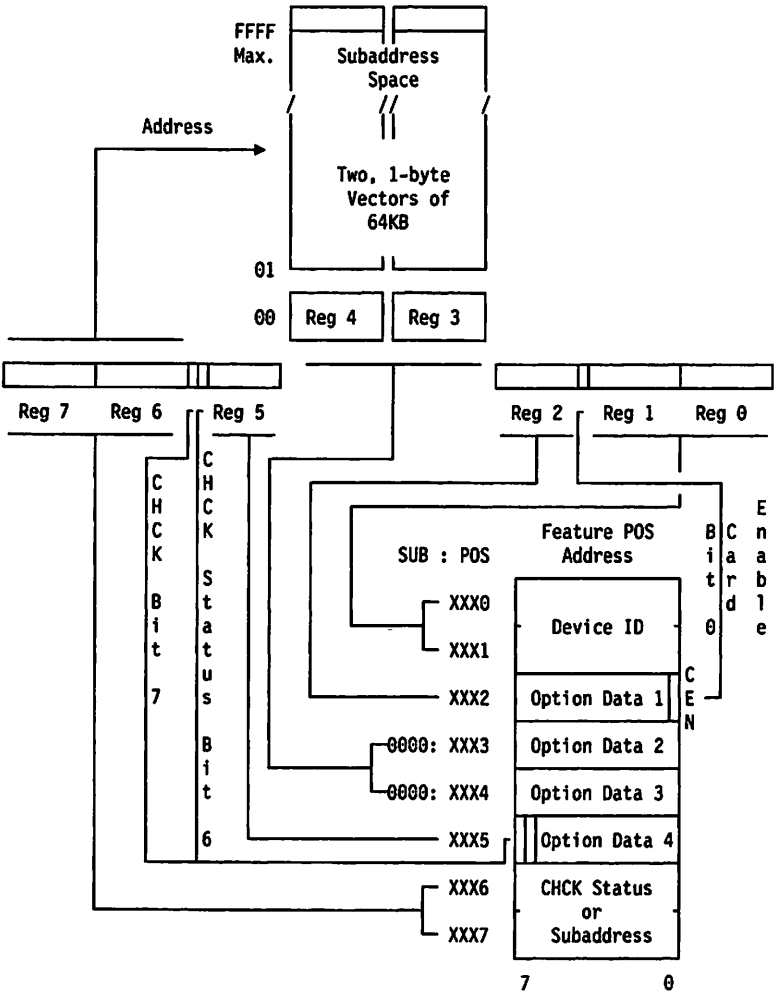


Figure 62. Subaddressing Model

The following is an example of how subaddressing could be used to store or access data in the POS extension.

- POS Register 6 is set to hex 01
- POS Register 7 is set to hex 00
- Access to the POS extension is through POS Register 4
- POS-extension data is in consecutive locations
- Extended POS addressing starts at hex 0001, unless specified by the adapter description file.

Note: The amount of POS-extension information that can be supported is system dependent.

Status Information Field

The Status Information field is contained in POS Registers 6 and 7. The field is used by an adapter to indicate its channel-check status or as a pointer to channel-check status.

System Configuration Procedure

The system configuration procedure uses setup cycles to transfer adapter ID and adapter configuration data between the system master and adapters. The system configuration procedure is similar to the basic transfer procedure, except for the following:

- An adapter is selected by an active -CD SETUP (not by an address decode or arbitration level).
- An adapter must not drive -CHCK active because of a setup cycle.
- A valid POS register address is driven on the address bus; only address bits A(0-2) are decoded by the adapter.
- M/-IO must be in the -IO state during the setup cycle, and adapters must decode M/-IO in the -IO state.
- The adapter does not drive -CD SFDBK active.
- All configuration data transfers are single-byte (8-bit) transfers. All transfers to an 8-bit data port occur on D(0-7). If a 16-bit or 32-bit slave drives -CD DS 16 or -CD DS 32, the 8-bit transfer occurs on the data lines indicated by -SBHE, A0, and A1, or by -BE(0-3) (see Figure 29 on page 53).
- The minimum setup cycle is 300 nanoseconds; adapters can extend the cycle be driving CD CHRDY inactive.

Address Bus During Setup Cycles

During a setup cycle, a valid POS-register address is driven on the address bus; only address bits A(0-2) are decoded by the adapter. The address driven on the bus must not be the I/O address of any adapter.

The following figure shows the organization of the address space used by POS during setup.

-CD SETUP	M/-IO	Address A2 A1 A0	Function
0	0	0 0 0	Adapter ID (Low Byte)
0	0	0 0 1	Adapter ID (High Byte)
0	0	0 1 0	Option Select Data Byte 1
0	0	0 1 1	Option Select Data Byte 2
0	0	1 0 0	Option Select Data Byte 3
0	0	1 0 1	Option Select Data Byte 4
0	0	1 1 0	Subaddress Extension/Channel Check Status (Low Byte)
0	0	1 1 1	Subaddress Extension/Channel Check Status (High Byte)

Figure 63. POS I/O Address Decode

Card Enable Field During Setup Cycles

An adapter selected by -CD SETUP must enable its drivers, regardless of the state of the Card Enable field.

Channel Check Status During Setup Cycles

When -CHCK is driven active, the Channel Check field indicates that the adapter drove -CHCK active (except for address parity). The Channel Check, Channel Check Status Indicator, and Status Information fields (and channel-check status, if the field is used as a pointer) must remain unchanged until the system master resets the Channel Check field to 1 or until CHRESET is driven active.

-CD SETUP During Setup Cycles

During a setup cycle, only one adapter at a time is selected. The adapter is selected by an active -CD SETUP, not by an address decode or arbitration level.

CD CHRDY During Setup Cycles

CD CHRDY can be used to extend the setup cycle, as in the basic transfer procedure (see "Asynchronous-Extended Cycle Timing (≥300 ns)" on page 50 for usage of CD CHRDY in an extended cycle).

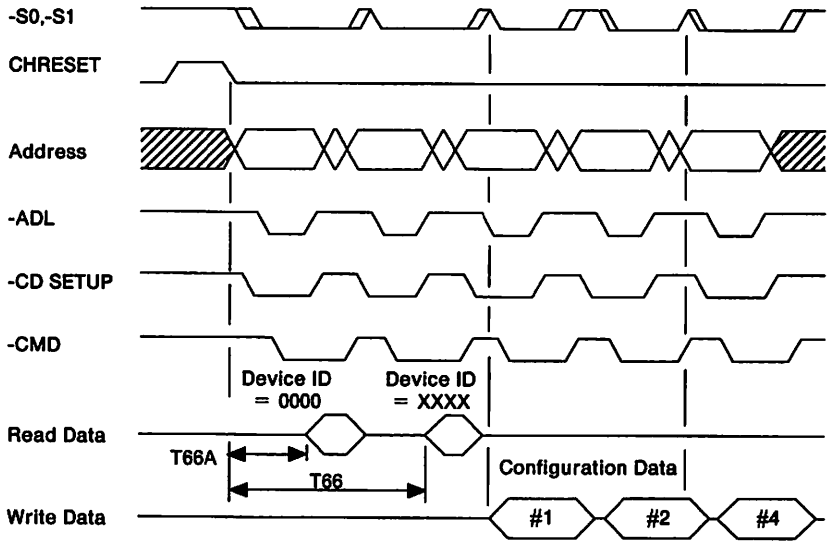


Figure 64. Setup Cycle Overview

System Configuration Timing Diagrams

The following provides the specification for the signal timings used for the system configuration procedure.

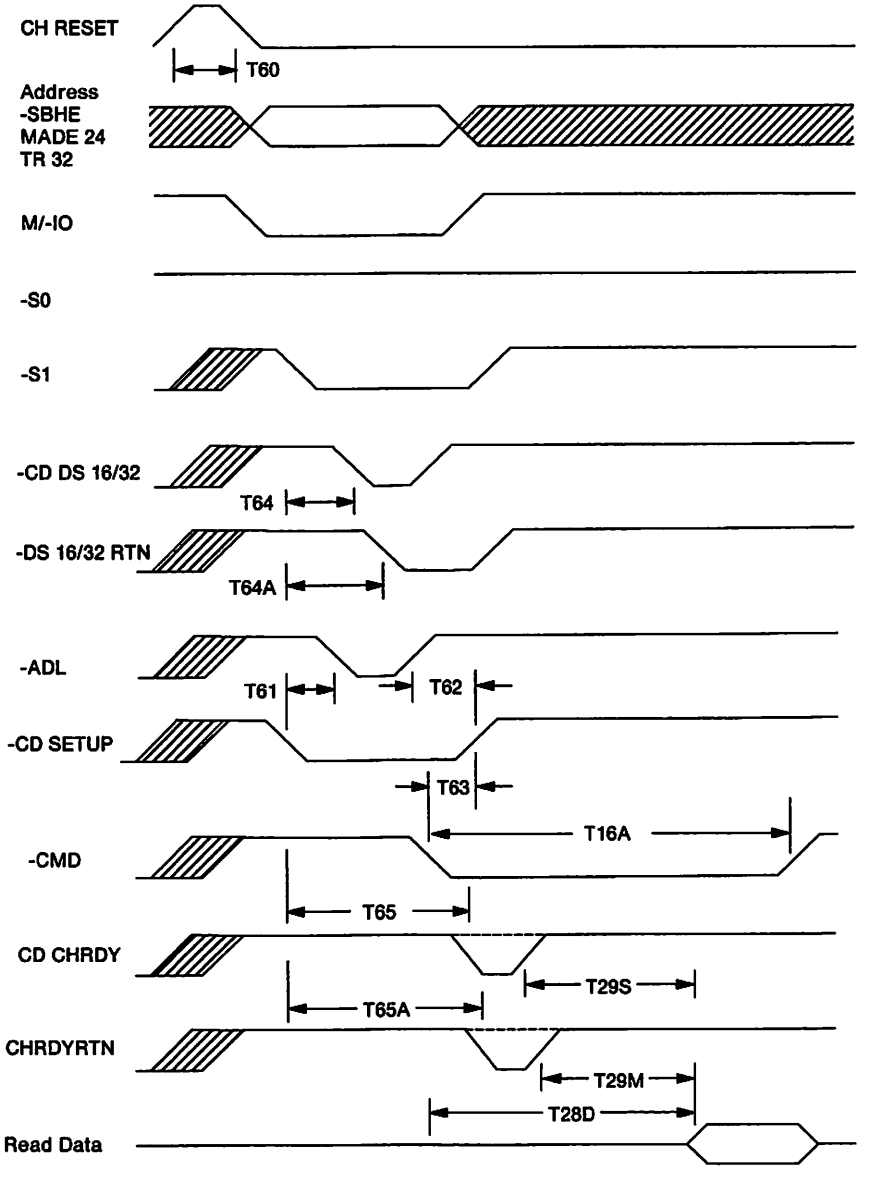


Figure 65. System Configuration Timing

	Timing Parameter	Min/Max	Note
T16A	-CMD Pulse width	190 / - ns	1
T28D	Read data valid from -CMD active	- / 160 ns	1
T29S	Read data from slave valid from CD CHRDY active	- / 60 ns	1
T29M	Read data valid to master from CHRDYRTN active	- / 60 ns	1
T60	CHRESET active pulse width	100 / - ms	
T61	-CD SETUP active to -ADL active	15 / - ns	
T62	-CD SETUP hold from -ADL inactive	25 / - ns	
T63	-CD SETUP hold from -CMD active	30 / - ns	
T64	-CD DS 16 active from -CD SETUP active	- / 25 ns	
T64A	-DS 16 RTN active from -CD SETUP active	- / 45 ns	
T64	-CD DS 32 active from -CD SETUP active	- / 25 ns	5
T64A	-DS 32 RTN active from -CD SETUP active	- / 45 ns	5
T65	CD CHRDY inactive from -CD SETUP active and Status active	- / 100 ns	3
T65A	CHRDYRTN inactive from -CD SETUP active and Status active	- / 120 ns	
T66	Adapter ID valid from trailing edge of CHRESET	- / 1 s	4
T66A	Adapter ID access from trailing edge of CHRESET	1 / - ms	4

Figure 66. System Configuration Timing Specifications

Notes:

1. See "Synchronous-Extended Cycle Timing (300 ns Minimum)" on page 47 and "Asynchronous-Extended Cycle Timing (≥300 ns)" on page 50 for specific details. The following are the timing parameters for the system configuration procedure that are different or additional to those specified for the basic transfer procedure.
2. The minimum setup cycle is 300 nanoseconds. A valid non-card-selecting address must be present on the address bus during setup. During setup, the address on the address bus must not be an I/O address of any adapter.
3. A slave can extend the setup cycle beyond 300 nanoseconds with CD CHRDY.

4. All adapters must present their adapter ID within 1 second after CHRESET is driven inactive. If an adapter is not ready to present its complete adapter ID before T66 time, it must present an ID of hex 0000. If an adapter continues to present an ID of hex 0000 for more than 1 second after CHRESET, it is considered defective.
5. Signals and timings only supported by the 32-bit connector.

Exception Condition Reporting and Handling

An exception condition is an event (including errors) that disrupts normal processing by the master. Resolving an exception usually requires system intervention. Exception conditions are reported to the controlling master or the system master using either -CHCK or an interrupt. Methods for reporting exceptions are described in "Exception Reporting" on page 131.

Note: An adapter whose master function detects exception conditions reports the condition through its I/O slave function by driving -IRQ active.

Exception Conditions

The following types of exception conditions are supported:

- Channel data and address parity exceptions
- Channel non-parity exceptions
- Master- and slave-dependent exceptions
- Channel-timeout exceptions.

Descriptions of each type are included in the following sections.

Channel Data and Address Parity Exceptions

Data and address parity support are optional features on the channel. This type of exception can occur during read or write operations. All participants that support data and address parity checking provide a POS bit (Data and Address Parity Enable field) to enable and disable this feature. The default state of this field is disable. (See "Data and Address Parity Enable Field" on page 106.)

When the Data and Address Parity Enable field is in the disable state, slaves disable checking data and address parity and do not activate -DPAREN during a data-transfer read cycle. Masters disable activation of -DPAREN and -APAREN and do not check data parity on the channel.

Note: Slaves do not disable the driving of -CHCK for non-parity exceptions (independent of the Data and Address Parity Enable field) and masters do not disable the detection of -CHCK.

The description of data and address parity support follows.

Data Parity

Data parity support is optional and allows the use of parity and non-parity adapters on the channel. For exceptions to be detected and reported, both the master and slave involved in the operation must support data parity. -DPAREN is activated by masters on write operations and by slaves on read operations. (-DPAREN is activated by the data source only, and indicates the support of data parity.) The data parity bits, DPAR (0-3), are provided at the same time as data on both write and read cycles, when parity is supported by the master and slave. Only the data bytes used during the data transfer need to have correct parity. For 64-bit streaming, the address parity bits are used during data transfers. -DPAREN is used to indicate that parity is valid on the address bus during 64-bit streaming data transfers. -APAREN remains inactive or is deactivated during the data portion of the transfer.

Read Operations - Basic Transfer

A read cycle with parity checking is shown in Figure 67 on page 121. It is a basic-transfer read cycle that adds data parity checking. During a read cycle, slaves supporting data parity drive -DPAREN active with the data following the activation of -CMD. When -CMD is deactivated, slaves must drive -DPAREN inactive then tri-state it. This ensures that -DPAREN is inactive before a write cycle begins. If a slave fails to restore -DPAREN, adjacent read and write cycles issued from a controlling master that does not support data parity could cause a slave that supports data parity to erroneously assume the write cycle included parity.

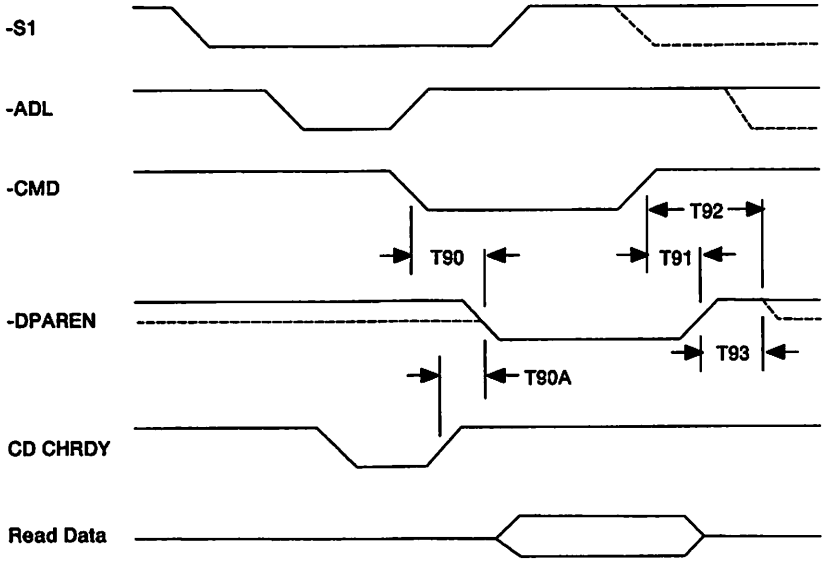


Figure 67. Data Parity Checking Read Cycle - Basic Transfer

	Timing Parameter	Min/Max	Note
T90	-DPAREN active from -CMD active	0 / 60 ns	1
T90	-DPAREN active from -CMD active	0 /160 ns	2
T90	-DPAREN active from -CMD active	0 / - ns	3
T90A	-DPAREN active from CD CHRDY active	- / 60 ns	3
T91	-DPAREN inactive from -CMD inactive	0 / 40 ns	4
T92	-DPAREN tri-stated from -CMD inactive	0 / 80 ns	4
T93	-DPAREN tri-stated from -DPAREN inactive	20 / - ns	

Figure 68. Data Parity Checking Read Timing Specifications

Notes:

1. Default cycle timing only.
2. Synchronous-extended cycle timing only.
3. Asynchronous-extended cycle timing only.
4. -DPAREN must be driven inactive before tri-stating.

Masters supporting data parity must check data parity on read operations, if -DPAREN is activated. If the master detects a parity error, the slave is not informed that the error occurred.

Read Operations - Streaming Data Transfer

Timing of -DPAREN for streaming-data read operations is shown in the following figure.

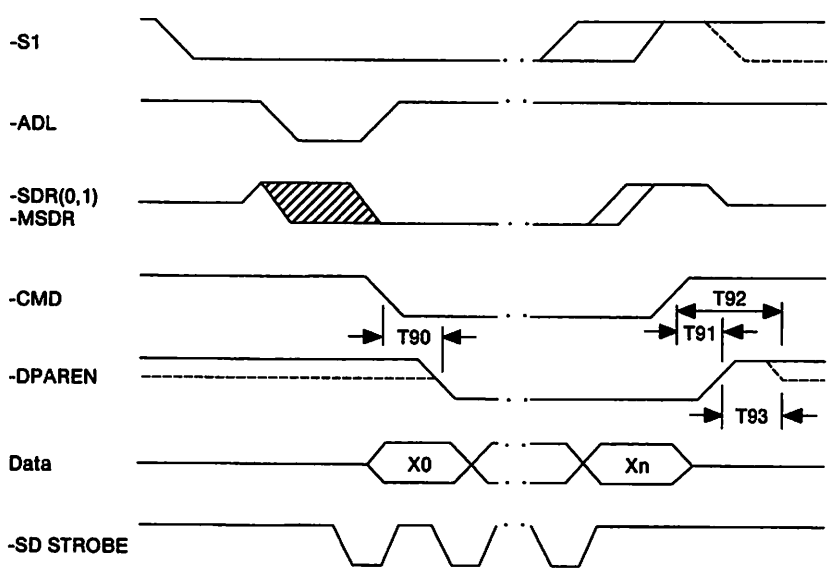


Figure 69. Data Parity Read Cycle - Streaming Data Transfer

Notes:

1. See Figure 68 on page 121 for timing specifications not listed here. For Streaming Data Read Cycles (when CD CHRDY is not driven inactive) T90 is the same as T90 for the basic-transfer cycle.

Write Operations - Basic Transfer

The timing of -DPAREN during write cycles is shown in Figure 70 on page 123. The figure is a basic-transfer write cycle but includes data parity. A master that supports data parity activates -DPAREN during write cycles to indicate that parity is associated with the data. To aid in parity checking at the slave, masters supporting parity are required to provide the data setup time with respect to -CMD. See Figure 70 on page 123 for -DPAREN and T96 timing specifications.

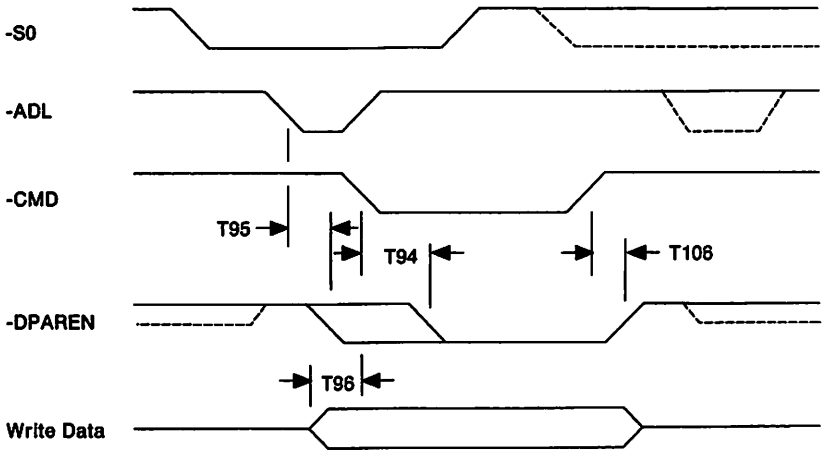


Figure 70. Data Parity Checking Write Cycle - Basic Transfer

	Timing Parameter	Min/Max	Note
T94	-DPAREN active from -CMD active	- / 20 ns	1
T95	-DPAREN active from -ADL active	40 / - ns	2,4
T96	-CMD active from write data setup	15 / - ns	2
T106	-DPAREN inactive from -CMD inactive	20 / - ns	3

Figure 71. Data Parity Checking Write Timing Specifications

Notes:

1. Controlling masters can drive -DPAREN active concurrently with -CMD during write operations. T94 specifies the maximum allowable skew.
2. Applies only to masters supporting data parity.
3. When a write cycle follows a basic-transfer write cycle, the master can optionally hold -DPAREN active between the cycles. When a write cycle follows a streaming data write cycle, the master must drive -DPAREN inactive between cycles. If a master does not hold -DPAREN active between cycles, it must drive -DPAREN inactive following the write cycle. The master must ensure -DPAREN has had time to be restored to the inactive state before sampling its parity check logic during the read cycle. (The master should allow for a 90 ns RC recovery of -DPAREN without an active restore.)
4. Applies only for a master executing a write cycle preceded by a read cycle.

5. See Figure 68 on page 121 for timing specifications not listed here.

Slaves that support data parity, check data parity when -DPAREN is active. In the event of a data parity error during a write, the slave activates -CHCK during the cycle to indicate to the master that an error occurred. Bit 7 in POS Register 5 is set to 0 in the slave when it activates -CHCK. See "Exception Reporting using -CHCK" on page 131 for more information. Slaves do not post an interrupt as a result of detecting a parity error. The slave determines what to do with the data. When sending data to a DMA slave, the DMA controller drives -TC active after detecting -CHCK. Alternatively, the DMA controller can drive -TC active on a subsequent operation with the DMA slave. When the DMA slave detects -TC, it terminates the data transfer and interrupts the system master.

Write Operations - Streaming Data Transfer

The timing of -DPAREN for streaming-data write operations is shown in Figure 72 on page 125. -DPAREN must remain active longer at the end of the cycle. The last data word is not valid until just before the rise of -CMD, and additional time is required for the slave to respond with -CHCK, if a parity error occurred. Slaves use -DPAREN to gate -CHCK onto the channel on streaming-data write operations, and the master must hold -DPAREN active until it can successfully sample -CHCK. Masters that support streaming data transfers with data parity drive -BURST on each transfer and keep it active to delay EOT. The EOT delay prevents conflicts between -DPAREN of the controlling master and the system master, which can execute cycles when ARB/-GNT is in the arbitration state.

The participant receiving the data checks parity on the bytes that are signaled as valid by the data transfer control signals A0, -SBHE or -BE(0-3). The participant driving the data bus (and address bus for 64 bit streaming data) includes parity with the valid data bytes. When the master replicates data bytes for steering purposes, the replicated bytes contain valid parity also. When the system provides data steering the steering includes parity.

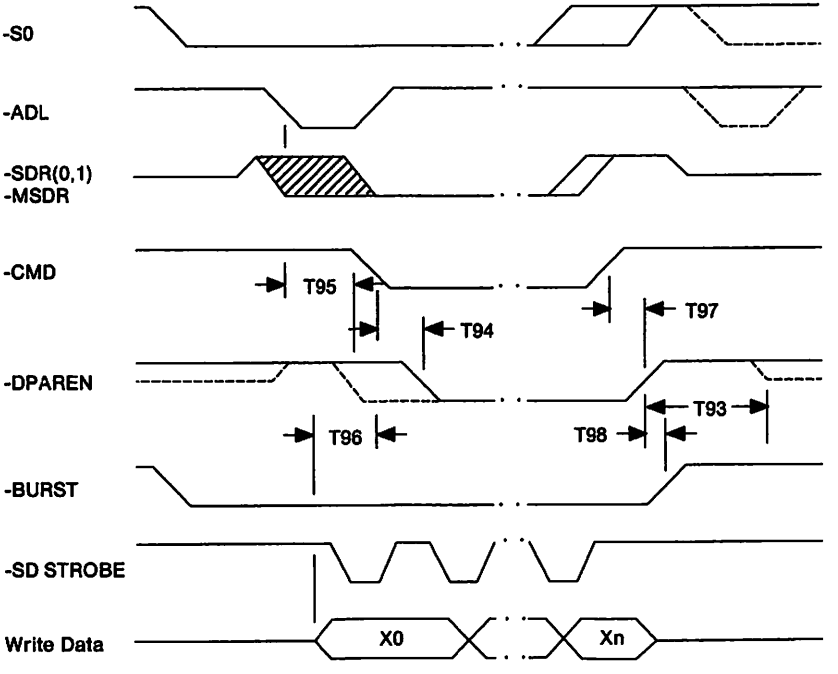


Figure 72. Data Parity Write Cycle - Streaming Data Transfer

	Timing Parameter	Min/Max	Note
T97	-DPAREN hold from -CMD inactive	40 / - ns	1
T98	-BURST inactive from -DPAREN inactive	0 / -	

Figure 73. Data Parity Checking Write Timing Specifications

Notes:

1. Masters must drive -DPAREN inactive before tri-stating it. (See T93 in Figure 67 on page 121).
2. See Figure 67 on page 121 and Figure 70 on page 123 for timing specifications not listed here.

Data Parity Exception Handling

The following is a summary of how participants handle data-parity exception conditions.

Bus Master

During a read cycle a master that supports data parity checking and detects a parity error should do one of the following:

- Suspend the data transfer and post an interrupt reporting a read data parity exception. The master determines what to do with the data.
- Complete the data transfer and post an interrupt reporting a read data parity exception. The master determines what to do with the data.

System Master

When a system master owns the channel and is enabled to support data parity, it must handle data parity errors.

DMA Controller

When a DMA controller that supports data parity detects a data parity error during a read cycle, it should record read data parity exception status and, if the DMA slave is selected, drive -TC active. Alternatively, the DMA controller can drive -TC active on a subsequent operation with the DMA slave.

The system master reads the DMA controller and DMA slave exception status at the end of the DMA operation to determine if the operation completed successfully.

Slaves

When a selected slave detects a parity error during a write cycle, it should activate -CHCK and set bit 7 in POS Register 5 to 0 to indicate that an exception condition occurred (see "Exception Reporting using -CHCK" on page 131). The slave determines what to do with the data.

Address Parity

Address parity support is optional and allows the use of parity and non-parity adapters on the channel. For the exception to be detected and reported, both the master and slave involved in the operation must support address parity.

A controlling master that supports address parity drives -APAREN active to indicate that the parity is associated with the address. When a slave that supports address parity finds -APAREN active and an address parity error is detected, the slave is not selected. The slave includes address parity checking in its address decode. The following figure shows address parity timing requirements.

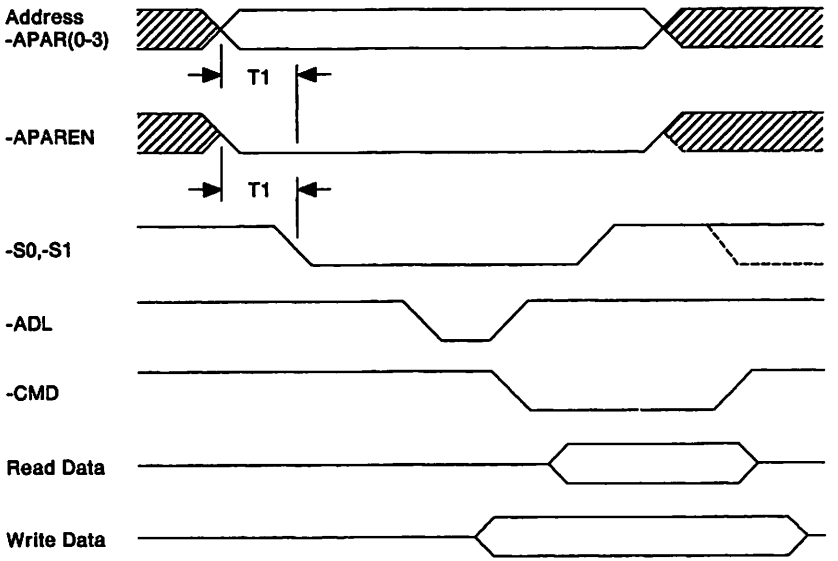


Figure 74. Address Parity

	Timing Parameter	Min/Max	Note
T1	Status active from Address Bus, M/-IO, -APAREN, and -REFRESH valid	10 / - ns	

Figure 75. Address Parity Timing Specifications

-APAREN and address parity bits are valid during the same interval that the address is valid (see Figure 23 on page 44). Only those address bytes used in the data-transfer cycle need to have correct parity. (See data parity for use of address parity bits during 64-bit streaming-data transfers).

Slaves that support address parity must include parity checking in their address decode logic so that, when parity errors occur, they do not become selected and do not drive -CD SFDBK. Slaves check address parity for all data transfers that have -APAREN active. Slaves that support address parity check each address and drive -CHCK active when an address parity error is detected. I/O slaves check parity on all I/O addresses, and memory slaves check parity on all memory addresses. They do not set the channel check status bit in POS Register 5.

Note: A slave treating the address bus as a command should ensure address parity is good before allowing the command to execute.

Channel Non-Parity Exceptions

These exceptions are channel exceptions other than data and address parity exceptions. These exceptions can be signaled by the current slave or detected by the master.

Invalid -DS 16 RTN and -DS 32 RTN Combination

If the system master or a bus master detects an inactive -DS 16 RTN and an active -DS 32 RTN, it suspends the data transfer and posts an interrupt, indicating with interrupt status that the operation was not valid. Detecting the invalid operation is optional.

The DMA controller drives -TC active during the cycle of the detected exception when the DMA slave is selected, and records the status of the invalid operation. Alternatively, the DMA controller can drive -TC active on a subsequent operation with the DMA slave. The timings for -TC are listed in Figure 45 on page 77, Figure 51 on page 89, and Figure 53 on page 90.

Selected Feedback Return Exception

Bus masters monitor -SFDBKRTN under control of a Selected Feedback Return Exception Enable field. Masters that do not support -SFDBKRTN do not need this POS bit. When the Selected Feedback Return Exception Enable field is enabled and -SFDBKRTN is inactive, the bus master sets appropriate status, suspends the data transfer and interrupts the system master.

The DMA controller monitors -SFDBKRTN under control of a Selected Feedback Return Exception Enable field. DMA controllers that do not support -SFDBKRTN do not need this POS bit. When the Selected Feedback Return Exception Enable field is enabled and -SFDBKRTN is inactive, the DMA controller sets the appropriate status and, if a DMA slave is selected, drives -TC active for that cycle. Alternatively, the DMA controller can drive -TC active on a subsequent operation with the DMA slave.

Systems can monitor the -CD SFDBK signals, or -SFDBKRTN, under control of the Selected Feedback Return Exception Enable field. When the Selected Feedback Return Exception Enable field is enabled and a -CD SFDBK is not returned, the system monitor should set appropriate status and generate an interrupt.

These types of exceptions can occur during either read or write operations.

Note: Masters must record the state of -SFDBKRTN when a channel check is detected regardless of the state of the Select Feedback Return Exception Enable field.

Master and Slave Dependent Exceptions

Master- and slave-dependent exceptions are not related to the channel but to the participants. Reporting of this type of exception is specific to each participant and to the specific exception condition.

This type of exception can occur during either read or write operations. A channel check can also occur. Therefore, a master must monitor -CHCK for exceptions during both read and write operations. This is required regardless of whether address parity or data parity are supported.

Master- and DMA slave-dependent exceptions should be reported with an interrupt. However, certain slave-dependent exceptions might require using -CHCK to report the exception. Examples of exceptions that could require using -CHCK are:

- Page fault
- Internal parity error (known bad data without an interrupt procedure to report the error)
- Command queue overflow
- Catastrophic exception condition.

Channel Timeout Exception

If a controlling master or DMA slave fails to release the channel within 7.8 microseconds of -PREEMPT going active, the central arbitration control point can drive ARB/-GNT to the ARB state. In this state, the controlling master degates the address and data buses and all data transfer signals.

Note: Reporting and handling of this type of exception is system dependent.

Exception Reporting

A slave normally reports exceptions with synchronous -CHCK. A master normally reports exceptions with an interrupt and termination of current bus ownership. Exception reporting using -CHCK is described in the following information.

Exception Reporting using -CHCK

Exceptions detected by a slave or channel monitor are usually reported by activating a synchronous -CHCK. If -CHCK is activated, the system master or a bus master suspends processing and generates an interrupt. This interrupt invokes a utility program to handle the exception condition.

Any adapter that activates -CHCK sets bit 7 in POS Register 5 to a 0, to indicate the source of the exception in all cases except address parity errors. If multiple conditions can set -CHCK, the adapter can optionally set status information indicating the cause of the exception. The status or a pointer to it should be placed in POS Registers 6 and 7. Bit 6 in POS Register 5 is then set to 0 to indicate that status information is available. The system master can reset the -CHCK indication by setting a logical 1 into bit 7 in POS Register 5. Setting the bit to 0 will produce unpredictable results. Adapters that do not support -CHCK must ensure bit 7 in POS Register 5 is interpreted as a logical 1.

Synchronous and asynchronous reporting of exceptions using the -CHCK signal are supported. Both reporting methods are described in the following sections.

Synchronous -CHCK

Synchronous -CHCK is signaled by a slave or channel monitor with a pulse on -CHCK. The conditions that are reported with synchronous -CHCK are:

- Data parity errors
- Address parity errors
- Channel non-parity exceptions
- Slave-dependent exceptions.

Note: Slaves or channel monitors that report channel non-parity or slave-dependent exceptions with synchronous -CHCK must support the Channel Check Exception Reporting field (see "System Configuration" on page 101 for a description of this field).

Figure 76 shows a synchronous-exception presentation for the basic transfer cycles.

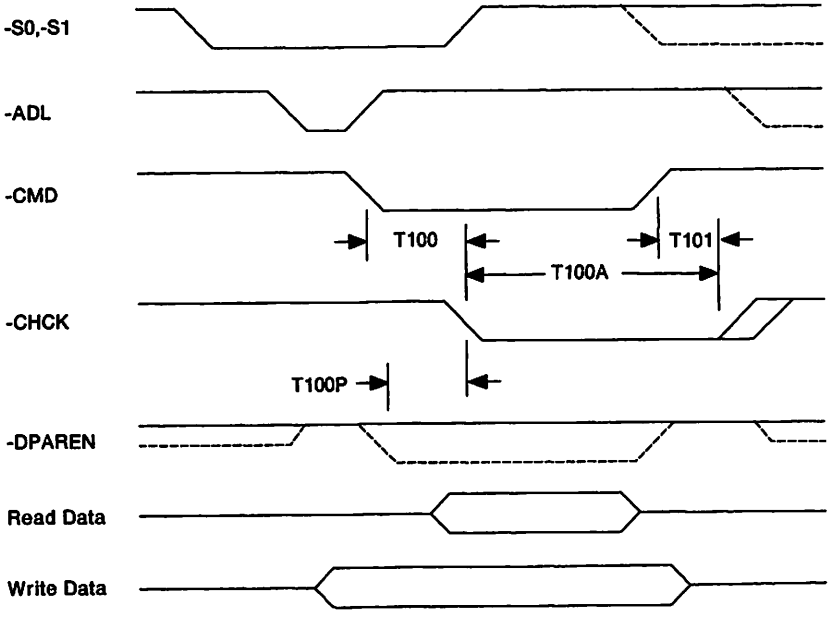


Figure 76. Channel-Check - Default Cycle

	Timing Parameter	Min/Max	Note
T100	-CHCK active from -CMD active	0 / 50 ns	1
T100A	-CHCK pulse width	50 / - ns	1
T100P	-CHCK active from -CMD and -DPAREN active	0 / 50 ns	2
T101	-CHCK disabled from -CMD inactive	0 / 40 ns	

Figure 77. Channel-Check Timing Specifications - Default Cycle

Notes:

1. Slave exception reporting for exceptions other than channel data parity.
2. Activation of -CHCK because of a write-data parity error is timed from -CMD and -DPAREN active, whichever occurs last.
3. Using synchronous -CHCK to report an exception during a transfer that is caused by a condition other than those listed is timed from the falling edge of -CMD.

In a default cycle, the slave activates -CHCK from the activation of -CMD or -CMD and -DPAREN. If the slave requires additional time, it can extend the cycle by deactivating CD CHRDY. Figure 78 shows an extended bus cycle.

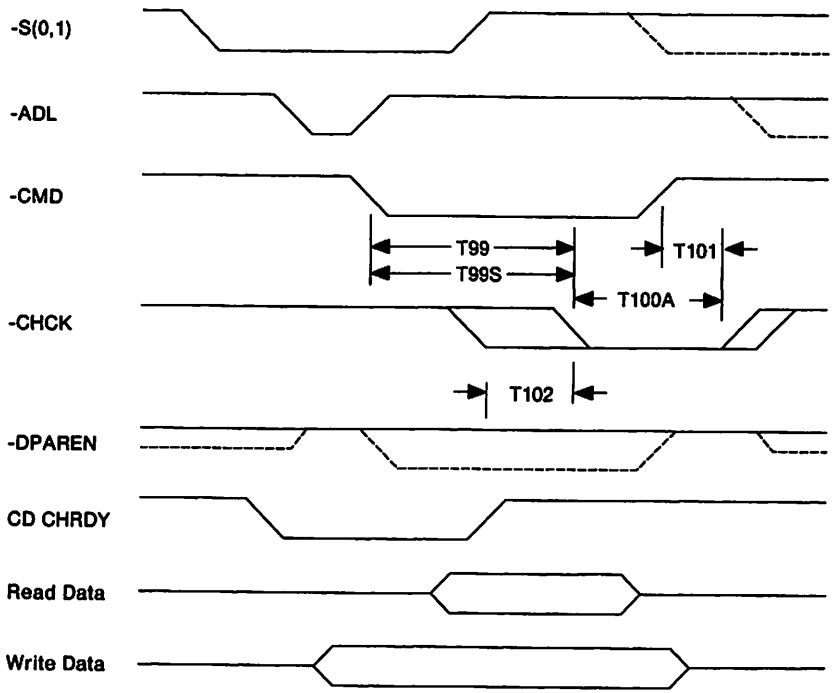


Figure 78. Channel-Check - Extended Cycle

	Timing Parameter	Min/Max	Note
T99	-CHCK active from -CMD active	0 / - ns	2
T99S	-CHCK active from -CMD active	0 /150 ns	1
T102	-CHCK active from CD CHRDY active	- / 50 ns	2

Figure 79. Channel-Check Timing Specifications - Extended Cycle

Notes:

1. For a synchronous-extended cycle only.
2. For an asynchronous-extended cycle and streaming data.
3. See Figure 77 on page 133 for timing specifications not listed here.

Presentation of exceptions for streaming data cycles is similar to a basic transfer cycle. Since exceptions can also occur on any of the data cycles, -CHCK can be activated at any time during the cycle. During read operations, -CHCK is gated by -CMD and is required to be valid coincident with the final data word (Figure 80). During streaming data write operations, -CHCK is gated by -DPAREN and this signal is held active beyond the rise of -CMD to allow time for -CHCK to be activated on the channel (see Figure 72 on page 125).

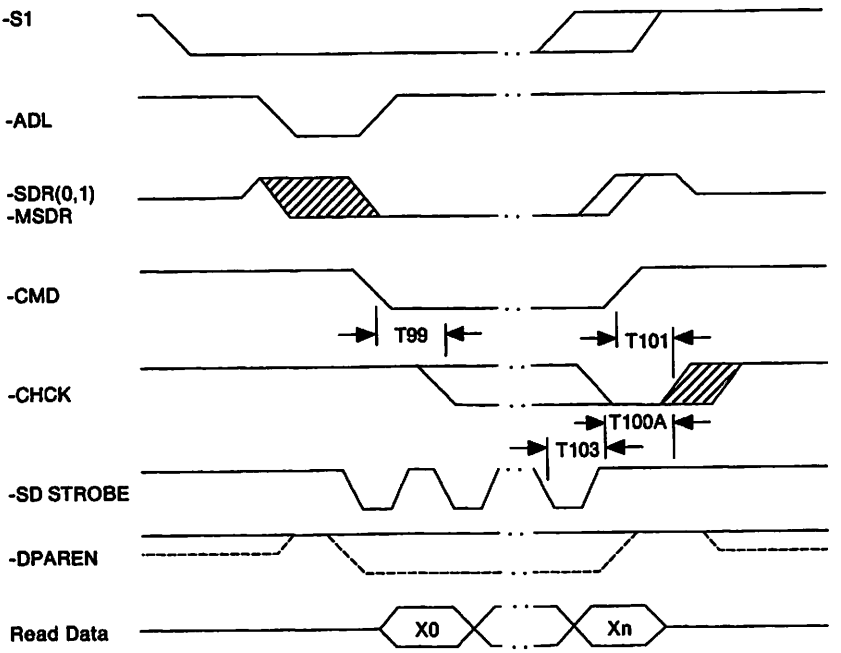


Figure 80. Channel Check - Streaming Data Read Cycle

	Timing Parameter	Min/Max
T103	-CHCK active from last -SD STROBE active	- / 50 ns

Figure 81. Channel-Check Timing Specifications - Streaming Data Read Cycle

Note: See Figure 77 on page 133 and Figure 79 on page 134 for timing specifications not listed here.

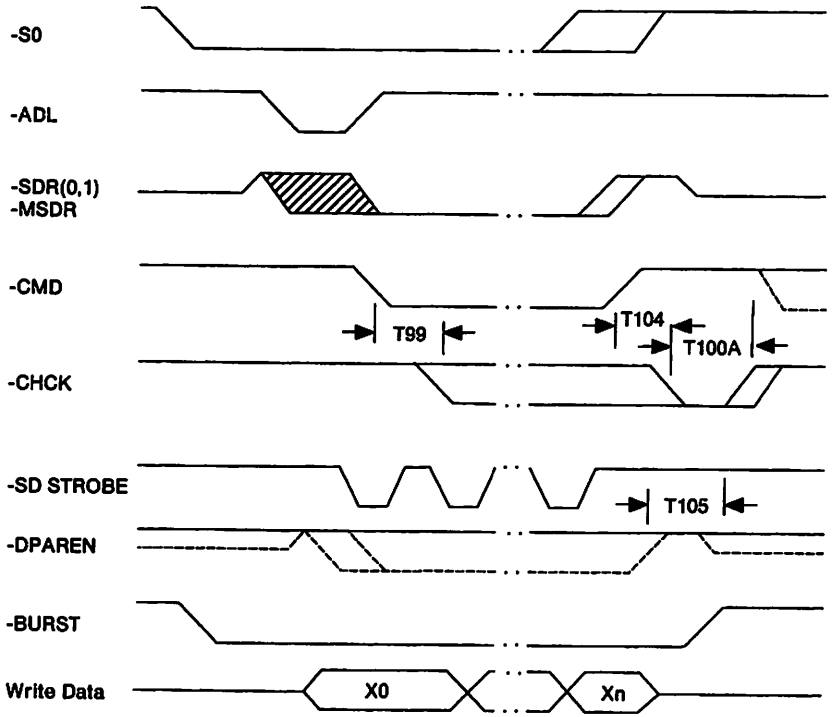


Figure 82. Channel Check - Streaming Data Write Cycle

	Timing Parameter	Min/Max
T104	-CHCK active from -CMD inactive	- / 40 ns
T105	-CHCK disabled from -DPAREN inactive	0 / 40 ns

Figure 83. Channel-Check Timing Specifications - Streaming Data Write Cycle

Note: T100A, T101, and T103 apply to -CHCK for all conditions other than data parity exceptions.

Asynchronous -CHCK

An asynchronous -CHCK is indicated by driving -CHCK active asynchronously to a data transfer. -CHCK remains active until reset by the system master. The adapter that activates an asynchronous -CHCK sets bit 7 in POS Register 5 to a 0. The asynchronous -CHCK is reset by the system master setting bit 7 in POS Register 5, to a 1 (in the adapter that activated -CHCK). The time that asynchronous -CHCK remains active is system master dependent.

Because asynchronous exceptions are latched, -CHCK is detected during the reading of the adapter configuration registers to determine the source of the exception. System masters should ignore -CHCK while collecting status associated with the signaling of -CHCK.

The conditions that can be reported with asynchronous -CHCK are:

- Channel non-parity exceptions
- Slave-dependent exceptions.

If the exception cannot be reported with an interrupt or a synchronous -CHCK and the condition would otherwise go undetected, the exception can be reported with an asynchronous -CHCK. **Using asynchronous -CHCK can cause momentary suspension of all I/O activity and could require a system reset.**

Exception Reporting Using Interrupts

If an interrupt is used to report an exception, status must be supplied to indicate the type of exception being reported.

- The minimum status for bus masters is:
  - Data parity error during a read operation
  - -CHCK state
  - -SFDBKRTN state (if received).
- The minimum status for slaves is:
  - Data parity error during a write operation
  - Internal error.

Exception Handling

The following describes how participants handle -CHCK conditions.

Bus Master

The bus master monitors -CHCK when it is the controlling master. When the bus master detects -CHCK, it should do one of the following:

- Suspend the data transfer and post an interrupt.
- Complete the data transfer and post an interrupt.

Data sent by the bus master after it detects -CHCK might not be received. Data received by the bus master after it detects -CHCK might not be valid.

System Master

The system master monitors -CHCK when it is the controlling master. If -CHCK is detected, the system master will handle the exception condition. The system master can monitor -CHCK at all other times also. The action taken is system dependent.

The method for handling exceptions is software dependent and will include setting the -CHCK status bit to 1 in participants that have activated -CHCK, if exception recovery is part of the action taken.

DMA Controller

A DMA controller monitors -CHCK when it is the controlling master. When it detects -CHCK active, the DMA controller should record channel-check status, record the state of -SFDBKRTN, and, if the DMA slave is selected, drive -TC active. Alternatively, the DMA controller can drive -TC active on a subsequent operation with the DMA slave.

The system master reads the DMA controller and DMA slave exception status, when the end of the DMA operation is signaled, to determine if the operation completed successfully.

Slaves

When a slave activates -CHCK, it must set bit 7 in POS Register 5 to a 0 (except for address parity checking). The system master, upon resolving the -CHCK exception, will reset the bit to a 1.

Select Feedback Exception Handling

A -SFDBKRTN exception occurs when the controlling master does not detect -SFDBKRTN active (or -CD SFDBK for system masters). Reporting of this exception is optional. However, all masters that receive -SFDBKRTN must record its state when an exception occurs. Reporting -SFDBKRTN exceptions is controlled by the Select Feedback Return Exception Enable field. If the master does not receive the signal, the POS field and the monitoring of the signal is not required.

Note: All masters that support address parity must receive -SFDBKRTN.

The following describes how participants handle a -SFDBKRTN exception.

Bus Master

A bus master that is enabled to report a -SFDBKRTN exception and does not receive a -SFDBKRTN should do one of the following:

- Suspend the data transfer and post an interrupt
- Complete the data transfer and post an interrupt.

The status provided with the interrupt includes the lack of -SFDBKRTN. Data sent by the bus master after it detects the lack of -SFDBKRTN might not be received. Data received by the bus master after it detects the lack of -SFDBKRTN might not be valid.

System Master

Reporting a -SFDBKRTN exception and handling the exception is system dependent.

DMA Controller

A DMA controller that is enabled to report a -SFDBKRTN exception and receives no -SFDBKRTN should record status and, if the DMA slave is selected, drive -TC active. Alternatively, the DMA controller can drive -TC active on a subsequent operation with the DMA slave.

At the end of the DMA operation the system master reads the DMA controller and DMA slave status to determine if the operation completed successfully.

Channel Reset

CHRESET is used to force all system functions and channel participants to a defined state. The following situations cause a channel reset.

- A programmed-controlled channel-reset request
- A system power-on reset, which occurs:
  - When the system is powered on
  - When a low-voltage condition is detected at the input of the power supply.

CHRESET is active for a minimum amount of time, after all voltages in the system are within operating limits. See Figure 66 on page 116 for the T60 timing specifications.

After a channel reset or a system power-on reset, the state of the Micro Channel buses and signals are as follows:

	System Power-On Reset		Channel Reset	
Buses and Signals	**Adapter**	**System**	**Adapter**	**System**
Address Bus and Associated Signals	Tri-state	Initial	Tri-state	Unknown
Data Bus and Associated Signals	Tri-state	Initial	Tri-state	Unknown
Arbitration Bus and Associated Signals	Tri-state	Initial	Tri-state	Unknown
IRQ signal	Inactive	Initial	Inactive	Initial
Channel Check Signal	Inactive	Initial	Inactive	Initial
All Others	Inactive	Initial	Inactive	Unknown

Figure 84. Micro Channel Bus and Signal Status after a CHRESET

Connector Definition

The channel provides all signal, power, and ground signals to adapters through 50-mil channel connectors.

The channel provides two basic types of connectors:

- 16-bit, which support 8- and 16-bit operations
- 32-bit, which support 8-, 16-, 24-, and 32-bit operations.

Pins 01 through 45 support 8-bit operations. Pins 46 and 47 are keys (physical notch). Pins 48 through 58 provide additional power and signals to support 16-bit operations. For 32-bit adapters that function in 16-bit connectors, pins 59 and 60 are keys; for 32-bit adapters that cannot function in 16-bit connectors, pins 59 and 60 are reserved. Pins 61 through 89 are used with pins 01 through 58 to support 32-bit operations.

The power and ground pins on side A of each connector are offset from side B by 2 pins, and every fourth pin on either side of each connector is an ac ground.

Video and matched-memory extensions to the basic 16- and 32-bit connectors are implemented on a system-by-system basis. For more information, see the technical reference for the specific system.

Note: Adapter designs should not extend the card-edge connector beyond the basic 16- or 32-bit connector unless the signals provided by the extension are used by the adapter.

All of the logic signal lines are transistor-transistor logic (TTL) compatible.

© Copyright IBM Corp. 1989, 1991

The following figure shows the basic types of channel connectors with optional extensions.

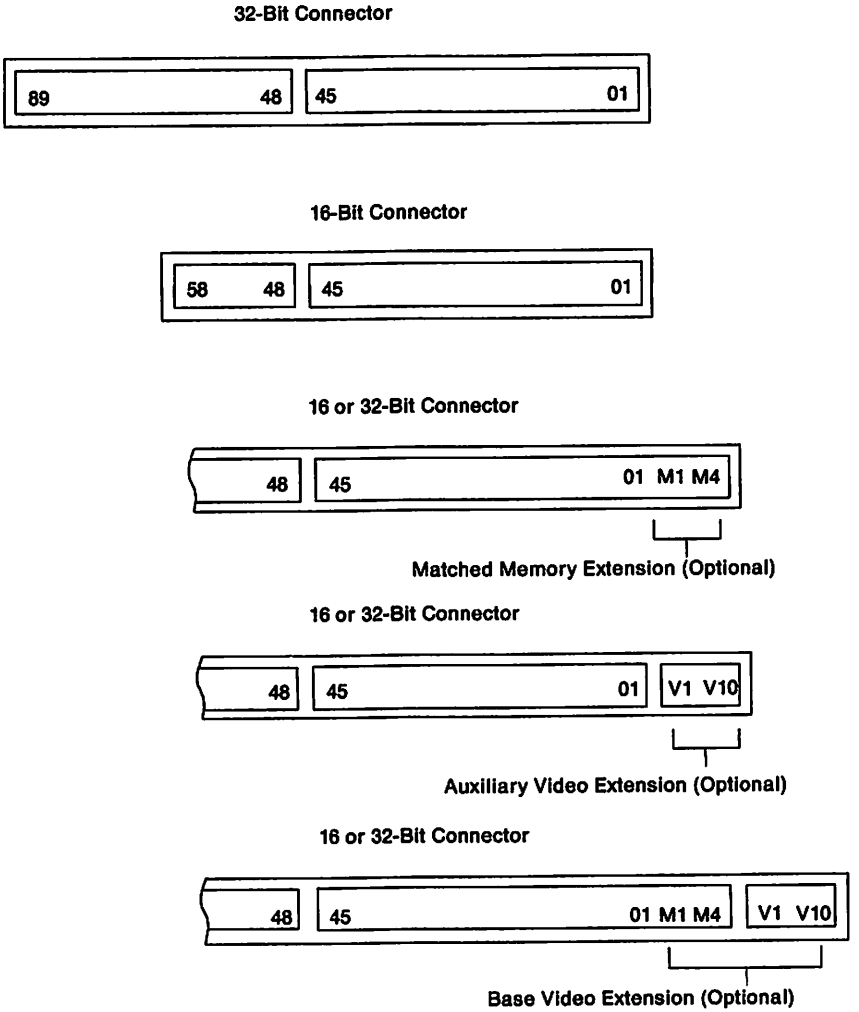


Figure 85. Micro Channel Connectors

Warning: Any signals shown or described as "Reserved" should not be driven or received. These signals are reserved to allow compatibility with future implementations of the channel interface. Serious compatibility problems, loss of data, or permanent damage can result to features or the system, if these signals are misused.

Micro Channel Connector

The following three figures show the signals and the voltages assigned to the 32-bit channel connector. The 16-bit connector is a subset of the 32-bit connector consisting of pins 1 through 58. A key is provided at pin locations 46 and 47 for mechanical alignment.

Rear of the System Board

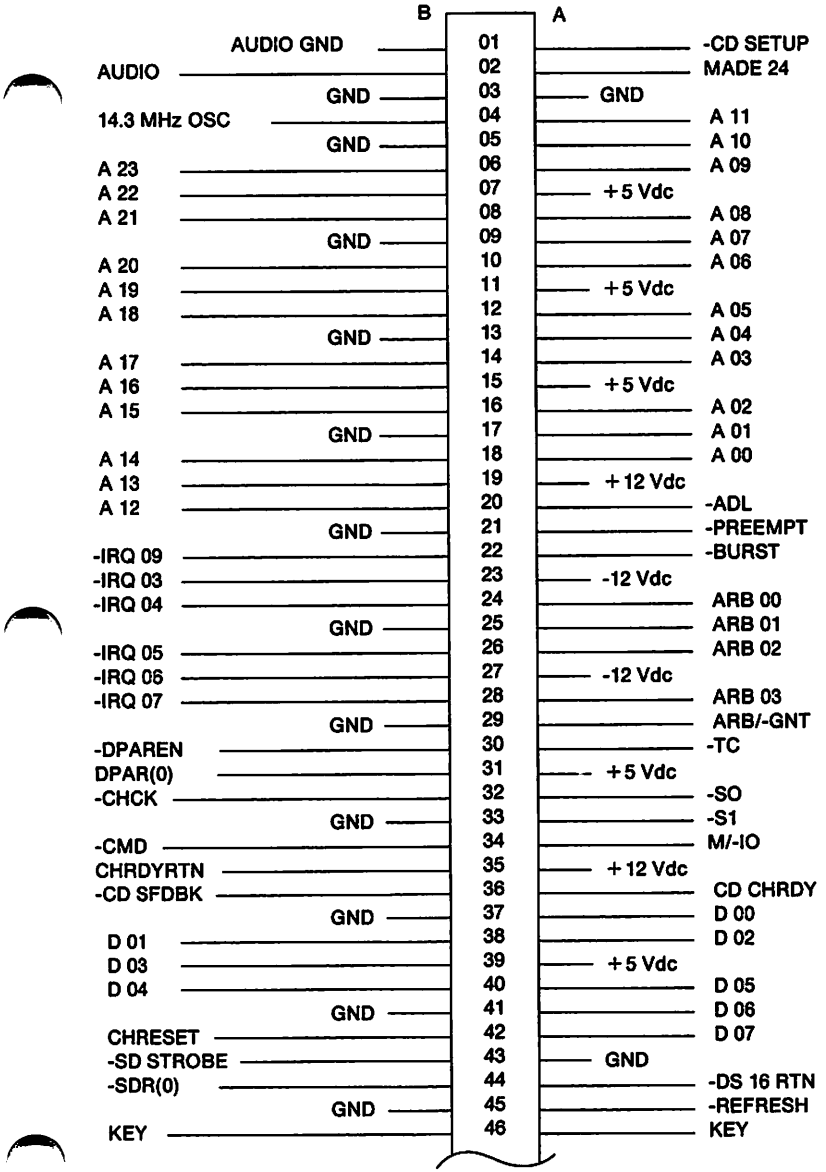


Figure 86. Micro Channel Connector

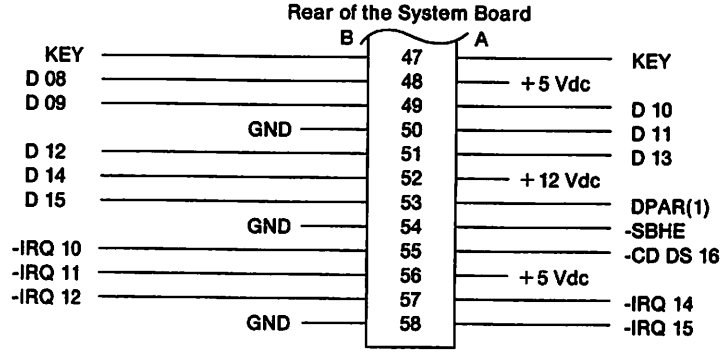


Figure 87. Micro Channel Connector - continued

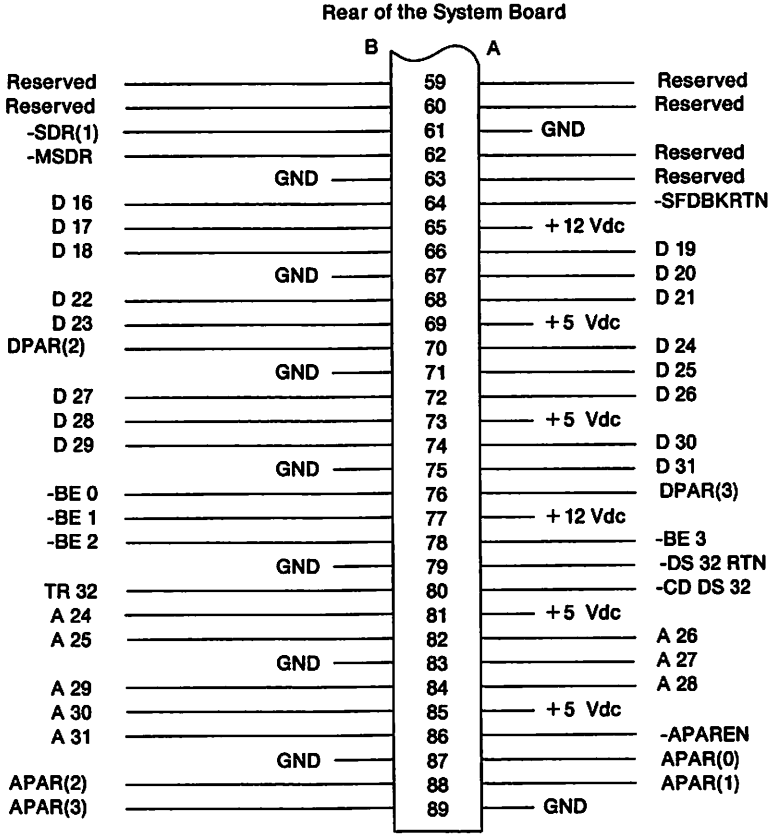


Figure 88. Micro Channel Connector - continued

Matched-Memory Extension

This extension provides additional signals to accommodate matched-memory cycles. The following figure shows a connector with a typical set of matched-memory signals. Refer to the system-specific technical reference for the system you are dealing with for further information.

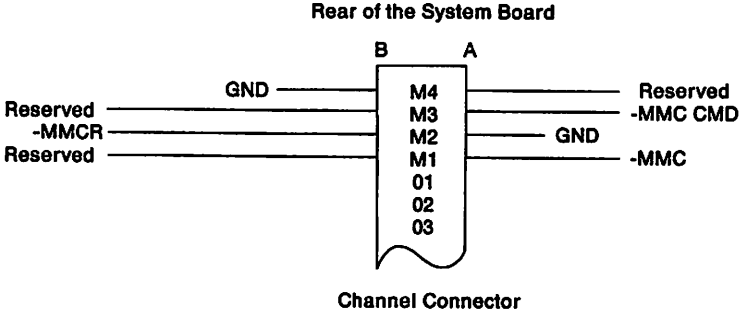


Figure 89. Channel Connector Voltage and Signal Assignments (Matched-Memory Extension)

Auxiliary Video Extension

This extension provides a video adapter with access to resources of the base video subsystem and allows it to extend or supplement the functions provided.

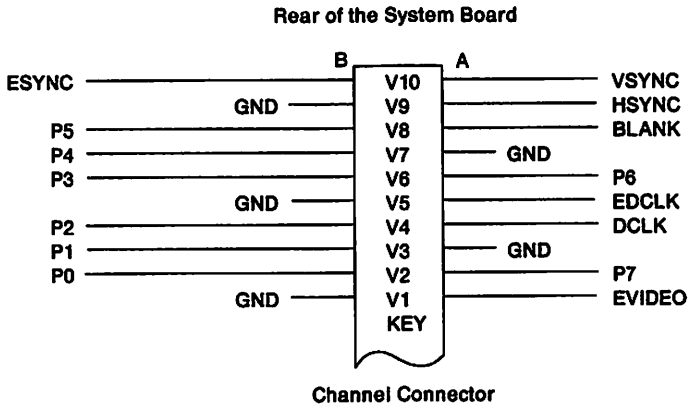
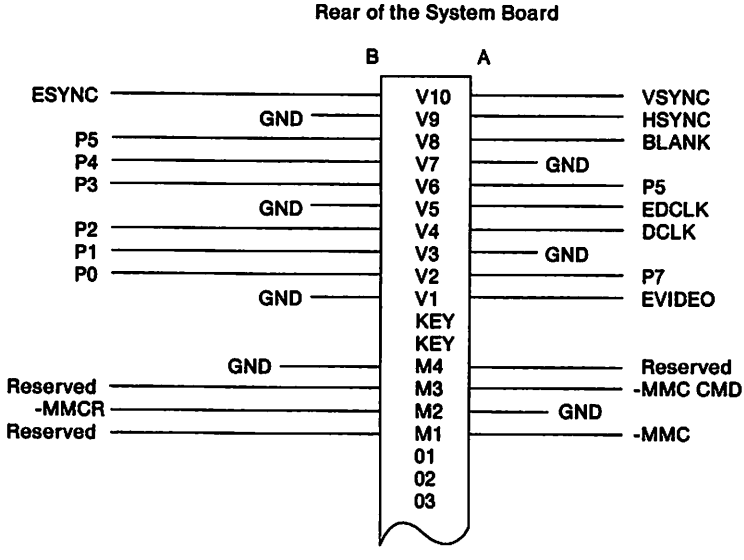


Figure 90. Auxiliary Video Extension

Base Video Extension

This extension is for adapters that provide the base video subsystem. Only systems without a base video subsystem on the system board have a connector with this extension.



Notes:

1. The KEY between M4 and V1 is a physical key.
2. Support of the matched-memory signals (M1 - M4) is optional.

Figure 91. Base Video Extension

Adapter Physical Specifications

The following are the physical specifications for Micro Channel adapters; these adapters can be Type 3, Type 3 half-card, or Type 5. The type number identifies the dimensions of the adapter. Systems specify the adapter type that they support.

Systems that specify support for the Type 3 adapter do not provide the space or the current capacity to accommodate Type 5 adapters (but can accommodate Type 3 half-card adapters). Systems that specify support for the Type 5 adapter support the Type 3, Type 3 half-card, and Type 5 adapters.

Adapter Size

The following are overall dimensions for Micro Channel adapters. Additional and more detailed information (plus tolerances) is available in Micro Channel Adapter Design.

Type 3 Adapter

- Card height - 3.475 in.
- Card plus retainer length - 12.3 in.

Type 3 Half-card Adapter

The Type 3 half card adapter does not require a retainer.

- Card height - 3.475 in.
- Card plus retainer length - 6.35 in.

Type 5 Adapter

The Type 5 adapter does not require a retainer.

- Card height - 4.825 in.
- Card length - 13.1 in.

Connector Spacing

- Minimum Spacing from Card Center to Card Center - 0.85 in. (Maximum spacing is system dependent.)
- Maximum Component Height (Measured from card surface.)
  A side (without electrical insulation) = 0.6 in
  A side (with electrical insulation) = 0.675 in
  B side = 0.105 in.

Electrical insulation is also required if the card bows in excess of 0.025 inch. Measure the bow at a point 0.1 inch from the top of the card and 5.75 inches from the connector end (rear) of the card.

Power

The following figures list the minimum available load current for each voltage present on each connector. Some systems can provide more current than the amount specified below. For more information, refer to the system-specific technical manual.

Supply Voltage	Minimum Available Current Per 16-Bit Connector	Minimum Available Current Per 32-Bit Connector
+ 5.0 Vdc	1.6 A	2.0 A
+12.0 Vdc	0.175 A	0.175 A
−12.0 Vdc	0.040 A	0.040 A

Figure 92. Minimum current available for the Type 3 adapter.

Supply Voltage	Minimum Available Current Per 16-Bit Connector	Minimum Available Current Per 32-Bit Connector
+ 5.0 Vdc	3.12 A	3.12 A
+12.0 Vdc	0.25 A	0.25 A
−12.0 Vdc	0.10 A	0.10 A

Figure 93. Minimum current available for the Type 5 adapter

The following formulas can be used to determine maximum *statistical* current values.

$$(I_{TC_1} + \cdots + I_{TC_n}) + \sqrt{((I_{MC_1} - I_{TC_1})^2 + \cdots + (I_{MC_n} - I_{TC_n})^2)}$$

Where:

- I_{MC} is the maximum current for each component on a given adapter.
- I_{TC} is the typical current for each component on a given adapter and the sum of all I_{TC} equals I_{TA}.

Note: If I_{MC} or I_{TC} is not available, estimate by using: $I_{TC} = 0.7 \times I_{MC}$

The total channel current is also determined in a statistical manner.
Total Channel Current =

$$(I_{TA_1} + \cdots + I_{TA_n}) + \sqrt{((I_{MA_1} - I_{TA_1})^2 + \cdots + (I_{MA_n} - I_{TA_n})^2)}$$

Where:

- I_{MA} is a maximum statistical current for a given adapter.

- I_{TA} is the typical current for a given adapter.

Voltage Regulation

The voltage regulation at the channel connector is shown in the following table.

Voltage	Pins	Tolerance
Ground	A3, B3, B5, B9, B13, B17, B21, B25, B29,	N/A
	B33, B37, B41, A43, B45, B50, B54, B58	N/A
	A61, B63, B67, B71, B75, B79, B83, B87, A89, BM4*, AM2*	N/A
+ 5.0 Vdc	A7, A11, A15, A31, A39, A48, A56, A69, A73, A81, A85	+5% −4.5%
+12.0 Vdc	A19, A35, A52, A65, A77,	+5% −4.5%
−12.0 Vdc	A23, A27	+9.5% −10% (−10.86 to −13.2 Vdc)
Ground (Auxiliary Video)	BV1, AV3, BV5, AV7, BV9	N/A

* These pins are in the Matched-Memory Extension.

Figure 94. Channel Voltage Regulation

The tolerance includes all power distribution losses in both power and ground planes, up to the pins of the channel connector. It does ***not*** include the drop due to the connector (30 milliohm maximum per contact), or the drop due to distribution within the adapter.

Index

A

abort cycle 43
adapter configuration data 103
adapter ID field 104
adapter I/O address select field 105
adapter ROM/RAM address select field 105
adapters, multiple card 102
address boundary alignment, streaming transfer 63
address bus signals 9
address bus translator 55
address decode latch signal 9
address management, streaming data 62
address parity 127
address parity enable field 106
address parity enable signal 9
address parity signals 10
address space, POS 103
addressing model 6
ADL 9
APAREN 9
APAR0 – APAR3 10
arbiter logic 31
arbiter mismatch 30
arbitration 27
arbitration and the system master 39
arbitration block diagram 29
arbitration bus 7
arbitration cycle 35
arbitration level field 105
arbitration timing 35
arbitration/grant signal 8
ARB0 - ARB3 7
ARB/-GNT 8
architecture extensions 25
asynchronous -CHCK 137
asynchronous-extended cycle 50
AUDIO 19
AUDIO GND 19
audio ground 19
audio sum node signal 19
auto-increment enable field 105
auxiliary video extension 149
auxiliary video signal 23
A0 - A31 9

B

base architecture, 32-bit and 16-bit 25
base video extension 149
basic transfer 40
BE signals 55, 56
BE0 - BE3 13
burst DMA transfer 90
burst mode 40
burst signal 8
burst transfer 8, 32, 86
burst transfer, preempted 33
bus master 3
bus, arbitration 7
byte enable 55
byte enable signals 13

C

card data size 16 signal 13
card data size 32 signal 14
card enable field 104
card selected feedback signal 10
card setup signal 19
CD CHRDY 14
CD DS 16 13
CD DS 32 14
CD SETUP 19
CD SFDBK 10
central arbitration control point 28
central data steering 55
central steering logic 56

© Copyright IBM Corp. 1989, 1991

central translator logic 56
channel check field 104
channel check signal 19
channel connector definition 142
channel connector diagram 143
channel connector voltages and signal assignments 144
channel connectors 142
channel definition 7
channel load current 152
channel ownership 27
channel ready return signal 14
channel ready signal 14
channel reset 141
channel reset signal 19
channel signal groups 20
channel support 55
channel time-out 34
channel timeout exception 130
CHCK 19
CHCK field 104
CHRDYRTN 14
CHRESET 19
CHRESET signal 141
CMD 14
command signal 14
conditionally-required fields 104
configuration data 103
configuration fields 103
configuration, system 101
connectors, channel 142
connector, auxiliary video extension 149
connector, base video extension 149
connector, matched-memory extension 148
connector, 16 bit 144
connector, 32 bit 144
controlling master 27
cycle, arbitration 35
cycle, asynchronous-extended 50
cycle, default 43
cycle, synchronous-extended 48

D

data and address parity enable field 119
data bus signals 13
data bus signals (0 - 31) 13
data bus steering 57
data bus steering control 54
data parity 120
data parity enable field 106
data parity enable signal 14
data parity exception handling 126
data parity read operations 120
data parity signals (0 - 3) 15
data parity write operations 122
data port 3
data replication, data bus 55
data size 16 return signal 15
data size 32 return signal 16
data steering 53
data transfer cycle 40
default cycle 43
default master 27
definition, channel 7
description, micro channel 1
device I/O address assignment field 106
direct memory access (DMA) procedure 83
DMA controller 3
DMA cycle termination 86
DMA slave 4
DMA use of -BURST 87
DPAREN 14
DPAR0 - DPAR3 15
DS 16 RTN 15
DS 32 RTN 16
D0 - D31 13

E

end of transfer 28, 86
EOT 28
error condition 118
exception condition reporting 118
exception conditions 118
exception handling 138

exception reporting 106, 131
exception reporting using external interrupt 138
exception, channel time-out 34
exiting inactive state 35
extensions, architecture 25

F

fairness enable field 108
fairness feature 34
first cycle after grant 37
formula, maximum statistical current 152
formula, total channel current 152

H

handling, exception condition 118

I

inactive state 34
inactive state, exiting 35
interrupt 98
interrupt signals 18
IRQ 14-15 18
IRQ 3-7 18
IRQ 9-12 18
I/O address decode 113
I/O slave 3

L

load current 152
local arbiters 28, 30

M

MADE 24 10
masters 2
master, controlling 27
master, default 27
matched memory enable field 107
matched-memory connector 148
matched-memory signal 23
maximum statistical current formula 152
memory address enable 24 signal 10
memory slave 4
memory/-input output signal 11
micro channel description 1
MSDR 16
multiple card adapters 102
multiple data transfers 28
multiplexed streaming data request signal 16
M/-IO 11

O

option select data fields 108
optional configuration fields 108
OSC 19
oscillator signal 19
ownership, channel 27

P

parity errors 119
POS I/O address selection 112
POS I/O address space 103
POS registers 103
power on reset 141
power specifications 152
PREEMPT 8, 33
preempt signal 8
preempted burst transfer 33
preemption 33
primary cards 102

R

REFRESH 19
refresh signal 19
regulation, voltage 153
reporting procedure, exception condition 118
required configuration fields 103
reserved signals 143
reset 141

S

SBHE 16
SD STROBE 16
SDR(0,1) 16
secondary cards 102
select feedback exception 140
selected feedback return exception enable field 107
selected feedback return signal 11
setup cycle timing 114
SFDBKRTN 11
signal descriptions 7
signal groups 20
signals
  -ADL 9
  -APAREN 9
  -BE0 - -BE3 13
  -BURST 8
  -CD DS 16 13
  -CD DS 32 14
  -CD SETUP 19
  -CD SFDBK 10
  -CHCK 19
  -CMD 14
  -DPAREN 14
  -DS 16 RTN 15
  -DS 32 RTN 16
  -MSDR 16
  -PREEMPT 8
  -REFRESH 19
  -SBHE 16
  -SD STROBE 16
  -SDR(0,1) 16
  -SFDBKRTN 11
  -S0, -S1 17
  -TC 18
  address bus 9
  APAR0 - APAR3 10
  ARB0 - ARB3 7
  ARB/-GNT 8
  AUDIO 19
  AUDIO GND 19
  auxiliary video 23
  A0 - A31 9
  CD CHRDY 14
  CHRDYRTN 14
signals *(continued)*
  CHRESET 19
  data bus 13
  DPAR0 – DPAR3 15
  D0 - D31 13
  IRQ 14-15 18
  IRQ 3-7 18
  IRQ 9-12 18
  MADE 24 10
  matched-memory 23
  M/-IO 11
  OSC 19
  TR 32 18
signals, address bus 9
signals, arbitration 7
signals, interrupt 18
signals, reserved 143
single DMA transfer 88
single transfer 84
slaves 3
specifications, power 152
status field 106
status information field 111
status signals 17
steering control, data bus 54
streaming data address management 62
streaming data cycle termination 73
streaming data enable field 108
streaming data request signals 16
streaming data request 1 signal 17
streaming data strobe signal 16
streaming data timing specifications 76
streaming data transfer 58
streaming transfer, address boundary alignment 63
synchronous -CHCK 132
synchronous-extended cycle 48
system byte high enable signal 16
system configuration 101
system configuration fields 103
system configuration procedure 112
system configuration timing 114

system master 3
system master and arbitration 39
S0, S1 17

T

TC 18
terminal count signal 18
termination, DMA cycle 86
termination, streaming data 73
timing diagrams, system configuration 114
timing, first cycle after grant 37
total channel current formula 152
TR 32 18
TR 32 signal 56
translate 32 signal 18
translator, address bus 55

V

voltage regulation 153

Numerics

16-bit architecture subset 26
16-bit connectors 142
32-bit base architecture 25
32-bit connectors 142

Micro Channel Adapter Design

General Guidelines 1
Dimensions 3
General Design Considerations 26
Safety 26
Thermal 26
Electromagnetic Compatibility 27
Diagnostics 28
Design Guidelines 28

Index 30

© Copyright IBM Corp. 1989, 1990

Figures

1. Examples of Adapter Connectors . . . . . .
2. Type 3 Adapter Dimensions (16-Bit) . . . . . .
3. Type 3 Adapter Dimensions (32-Bit) . . . . . .
4. Type 3 Adapter Dimensions (Half-Card) . . . . . .
5. Type 5 Adapter Dimensions (32-Bit) . . . . . .
6. Connector Dimensions (16-Bit) . . . . . .
7. Connector Dimensions (32-Bit) . . . . . .
8. Connector Dimensions (Auxiliary Video Extension) . . . . .
9. Connector Dimensions (Matched Memory Extension) . . . .
10. Connector Dimensions (Base Video Extension) . . . . . .
11. Connector Dimensions (Common Detail) . . . . . .
12. Typical Assembly for Type 3 Adapter . . . . . .
13. Typical Assembly for Type 5 Adapter . . . . . .
14. Adapter Holder . . . . . .
15. Adapter Retainer . . . . . .
16. Adapter Bracket . . . . . .
17. Vendor Adapter ID Assignments . . . . . .

© Copyright IBM Corp. 1989, 1

General Guidelines

This section provides some basic guidelines to design adapters for systems using the Micro Channel architecture. Topics include physical specifications, general design considerations and design guidelines.

There are two sizes of adapters: Type 3 and Type 5. The type number corresponds to the relative height of the adapter. Type 3 and Type 3 half card adapters can be plugged into systems that provide space and current capacity for Type 5 adapters.

Systems can support different combinations of connectors and extensions on adapters. Figure 1 on page 2 shows the 32-bit and 16-bit connectors and the acceptable extensions for video and matched-memory.

Note: Not all systems support all types of adapters, or connectors and extensions. For information on the type of adapters, connectors, and extension supported and the power provided, see the system-specific technical manual.

To prevent wear, connector contacts should be provided for signals not used by an adapter. See the description of the Micro Channel architecture for more information on channel connectors and signals.

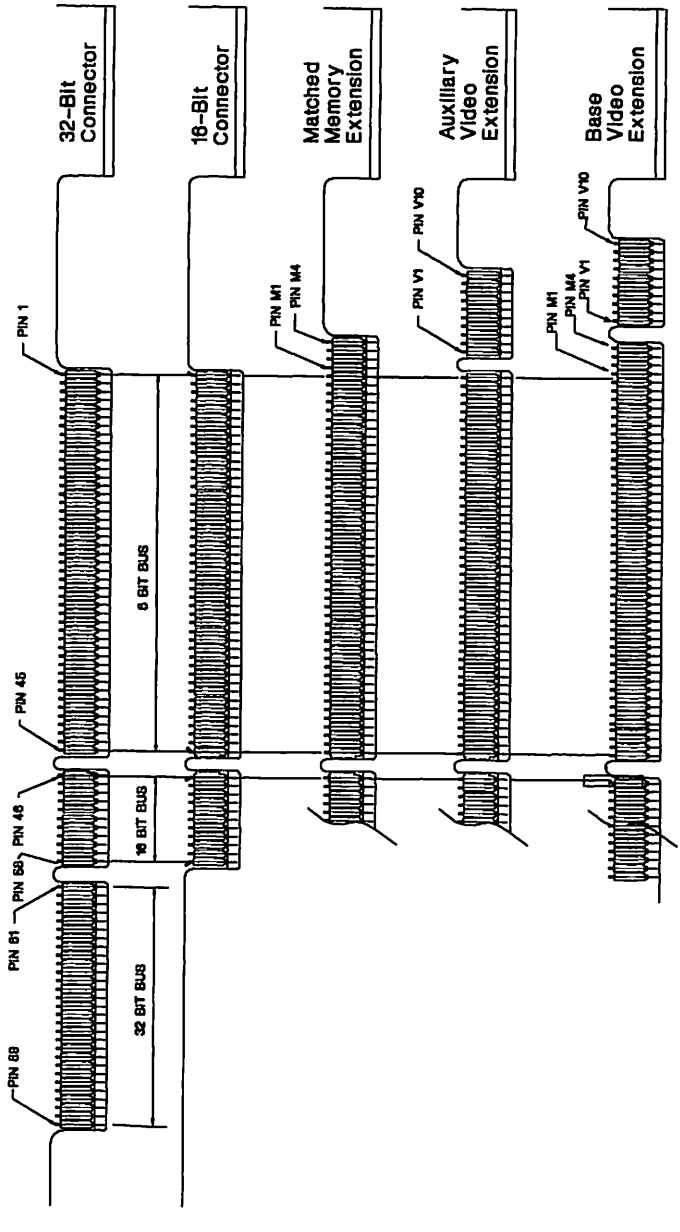


Figure 1. Examples of Adapter Connectors

Dimensions

The following figures show the dimensions of each type of adapter and the associated mounting hardware. The tolerances shown include all individual process tolerances and are not cumulative.

Regardless of the size of the adapter, the maximum height for components mounted on the adapter is 15.240 millimeters (0.600 inch) on the component side (A side) and is 2.667 millimeters (0.105 inch) on the B side. The maximum height on the A side can be extended to 17.145 millimeters (0.675 inch) if the card can be properly cooled and electrically insulated. The width of the electrical insulator must be included in this dimension.

Electrical insulation is also required if the card bows in excess of .635 millimeters (0.025 inch). Measure the bow at a point 2.54 millimeters (0.100 inch) from the top of the card and 146 millimeters (5.75 inches) from the connector end (rear) of the card.

Adapters using CMOS technology should have all plated connector contacts the same length to reduce the exposure of incorrect bias to modules.

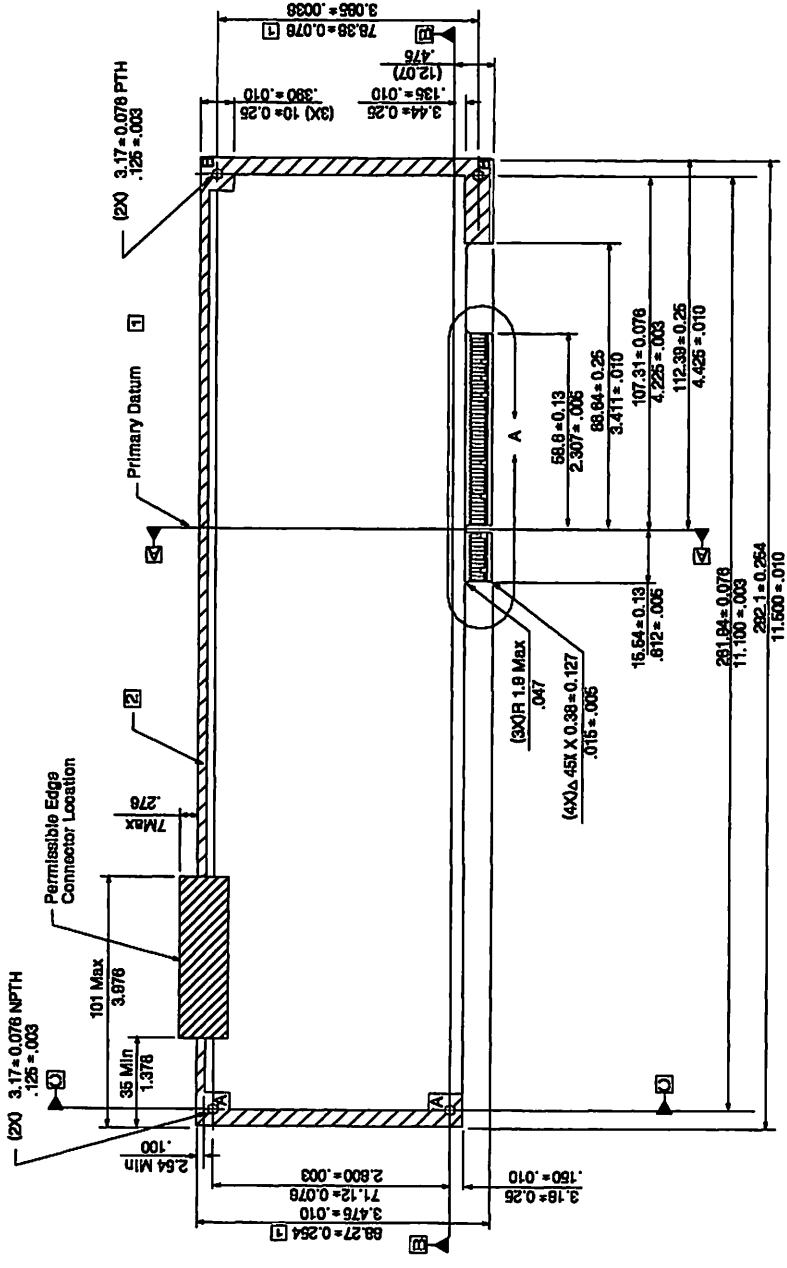


Figure 2. Type 3 Adapter Dimensions (16-Bit)

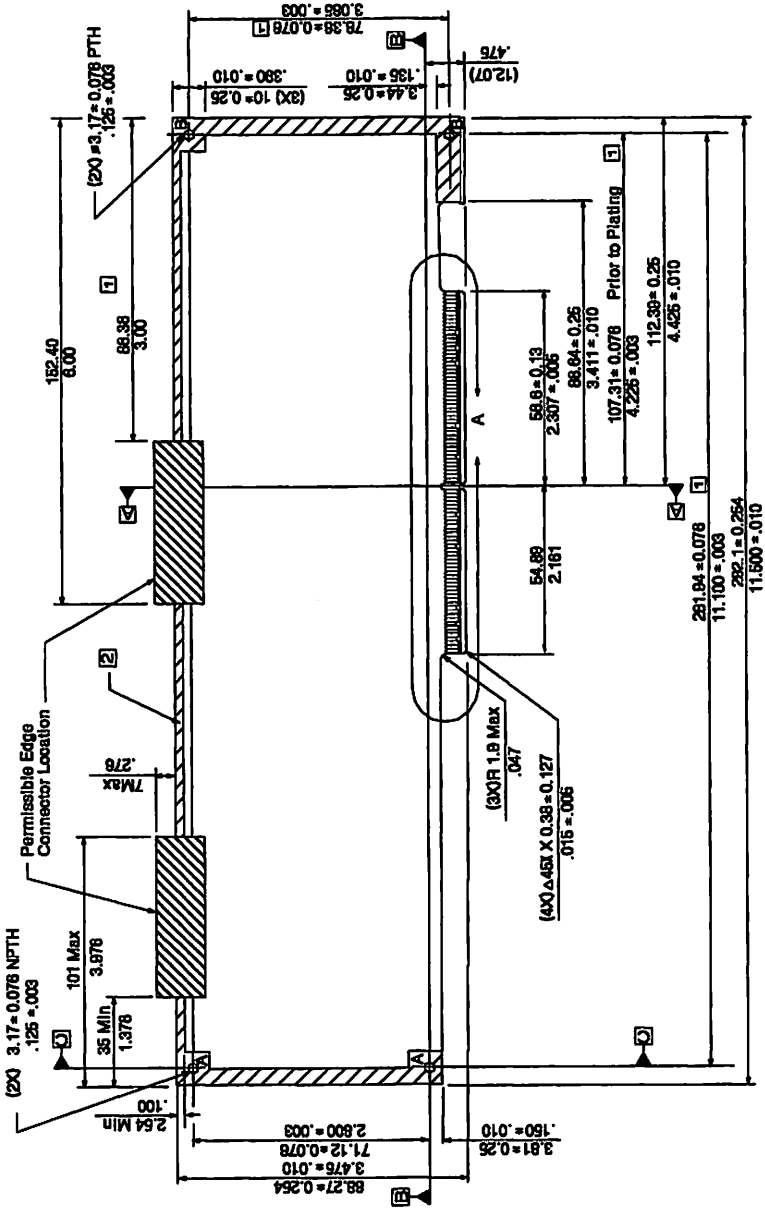


Figure 3. Type 3 Adapter Dimensions (32-Bit)

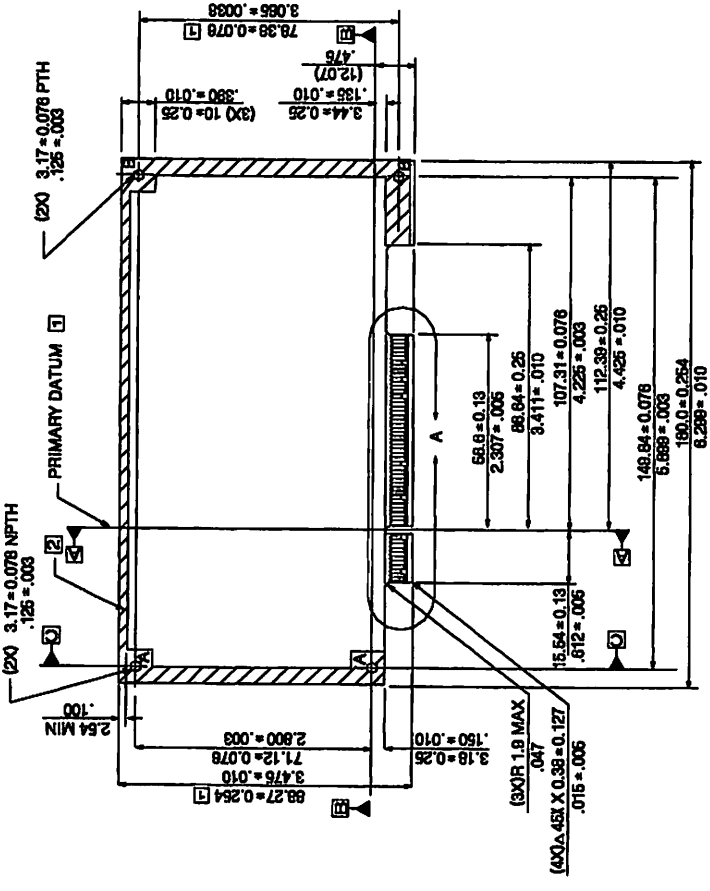


Figure 4. Type 3 Adapter Dimensions (Half-Card)

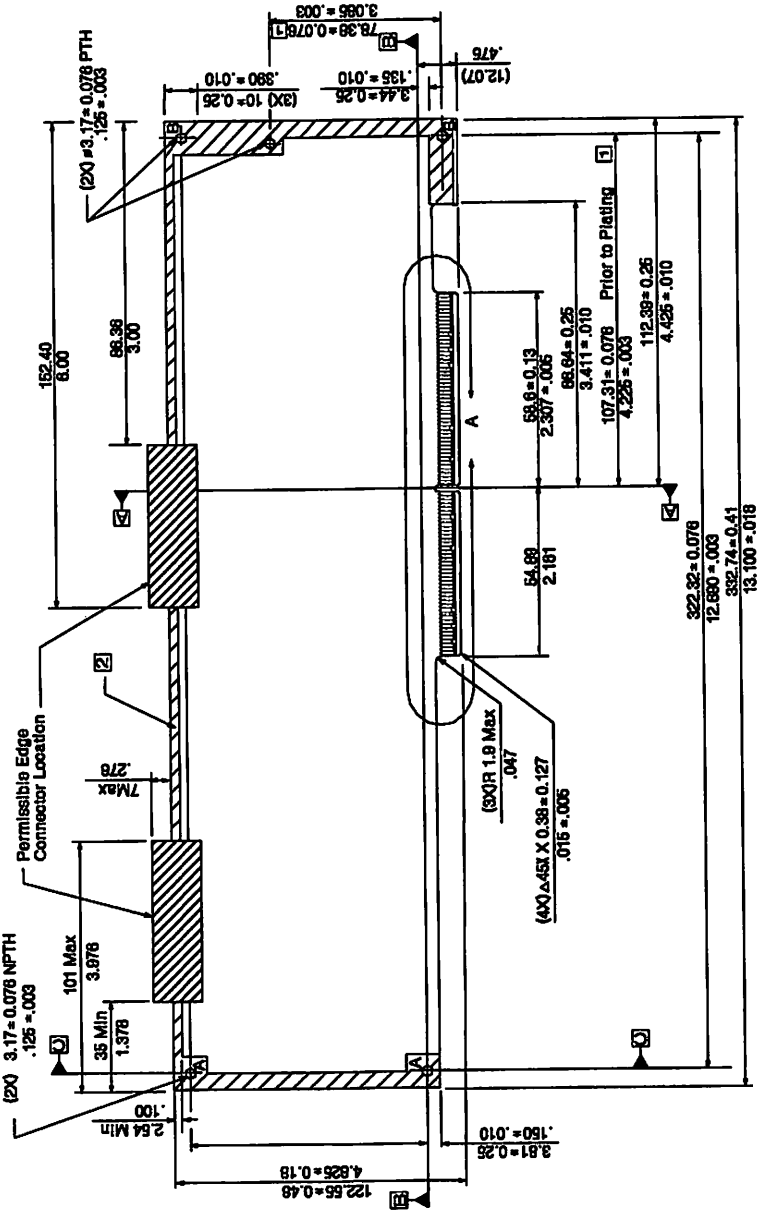


Figure 5. Type 5 Adapter Dimensions (32-Bit)

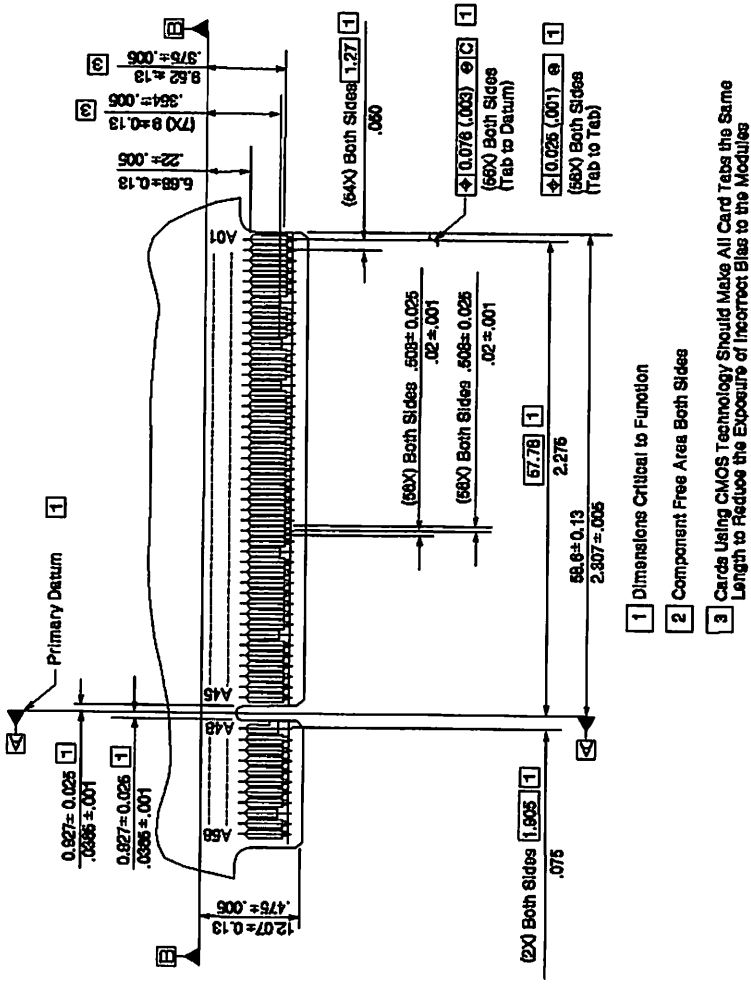


Figure 6. Connector Dimensions (16-Bit)

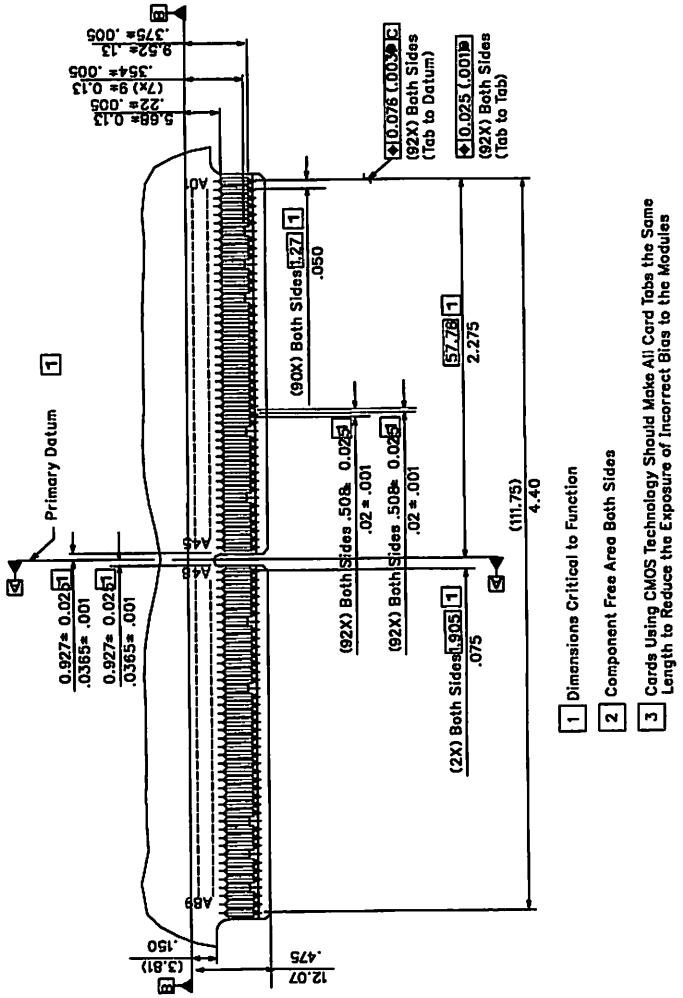


Figure 7. Connector Dimensions (32-Bit)

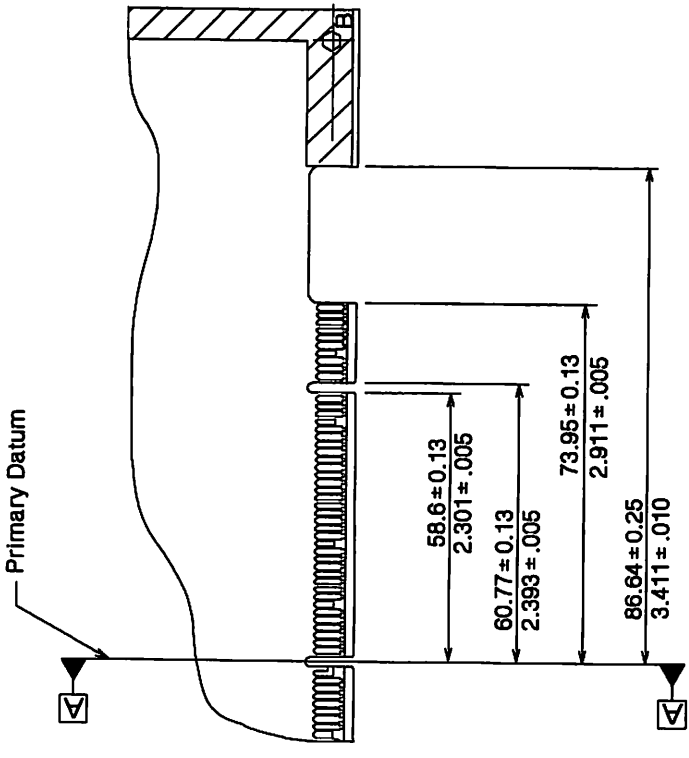


Figure 8 (Part 1 of 2). Connector Dimensions (Auxiliary Video Extension)

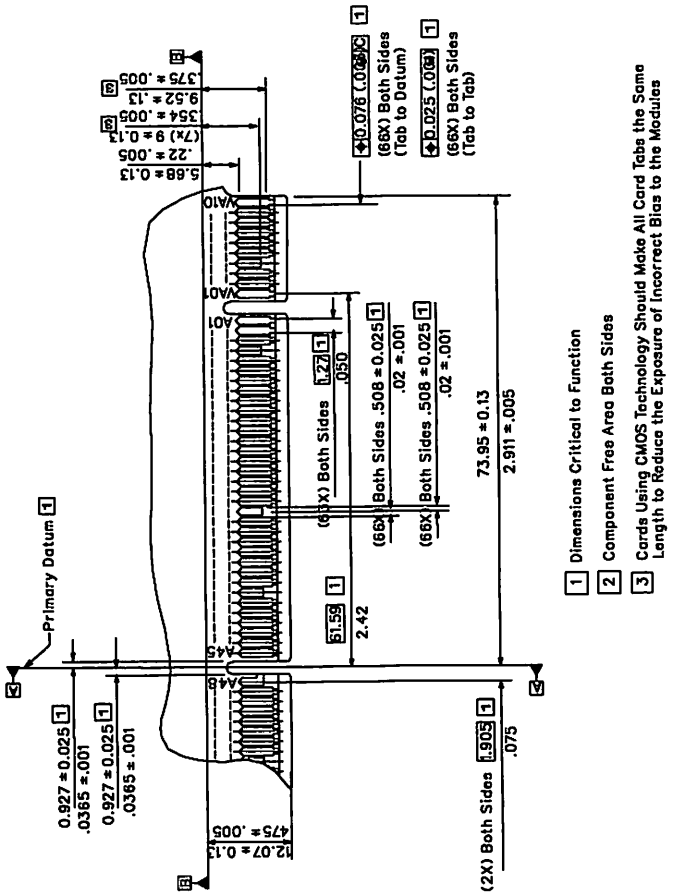


Figure 8 (Part 2 of 2). Connector Dimensions (Auxiliary Video Extension)

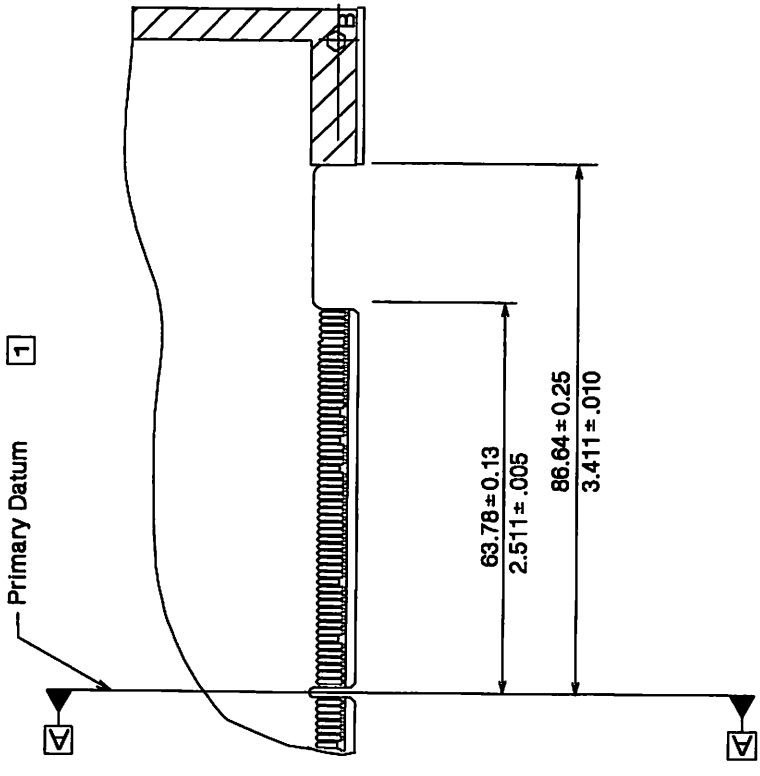


Figure 9 (Part 1 of 2). Connector Dimensions (Matched Memory Extension)

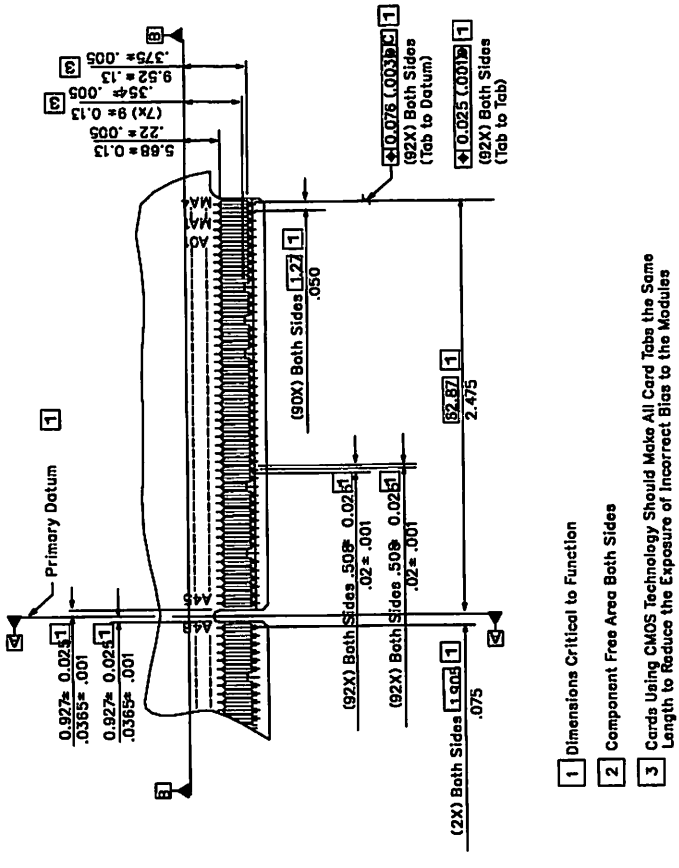


Figure 9 (Part 2 of 2). Connector Dimensions (Matched Memory Extension)

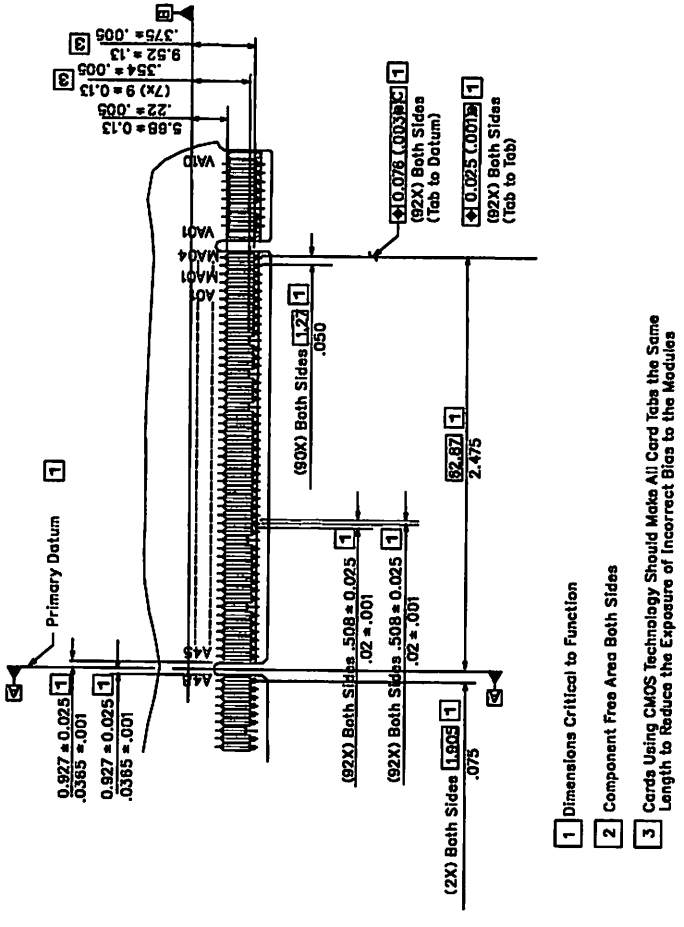


Figure 10 (Part 1 of 2). Connector Dimensions (Base Video Extension)

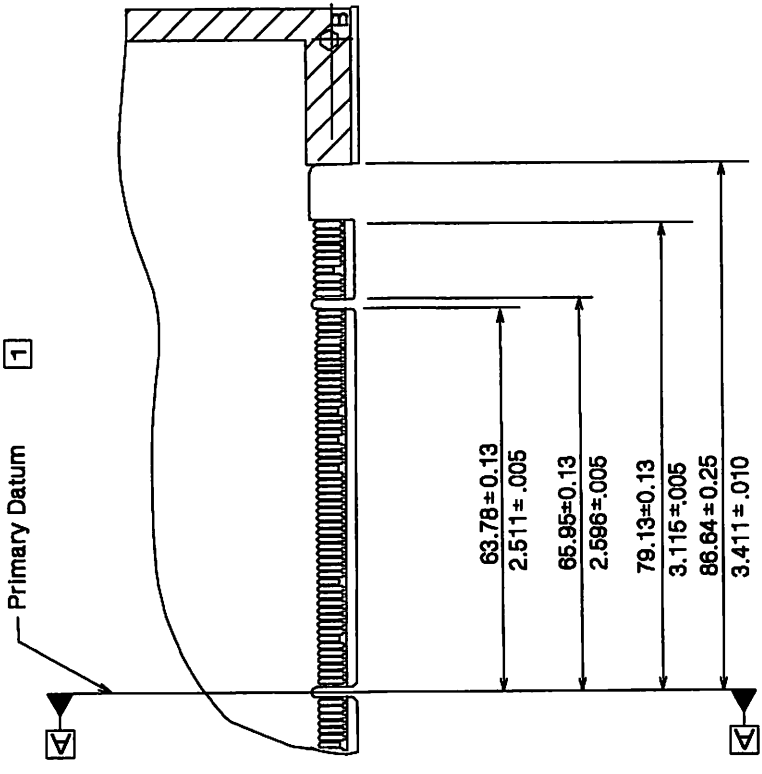


Figure 10 (Part 2 of 2). Connector Dimensions (Base Video Extension)

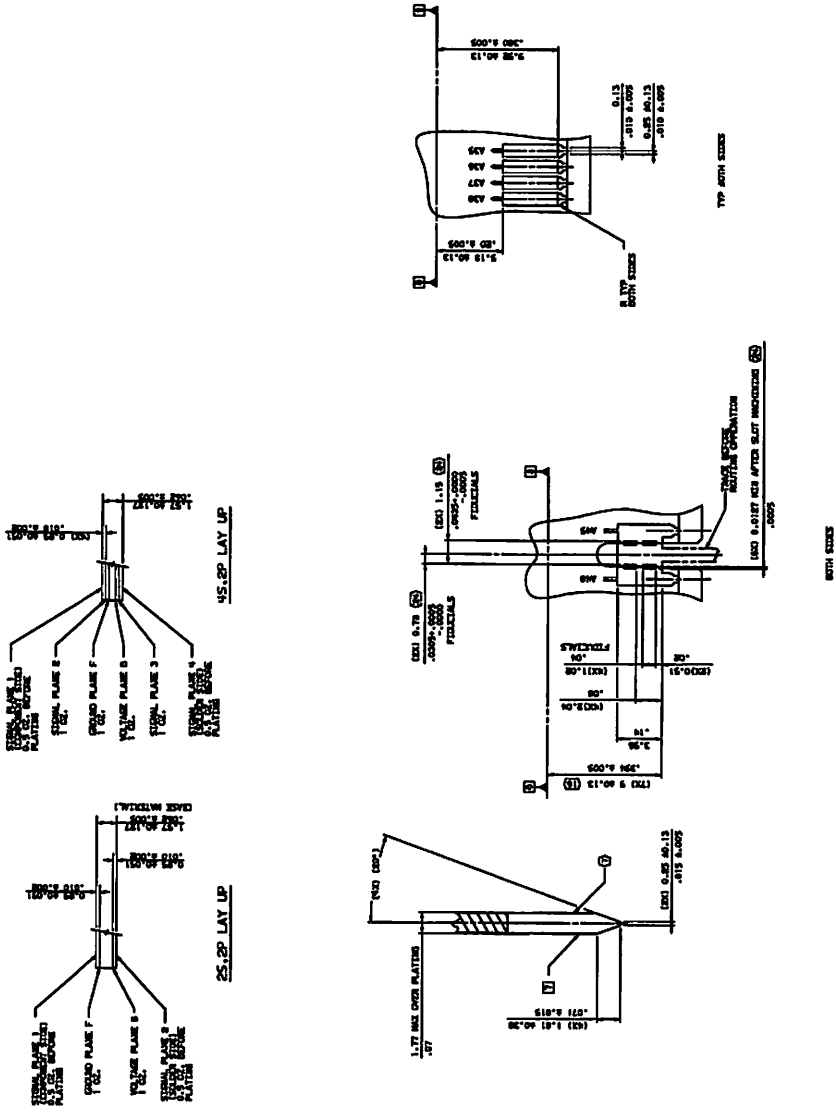
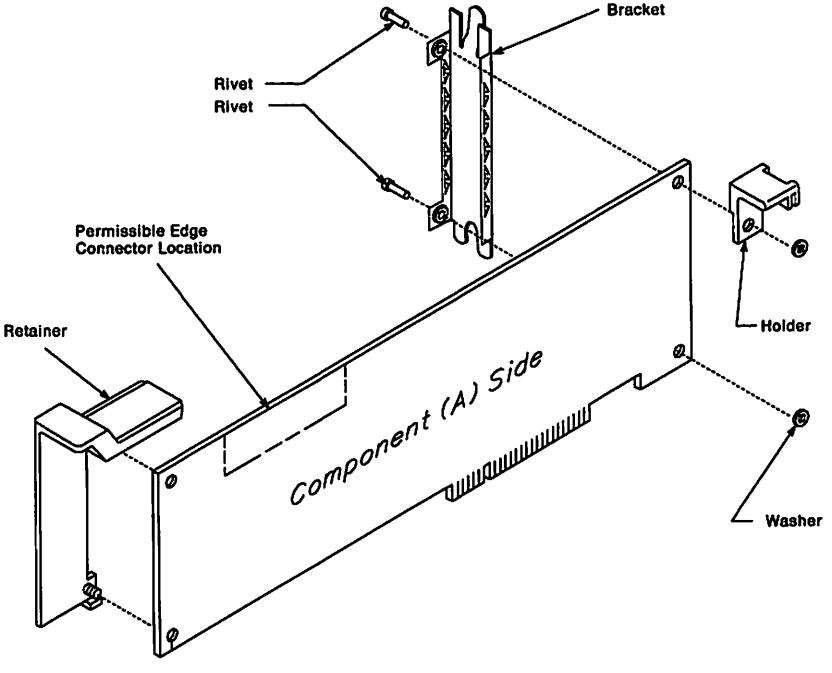


Figure 11. Connector Dimensions (Common Detail)



Materials:
Holder and Retainer - Polycarbonate UL 94 V-0
Bracket - AISI Type 302 1/4 Hard Stainless Steel

Figure 12. Typical Assembly for Type 3 Adapter

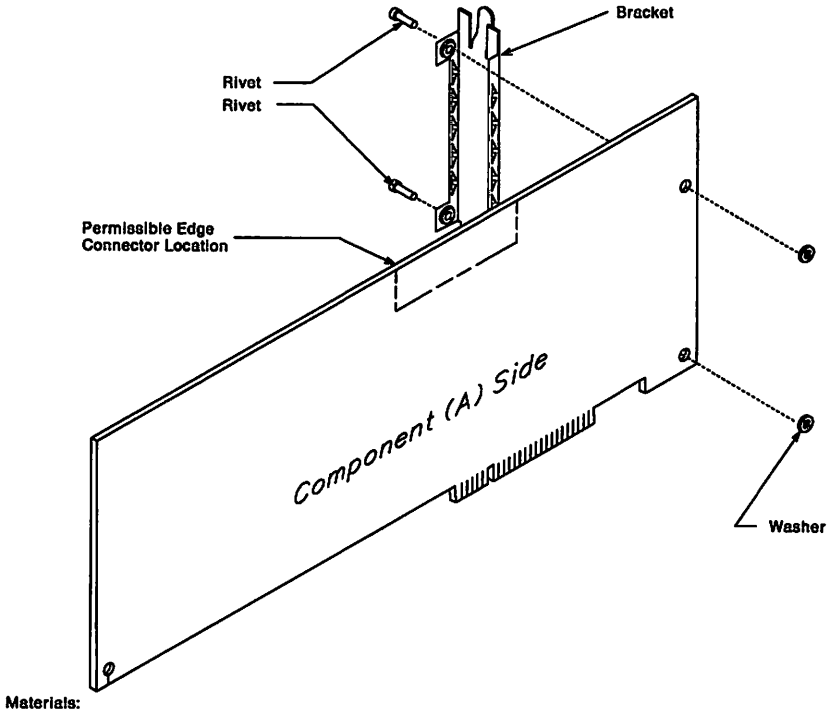


Figure 13. Typical Assembly for Type 5 Adapter

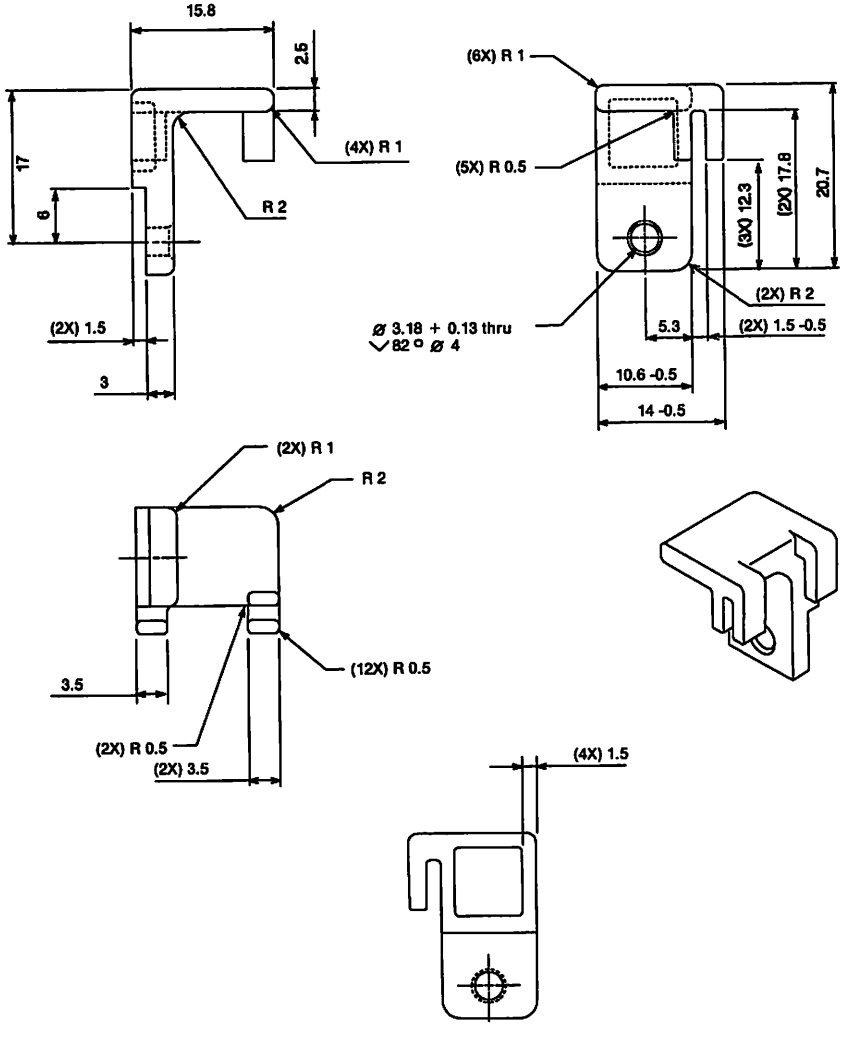


Figure 14. Adapter Holder

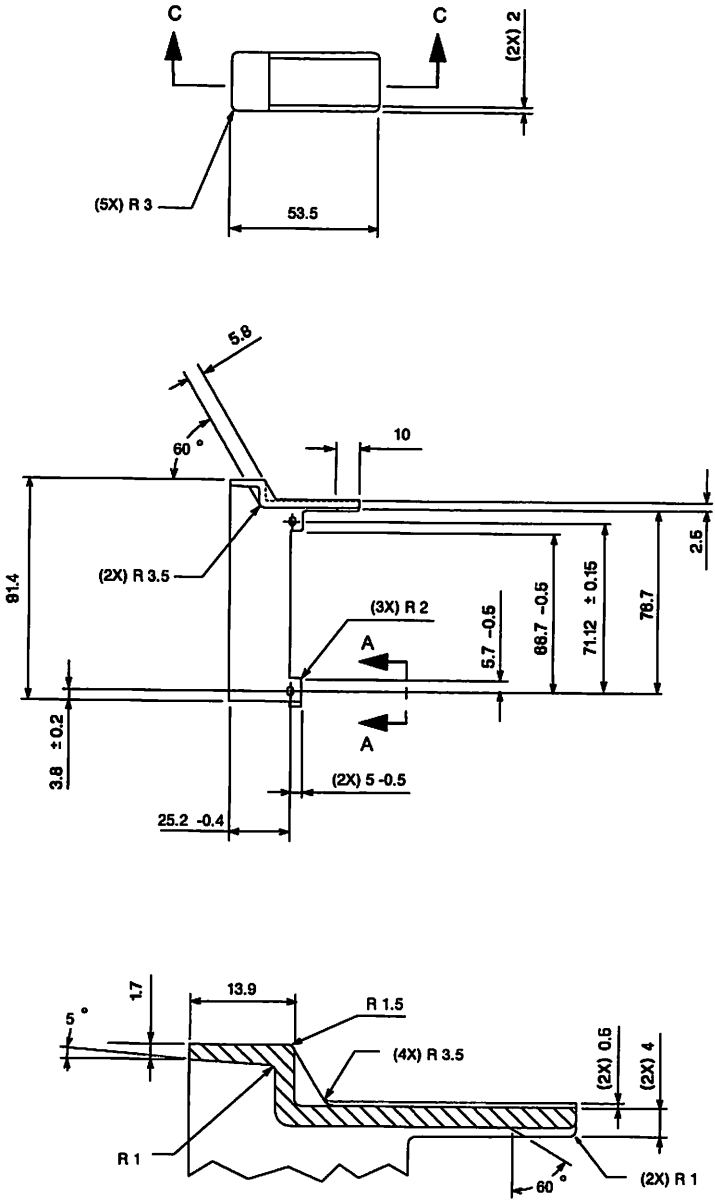


Figure 15 (Part 1 of 2). Adapter Retainer

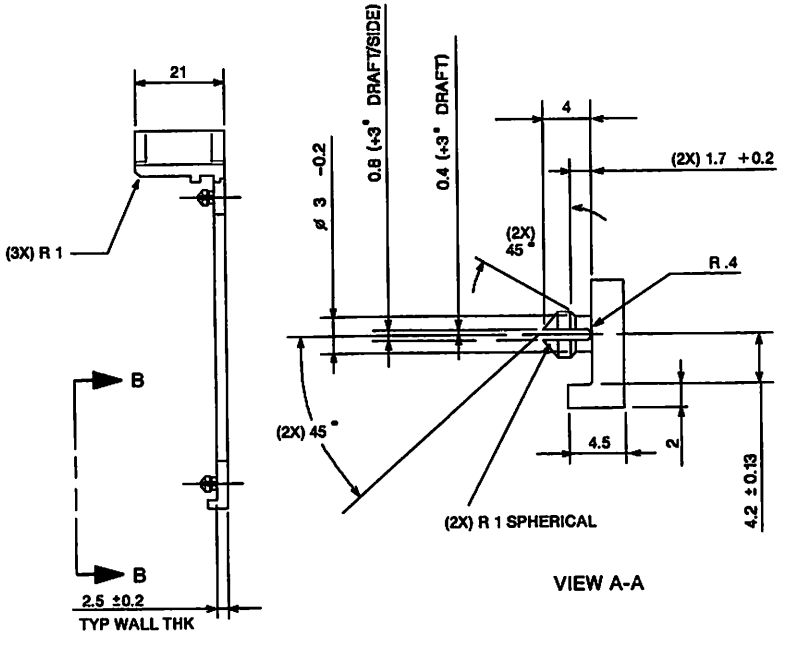
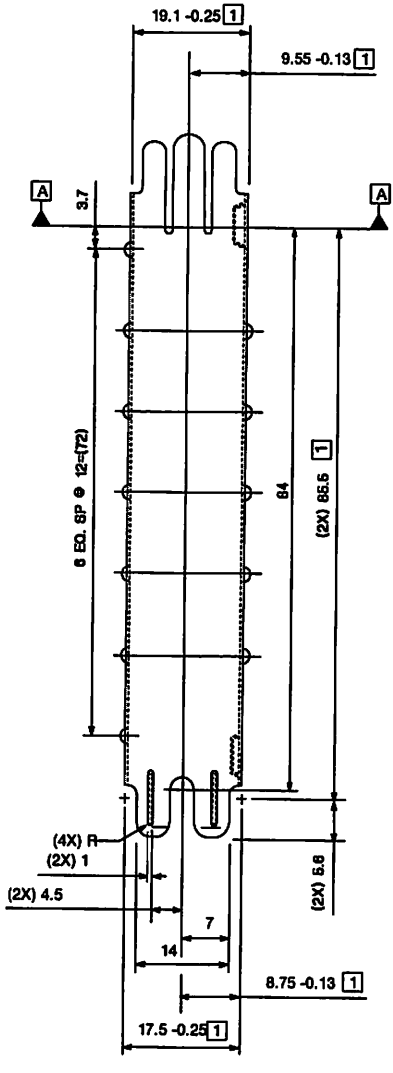


Figure 15 (Part 2 of 2). Adapter Retainer



[1] DIMENSION IS CRITICAL TO FUNCTION.

Figure 16 (Part 1 of 4). Adapter Bracket

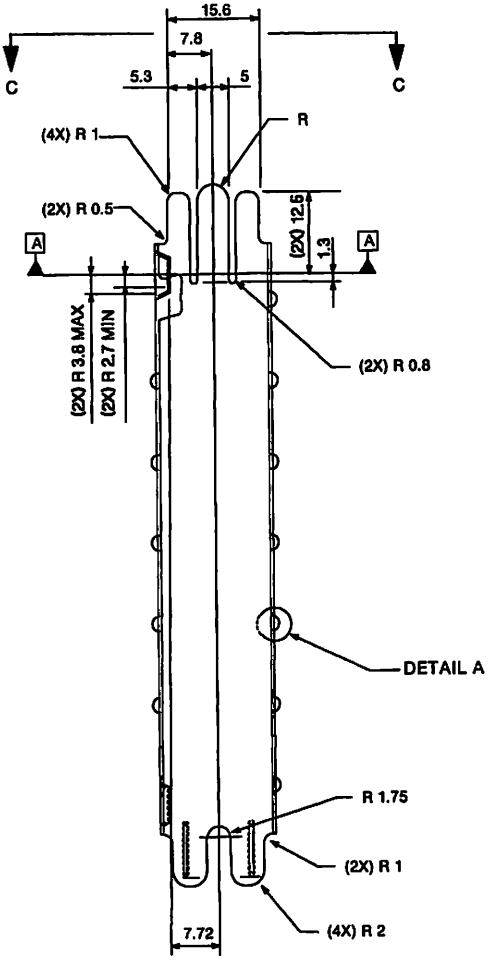


Figure 16 (Part 2 of 4). Adapter Bracket

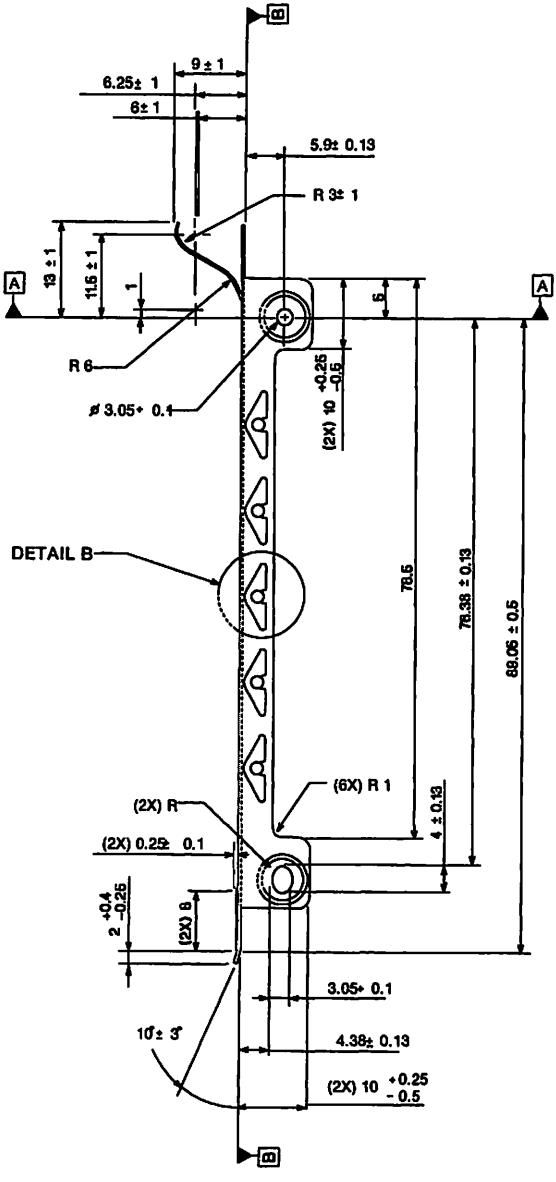
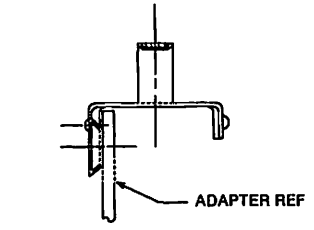
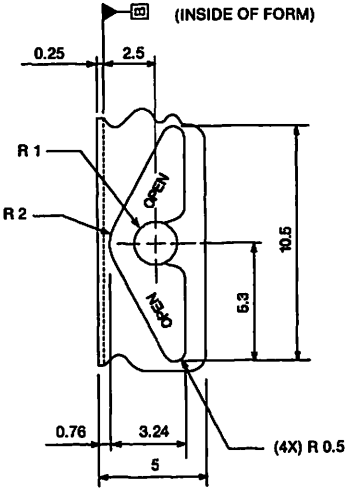


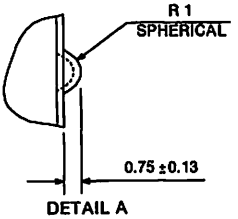
Figure 16 (Part 3 of 4). Adapter Bracket



VIEW C-C



DETAIL B
SCALE 10/1
(12X)



DETAIL A
SCALE 10/1
(12X)

Figure 16 (Part 4 of 4). Adapter Bracket

General Design Considerations

Each designer must take the precautions necessary to protect the safety of the end user, provide reliable operation of the device, and ensure the device does not interfere with the operation of the system or any other installed devices. The design considerations described in this section are not the **only** considerations, but rather those that might otherwise be overlooked.

Safety

Avoid exposed high-voltage or current points, sharp edges, and exposed components that operate at high temperatures. Devices must not channel dc power outside the system unit in any manner that violates Underwriters Laboratory and Canadian Standards Association guidelines.

Note: Canadian Standards Association C22.2, paragraph 4.11.3, number 154 requires protection of conductors of external interconnecting cords and cables connected to secondary circuits.

IBM does not support installing or removing Micro Channel adapters or components when the system power is on.

Thermal

The system unit is cooled internally by low-volume forced air. Adapter designs must allow for adequate air space between the adapters. Avoid using internal cables as a mechanism for signal communication inside the system unit, they can interfere with the air flow. If internal cables are required, they must be positioned to minimize the impact on airflow. The maximum height for components mounted on the adapter should not exceed the dimensions specified under "Dimensions" on page 3. The adapter design should avoid clustering of high-temperature components. No component should exceed its maximum thermal rating.

Electromagnetic Compatibility

Adhere to the following guidelines to reduce electromagnetic compatibility (EMC) problems.

- The adapter end brackets make a continuous 360° connection to the outside "skin" of the system unit cabinet. A similar 360° connection to the inside skin should also be provided. The adapter bracket must not be used as a dc voltage return path, a logic-ground connection, or an audio ground connection.
- The end bracket at the rear of the adapter is isolated from dc ground on the adapter. The bracket must be grounded through a screw connection to the system unit and designed as shown in Figure 16 on page 22.
- All connector ground pins must be connected to the interplane ground at the channel connector, and the +5 Vdc power must be immediately connected to the +5 Vdc power plane.
- All adapters must provide nonsegmented internal power and ground planes.
- Each surface-mount technology module position should provide a decoupling capacitor pad with minimal connection inductance. Pin-in-hole modules should be decoupled if they drive or contain edge-triggered logic. Capacitors can range between 0.01 and 0.10 microfarad and should be low-inductance ceramic or a layered design.
- Internal cables should be avoided as a mechanism for signal communication inside the system unit. The channel should not be extended outside the system unit, except by an adapter.
- Clocks should be properly imbedded and terminated. When clocks, strobes, and handshakes are generated or received, care should be taken to control the rise-and-fall times to minimize radiation.
- External cables should connect through 360° shielded D-shell or equivalent connectors. Avoid the use of "pigtail" shield connections. Shield terminations should be connected to the external shield of the cable connector. Do not bring the shield through the connector and connect it to either logic ground or the inside skin of the cabinet.
- High-current power within the system unit should provide adjacent return paths to allow the maximum cancelation of radiated magnetic fields by the mutual coupling between the supply and return lines.

Diagnostics

All writable registers typically are readable at the same address. External interfaces typically include 100% diagnostic wrap capability by electronic switching or an external wrap tool.

Design Guidelines

Adapters designed for the Micro Channel architecture must comply with the following design guidelines:

- Each I/O adapter must decode all 16 lines of the I/O address.
- Each 24-bit memory adapter must decode all 24 lines of the memory address and MADE 24.
- Each 32-bit memory adapter must decode all 32 lines of the memory address when MADE 24 is inactive. When MADE 24 is active, A24 through A31 are undefined.
- Each adapter must replace the function of switches and jumpers with registers that incorporate POS logic.
- Each adapter must respond with an *adapter ID* to the data bus when interrogated.

  The following figure shows the recommended ID values for vendors. ID values 8200 to FFFE are assigned for IBM products only.

  Note: Programs should not make decisions based on the high nibble of the Adapter ID groupings.

ID	Definition
0000	Device Not Ready
0001 to 0FFF	Bus Master
5000 to 5FFF	Direct Memory Access Devices
6000 to 6FFF	Direct Program Control (Including Memory-Mapped I/O)
7000 to 7FFF	Storage*
8000 to 81FF	Video
FFFF	Device Not Attached

* Multiple-function adapters containing storage typically respond as storage.

Figure 17. Vendor Adapter ID Assignments

The adapter IDs hex 7DB8 through 7DBF are reserved for the prototyping, developing, and testing of adapters and adapter designs and for adapters used in the testing of system designs.

Note: Do not use these IDs for purposes other than the local development of your adapters or systems.

- Each enabled adapter must return a '-card selected feedback' signal (-CD SFDBK) to the system microprocessor when an access is made to the address space of the adapter, or when the adapter is selected by arbitration level. -CD SFDBK must not be generated when the '-card setup' signal (-CD SETUP) is active.
- Each adapter design must be capable of degating all outputs to the system board (including -CD SFDBK, -DS CD 16/32, interrupts, and so on) if bit 0 of POS Register 2 is set to 0.
- Following a reset, each adapter must set bit 0 (Card Enable) of its POS Register 2 to 0.
- All adapters must provide adapter description files (.ADF) on a 3.5-inch diskette for system configuration.
- Adapter designs should not extend the card-edge connector beyond the basic 16- or 32-bit connector unless the signals provided by the extension are used by the adapter.
- A physical notch at positions 59 and 60 is required for 32-bit adapters that can function as a 16-bit adapter in a 16-bit connector. 32-bit adapters that can function in a 16-bit connector are usable in both 16 and 32-bit systems and have improved performance in 32-bit systems. The improved performance is a result of the increased data width and the ability to address over 16MB of memory.

Index

A
adapter bracket 22
adapter connector examples 2
adapter design 1
adapter design considerations 26
adapter diagnostics 28
adapter dimensions 3, 4
adapter holder 19
adapter retainer 20
assembly, typical adapter 17
assignments, recommended 28

B
bracket, adapter 22

C
circuit protection, secondary 26
CMOS technology 3
common detail, connector 16
compatibility, electromagnetic 27
component height 3
conductor protection 26
connector common detail 16
connector description 16
connector dimensions 8

D
design considerations 26
design guidelines 28
design, adapter 1
diagnostics 28
dimensions, adapter 3, 4
dimensions, connector 8
dimensions, video adapter 10, 14

E
electromagnetic compatibility 27
examples, adapter connectors 2

G
guidelines, design 28

H
height of components 3
holder, adapter 19

M
micro channel adapter design 1

P
protection of conductors 26

R
recommended assignments 28
retainer, adapter 20

S
safety 26
secondary circuit protection 26

T
technology, CMOS 3
thermal 26
typical adapter assembly 17

V
video adapter dimensions 10, 14

© Copyright IBM Corp. 1989, 1990

Setup

Reference Diskette 1
Option Diskette 2
System Configuration Utilities 2
Automatic Configuration Utility 3
Backtracking 4
Change Configuration Utility 5
View Configuration Utility 5
Backup and Restore Configuration Utilities 6
Copy an Option Diskette Utility 6
Adapter Description Files 7
Format 7
Syntax 7
Example 17
Adapter Description Program 20
Syntax for ADPs 21
System Memory ADPs 21
Nonsystem Memory ADPs 22
Program Call 23
ADP Functions 28
Configure the Adapter 28
Configure All Adapters 29
Deallocate Resources for an Adapter 29
Deallocate Resources for All Adapters 29
Deduce Adapter Configuration 29
Disable Block of System Memory 30
Remap System Memory to Above 16MB 31
Remap System Memory to Below 16MB 31
Configure System Memory on an Adapter 31
Configure System Memory on All Adapters 32
Deallocate System Memory on an Adapter 32
Deallocate System Memory on All Adapters 32
Deduce System Memory Configuration on an Adapter 32
Check Validity of Choice 33
Resource Query Functions 34
Request Memory Block 35
Check Memory Block 36
Deallocate Memory Block 37
Enable an Adapter 37
Disable an Adapter 38
Enable Memory by Prompt 38
Get Number of Slots 39
Get Slot Information 39

© Copyright IBM Corp. 1989, 1991

Error Codes 40
ADP Example 41
Initialization Programs 49

Index 51

Figures

1. Files with the ADF Extension 1
2. Syntax Symbol Key 7
3. Adapter Description File Syntax 8
4. ADP Parameter Block 24
5. ADP Functions 28
6. Check Validity of ADF Choice Function. 33
7. Resource Query Function Calls. 34
8. Resource Query Parameter Block 35
9. Request Memory Block Function 36
10. Check Memory Block Function 36
11. Deallocate Memory Block Function 37
12. Enable an Adapter Function 37
13. Disable an Adapter Function 38
14. Enable Memory by Prompt Function 38
15. Get Number of Slots Function 39
16. Get Slot Information Function 39
17. Error Codes 40

© Copyright IBM Corp. 1989, 1991

Notes:

The integrated I/O functions of the system board use POS information during the setup procedure. The bit assignments and functions may vary from system to system (refer to the system board setup information in the system-specific technical reference for the system you are dealing with). The setup routine (Automatic Configuration) obtains adapter information from adapter description files and programs.

Reference Diskette

Each system has a Reference Diskette containing the System Configuration and diagnostic utilities. The System Configuration utilities identify the installed hardware and interpret the system resources (I/O ports, memory, interrupt levels, and arbitration levels) for each device. The diagnostic utilities perform customer and service testing on the system, its components, and adapters made by IBM.

All the files on the Reference Diskette must be copied onto a backup copy, and the backup copy is used for configuring the system and running diagnostic routines. As options are added, the files needed for configuration and problem determination of the new options are merged onto this backup copy.

The following table shows the naming convention for files used by adapters during configuration and the maximum size of each file type. The CardID is the POS adapter ID in hex.

File Name	Description	Maximum Size	
@CardID.ADF	Adapter Description File	Required	--
CCardID.ADF	Adapter Description Program	Optional	10KB
ICardID.ADF	Initialization Program	Optional	1KB
Note: The total size for all files required to configure an adapter should not exceed 35KB.			

Figure 1. Files with the ADF Extension

Option Diskette

The Option Diskette supplied with the adapter should contain the adapter description file (ADF) and, as needed by the adapter, the adapter description program (ADP) and the initialization program.

Before a new adapter is installed, its ADF and other associated files are merged onto a backup copy of the Reference Diskette. These files are merged by selecting the Copy an Option Diskette program from the Main Menu and following the instructions on the screen.

Additional files may be required on the Option Diskette to support keywords or functions used in the ADF or ADP.

Note: Developers of adapters and their customers are granted permission to use, but not to modify, the Set Configuration program to support their adapters on IBM PS/2 systems.

System Configuration Utilities

The System Configuration utilities are contained within the Set Configuration program. This program enables the user to configure the system in one of two ways:

- Automatically after a configuration error is displayed
- Manually by selecting Set Configuration from the Main Menu.

The Set Configuration program uses information contained in the adapter description file (ADF) to track and allocate system resources. Each ADF describes the resources that can be allocated to a specific adapter and the POS setting used to indicate those resource assignments. When more than one device is configured to the same resource and that resource cannot be shared, only one of the conflicting devices is enabled.

Each adapter contains a 16-bit adapter ID and upto four 8-bit POS registers. The Set Configuration program initially sets all bits in the POS registers to 0. It then uses the ADF and ADP to set the contents of the POS registers to the apppropriate values.

Adapter IDs and the POS information are stored in CMOS or nonvolatile RAM by the Set Configuration program. The Set Configuration program determines the system type and handles

differences between systems such as the nonvolatile RAM storage and the number of adapter slots available.

Automatic Configuration Utility

Automatic Configuration can be run after a configuration error has occurred or by selecting Run Automatic Configuration from the Set Configuration menu. Each time the system is powered on, POST compares the configuration of the system to the configuration indicated by nonvolatile RAM. If differences between the two are detected, an error is displayed and logged in system RAM. If the Reference Diskette is installed, an expanded message is displayed that further explains the POST error.

If a configuration error is caused by a loss of configuration data, user is given a choice to either run Automatic Configuration or restore configuration after the POST error message is displayed. If the error is caused by a change to the configuration, the user may be given a choice to either run Automatic Configuration or continue to the Main Menu. If the user continues with the Main Menu, the changed areas of the system are configured and nonvolatile RAM is updated when Set Configuration is selected from the Main Menu.

Depending on the cause of the error, Automatic Configuration either reconfigures the entire system or configures only the areas of the system that have been changed since the last time a configuration was performed. The following POST errors cause the system to be completely reconfigured:

- 161 (battery failure)
- 162 (CRC error in nonvolatile RAM)
- 169 (processor complex error)
- 173 (CRC error in nonvolatile RAM).

The following POST errors cause only the areas of the system that have changed to be reconfigured:

- 162 (system configuration error not caused by a CRC error)
- 164 (memory configuration)
- 165 (adapter configuration)
- 174 (adapter configuration error caused by attached devices).

An adapter is considered previously configured when the POS data stored in nonvolatile RAM matches a set of POS settings in the appropriate ADF. If an ADF for an installed adapter cannot be found, the user is prompted to insert the Option Diskette. If system configuration is continued without its ADF being found, the adapter is not configured and is not enabled.

During Automatic Configuration, devices are configured to the first nonconflicting values defined in the ADF. Adapters are configured in the order of their channel position. The system board is configured first, followed by each adapter slot starting with slot 1. Interrupt levels are a shared resource and are not considered to be in conflict when more than one device is assigned the same level. If a named item in the ADF contains a list of interrupt levels to choose from, the Automatic Configuration program assigns the level that is assigned to the fewest devices at that time.

Automatic Configuration will backtrack to previously configured adapters to resolve conflicts in nonsystem memory resources. If Automatic Configuration cannot resolve a conflict, the conflict must be resolved by choosing a nonconflicting resource option through the Change Configuration utility. Any adapter having a resource conflict that cannot be resolved by the Set Configuration program is disabled; the program sets bit 0 of POS Register 2 (hex 0102) to 0 in nonvolatile RAM.

Backtracking

The Set Configuration program *does not* backtrack to resolve conflicts involving the allocation of:

- I/O address space
- Arbitration levels.

The Set Configuration program *does* backtrack if an adapter cannot map its memory into the hex 0C0000 to 0DFFFF space or if an adapter with nonsystem memory cannot find an available block of memory address space. The backtracking process involves changing the order in which the devices with memory requirements are configured. The different permutations are checked until all memory devices are properly configured or all permutations have been checked.

| If an adapter is requesting nonsystem memory below 16MB and conflicts still exist, Set Configuration performs the following steps until the conflict is resolved. (The steps performed depend on the operations supported by the system.)

1. The split memory on the system board is disabled.
2. Blocks of system-board memory are remapped to above 16MB.
3. Blocks of system-board memory are disabled if remapping is not supported.

Change Configuration Utility

The Change Configuration utility allows the user to change the current configuration settings to resolve unusual conflicts or to customize configuration.

Resource conflicts are indicated by an asterisk (*) next to the conflicting items and also by the "* Conflicts" string in the upper right corner of the Change Configuration window. Conflicts with fixed resources have the asterisk (*) to the left of the slot number of the adapter. Adapters with conflicts are disabled; the program sets bit 0 of POS Register 2 (hex 0102) to 0.

The user can scroll or page through the various screens. Changes are made by rotating field value names through a set of choices using the F5 (Previous) and F6 (Next) keys. Changes are not saved in nonvolatile RAM until the F10 (Save) key is pressed. Help text is provided for each item by pressing the F1 (Help) key.

Note: Automatic Configuration will restore all customized settings to the first setting listed in the ADF that does not conflict.

View Configuration Utility

The View Configuration utility is provided to view the configuration. This is the Change Configuration utility with the change capabilities disabled.

Backup and Restore Configuration Utilities

The Backup Configuration utility provides a means to back up the configuration data to a file on the Reference Diskette. If the battery fails or the battery is changed, the user can use the Restore utility to restore the configuration.

Note: A copy of the Reference Diskette that is not write protected is needed for this backup and restore process.

Copy an Option Diskette Utility

The Copy an Option Diskette utility is a separate program from the Set Configuration program and is accessible through the Main Menu. This utility is used to merge the files with the extension of ADF[1] from an Option Diskette onto a backup copy of the Reference Diskette. The files found on this diskette are compared to the files on the backup copy of the Reference Diskette. If the file does not exist on the Reference Diskette, it is copied. If the file already exists on the Reference Diskette, the dates of the two files are compared, and the file is copied if it has a date later than, or the same as, the corresponding file on the backup diskette.

The Option Diskette must be a DOS formatted diskette.

---

1 The following files are also copied by this utility; however, the file names are reserved for use by IBM: *.DGS, *.PEP, COMMAND.COM, DIAGS.COM, CMD.COM, SC.EXE, UPDATE.EXE, and SYSCONF.

Adapter Description Files

Adapter description files (ADFs) provide POS information and system resource information for system configuration. The ADFs also provide text for the System Configuration utilities, help screens, and prompts. This section provides guidelines for developing the adapter description files.

Format

- File names: @CardID.ADF (high byte of the adapter ID first).
- Type of file: ASCII text.
- Not case sensitive: Key words can be lowercase, uppercase, or mixed. The case is preserved within the user interface text strings.
- Blanks, tabs, new lines: These are considered as white space and ignored, except when in text strings for the Change Configuration user interface.
- Comments: Lines beginning with semicolons are comments and are ignored.

Syntax

The following table shows the meaning of special symbols used in the adapter description file syntax. Values shown as hexadecimal can be specified as decimal or hexadecimal using the Intel format. Values shown as binary must be specified as binary.

Symbol	Meaning
{ }	an optional item
{ }*	0,1,2,... items allowed
{ }+	1,2,3,... items allowed
x | y	either x or y allowed
x{n}	n x's required
[0-9]+	one or more decimal digits

Figure 2. Syntax Symbol Key

```
adf_file => card_id card_name nbytes {fixed_resources}
            {system_memory} {initial_program} {lock_enable}
            {address_width} {programmed_io} {items}*
```

This statement defines the contents of an ADF. The following definitions describe each parameter in detail.

```
card_id => AdapterId number
```

Each ADF must contain the card_id parameter. The character string AdapterId is a keyword and must be present. The Set Configuration program looks for this ID, which matches the ID used in the filename.
Example: `AdapterId 0DEFFh`

```
card_name => AdapterName string
```

Each ADF must contain the card_name parameter. The character string AdapterName is a keyword and must be present.
The string following the AdapterName keyword is displayed as the adapter name in the Change Configuration and View Configuration screens. The length of the AdapterName string is limited to (71 - (length of 'Planar Device')) characters.
(The length for US English is 57 characters)
Example: `AdapterName "IBM Multiprotocol Communications Adapter"`

```
nbytes => NumBytes number
```

Each ADF must contain the nbytes parameter; it defines the number of POS bytes used by the adapter. The character string NumBytes is a keyword. The number of POS bytes used includes all bytes from POS [0] to the last POS byte used. (If POS [3] is the only POS byte defined, NumBytes should be set to 4.)
Example: `NumBytes 4`

```
fixed_resources => FixedResources pos_setting resource_setting
```

The fixed_resources parameter is optional, and it defines those resources required by an adapter and the corresponding POS data. The character string FixedResources is a keyword and must be present if the adapter has a specific system resource that it must use, such as a hard-wired I/O address or interrupt level.
Note:
Using fixed resources is not recommended because conflicts cannot be resolved automatically and can result in the adapter being disabled.
Example: `FixedResources  POS[1]=XXXX01XXb int 3`

Figure 3 (Part 1 of 9). Adapter Description File Syntax

```
system_memory => SysMem granularity {address_size}
```

The system_memory parameter is required for adapters that use an ADP to configure their system memory. The character string SysMem is the keyword.
The granularity field is required and specifies the smallest block of memory on the adapter that can be disabled. This block size must be divisible by 4KB. The value used in this field specifies the number of 1KB blocks.
The address_size field specifies the number of address bits supported by the adapter, 24 or 32 bits. The default size is 24.
Note:
This keyword requires an ADP.
Example: `SysMem 1024 32`

```
initial_program => InitProg attribute
```

The initial_program parameter is required for adapters that have an initialization program associated with them. The character string InitProg is the keyword. The initialization programs are loaded onto the fixed disk and executed before the operating system is booted from the fixed disk.
The attribute is a number from 0 to 127 and determines the order in which the initialization programs are run. The lower numbers are run first. Values outside the range from 30 to 80 are reserved.
Example: `InitProg 30`
(See "Initialization Programs" on page 49 for more information.)

```
lock_enable => Lock
```

The lock_enable parameter is optional. The character string Lock is the keyword, and its presence enables the Lock function.
An adapter that wishes to communicate with other masters through a memory semaphore protocol (an indivisible test and set operation, such as the Exchange instruction) must indicate this to the Set Configuration program by the Lock keyword.
Example: `Lock`

```
items => {{named_item | input_item} | {begin_device}}*
```

The items parameter is optional and can be any combination of the named_item and input_item parameters, or it can be any number of begin_device parameters.

Figure 3 (Part 2 of 9). Adapter Description File Syntax

named_item => NamedItem prompt {named_choice}+ help

The named_item parameter is optional, and it defines a field that provides a choice of one or more resources. Each Choice field sets specified POS bits to a unique setting used to identify resources assigned to the adapter. The character string NamedItem is a keyword and must be present if the adapter can be configured to use different resources. The adapter determines the resources to which it is configured by how the POS bytes are set. When a NamedItem is defined, it must be accompanied by a prompt, at least one named_choice, and a help field.
To provide flexibility during configuration, each named_item should contain more than one choice.
Note:
Each ADF is limited to 10 named_items per adapter.
Example:

```
NamedItem
Prompt "Communications Port"
choice "SDLC_1"   pos[0]=XXX1000Xb  io 0380h-038ch  int 3 4
choice "SDLC_2"   pos[0]=XXX1001Xb  io 03a0h-03ach  int 3 4
choice "BISYNC_1" pos[0]=XXX1100Xb  io 0380h-0389h  int 3 4
choice "BISYNC_2" pos[0]=XXX1101Xb  io 03a0h-03a9h  int 3 4
choice "SERIAL_1" pos[0]=XXX0000Xb  io 03f8h-03ffh  int 4
choice "SERIAL_2" pos[0]=XXX0001Xb  io 02f8h-02ffh  int 3
choice "SERIAL_3" pos[0]=XXX0010Xb  io 3220h-3227h  int 3
choice "SERIAL_4" pos[0]=XXX0011Xb  io 3228h-322fh  int 3
choice "SERIAL_5" pos[0]=XXX0100Xb  io 4220h-4227h  int 3
choice "SERIAL_6" pos[0]=XXX0101Xb  io 4228h-422fh  int 3
choice "SERIAL_7" pos[0]=XXX0110Xb  io 5220h-5227h  int 3
choice "SERIAL_8" pos[0]=XXX0111Xb  io 5228h-522fh  int 3
Help
"This port can be assigned as the primary (SDLC1) or
secondary (SDLC2) sdlc, the primary (BISYNC1) or secondary
(BISYNC2) bisync, or as a serial port (Serial 1 through Serial
8).  Use the F5=Previous and the F6=Next keys to change this
assignment in the 'Change configuration' window.  Conflicting
assignments are marked with an asterisk and must be changed
to use the adapter."
```

Figure 3 (Part 3 of 9). Adapter Description File Syntax

```
input_item => Input prompt pos_byte_setting range help
```

The input_item parameter is optional; it defines a field that accepts operator-entered data to initialize a single POS byte, or to define specific bits within that byte. The character string Input is the keyword. The operator can enter up to a three-digit positive integer. The bits to be altered by the input are defined using 0 or 1. The X specifies bits that remain unchanged by the input. The operator input is aligned with the least-significant bit of the pos_byte_setting field not defined as X. The two values are then ORed before the mask is applied and the result is stored. The range defines the limit of the values acceptable as user input. The first number listed in the range will be used as the input for the adapter during automatic configuration.

Example:
```
Input prompt "Enter value"pos[0]=XXXX1111b 0 - 15
      help "Enter a value from 0 - 15."
```

```
prompt => Prompt string
```

The prompt is required when a named_item or input_item is defined. The prompt is used to define a title for a NamedItem or an Input field. The character string Prompt is a keyword and must be present when a NamedItem or Input parameter is present. The string following the Prompt keyword appears after the adapter name in the Change Configuration and View Configuration screens.

Following the prompt string is a field that can be toggled in the Change Configuration screen if two or more named_choices are defined. The length of the prompt cannot exceed 38 characters.

Example: (See the example for named_item).

```
named_choice => Choice {valid_check} choice_name pos_setting
      resource_setting
```

At least one named_choice is required when a NamedItem is defined; however, more than choice should be defined to allow for flexibility during configuration. The character string Choice is a keyword and must be present when a NamedItem is present. The field is specified after the prompt.

Each choice must contain the choice_name, which describes the current choice. The choice_name string must not exceed 28 characters.

Each choice must define a pos_setting for at least one POS byte that uniquely identifies the resource_setting defined in the named_choice.

Example:
```
Choice "IO 3F80" pos[1]=00111111b pos[2]=10000000b
      io 3f80-3f87
```

Figure 3 (Part 4 of 9). Adapter Description File Syntax

```
valid_check => Vcheck
```

The valid_check parameter invokes the ADP to check the specified choice for validity. The character string Vcheck is the keyword. The ADP ensures that the choice is valid for the system configuration. If the Set Configuration program is unable to load the ADP, the choice is treated as invalid. If all the choices within a named_item are invalid, the Set Configuration program deletes the named_item from the list of items for that adapter.

```
help => Help string
```

The help field is a string of text used to give the user assistance at a prompt. This text is displayed in the Change Configuration and View Configuration screens when the cursor is at the associated prompt and the F1 key is pressed. The character string Help is a keyword and must be present when the NamedItem or Input parameter is present. The string following the Help keyword is the text that describes the prompt. The length of the help string is limited to 1000 characters.
Note:
Messages should indicate hexidecimal values with an "h" at the end of the number.
Example: (See the example of named_item.)

```
pos_setting => {pos_byte_setting}+
```

The pos_setting parameter must contain at least one pos_byte_setting. See the definition of pos_byte_setting.

```
pos_byte_setting => POS[number]=pos_bit{8}b
```

Each pos_byte_setting field defines one POS byte.
The character string POS is a keyword and precedes each POS byte defined. The number of the POS byte is specified in brackets and refers to the POS bytes as shown below:

number = 0, POS byte 0 at port 102h
number = 1, POS byte 1 at port 103h
number = 2, POS byte 2 at port 104h
number = 3, POS byte 3 at port 105h

The closing bracket must be followed by an equal sign, and then a bit definition of the POS byte (See pos_bit for more). The bit definition must define all 8 bits of the byte. Bit 0 of pos[0] and bit 7 of pos[3] should always be defined as X.
Example: `pos[0]=XXX1001Xb`

```
pos_bit => x | X | 0 | 1
```

A pos_bit can be defined as a mask bit (x or X), a clear bit (0), or a set bit (1).
Example: `pos[0]=XXX1001Xb`

Figure 3 (Part 5 of 9). Adapter Description File Syntax

```
resource_setting => {ioblock_list} {interrupt_list} {arb_list}
                    {memaddr_list} {exec_adp}
```

The resource_setting parameter defines a list of system resources. The resources can be fixed resources required by the adapter, or they can be resources the adapter uses when configured to a specific choice in a named_item. The resources can consist of the following:

- Range of I/O addresses (limited to 16 ranges).
- List of interrupt levels (limited to 16).
- List of arbitration levels (limited to 16).
- Range of memory addresses (limited to 2 ranges).
- An ADP call for nonsystem memory allocation.

Example: (See the specific resource definition).

```
ioblock_list => IO {range}+
```

The ioblock_list parameter is a list of one or more ranges of I/O addresses. The character string IO is a keyword and must be present in the ioblock_list.
Example: `IO 4220h-4227h`

```
interrupt_list => Int {number}+
```

The interrupt_list parameter is a list of one or more interrupt levels that are used concurrently. The character string Int is a keyword and must be present in the interrupt_list.
Example: `Int 3 4`

```
arb_list => Arb {shared_arb}{number}+
```

The arb_list parameter is a list of one or more arbitration levels that are used concurrently. The character string Arb is a keyword and must be present in the arb_list.
Example: `Arb 1`

```
shared_arb => SharedArb
```

The character string SharedArb is the keyword and indicates that, in a controlled environment, the arbitration level in the named_item parameter can be shared with the arbitration level of other devices that support sharing of arbitration levels.

Example: `Arb SharedArb 1`

Figure 3 (Part 6 of 9). Adapter Description File Syntax

```
memaddr_list => Mem {range}+
```

The memaddr_list parameter is a list of one or more ranges of RAM or ROM addresses. The character string Mem is a keyword and must be present in the memaddr_list. This keyword is used to allocate the memory address space in the hex C00C0000 through hex C00DFFFF or the nonsystem memory above 1MB.
Include the address for each Choice in the choice_name field so that the address is also displayed during Change Configuration.
Note:
Because of the limited space available, each adapter should limit the address space used in segments 0C000 and 0D000 to 16KB.
Example: `Mem 0CC000h - 0CDFFFh 0E00000h - 0EFFFFFh`

```
exec_adp => Exec
```

The exec_adp parameter is required for adapters containing nonsystem memory that cannot be allocated by an ADF. The character string Exec is the keyword that defines the ADP and should be the last Choice in the list of named_choices.
The Exec keyword indicates to the Set Configuration program that the adapter has an ADP.
Note:
This keyword requires an ADP.
See the example on page 22.

```
address_width => Address 24|32
```

The address_width field specifies the number of address bits supported by the adapter (24 or 32 bits). The character string Address is the keyword and the default size is 24.
Example: `Address 32`

```
programmed_I/O => PIO
```

The programmed_I/O parameter is used to specify whether an adapter includes address bits 24 through 31 in its address decode. The character string PIO is the keyword. When the keyword is specified, the adapter does not decode these address bits. When not specified, the adapter does decode them.
Note:
The PIO keyword is not required if the connector is limited to 24-bits of address.
Example: `PIO`

Figure 3 (Part 7 of 9). Adapter Description File Syntax

```
begin_device =>
  Begin
  {device} {rs422} {DMA_not_used}
  {named_item |input_item}*
  End
```

The begin_device parameter is optional and is used to define a device block on an adapter. A device block is a series of items that define POS settings for like devices on an adapter, but are not associated with any other devices that might be on the adapter.

```
device =>  Device device_id {implement_id {revision_level}}
```

The device parameter is optional and defines the device types used for the individual devices on an adapter or the system board. The character string Device is the keyword.

The device_id parameter is the device identifier, the implement_id parameter is the implementation identifier, and the revision_level parameter is the implementation revision level. The implementation identifier and revision level are optional and default to 0 if not specified; however, the revision level cannot be specified without the implementation identifier.
Example: Device 2 1

```
rs422 => RS422
```

The rs422 parameter is optional and indicates that the serial device uses a RS422-type interface. The character string RS422 is the keyword.
Example: RS422

```
DMA_not_used => NoDMA
```

The DMA_not_used parameter indicates that arbitration levels for the named_items within the begin_device do not require a DMA channel. The character string NoDMA is the keyword. If this parameter is specified, the Set Configuration program does not indicate that a DMA channel is required when an arbitration level is used. If this parameter is not specified, the Set Configuration program indicates that a DMA channel is required when an arbitration level is used.
Note:
Bus masters should use the NoDMA keyword.

Figure 3 (Part 8 of 9). Adapter Description File Syntax

```
range => number - number

number => [0-9]+ {d} |  [0-9a-f]+ h |  [0-9A-F]+ H

string => " [ascii except for "]+ "
```

The string field is a set of ASCII characters beginning with a double quote (") and ending with a double quote.
Example:

```
"This port can be assigned as:  primary (SDLC1) or
secondary (SDLC2) sdlc port, primary (BISYNC1) or secondary
(BISYNC2) bisync port, or as a serial port (Serial 1 through
Serial 8).  Use the F5=Previous and the F6=Next keys to change
this assignment in the 'Change configuration' window.
Conflicting assignments are marked with an asterisk and must
be changed to use the adapter."
```

Figure 3 (Part 9 of 9). Adapter Description File Syntax

Example

The following is an example of an adapter description file for the IBM Personal System/2 Multiprotocol Communications Adapter/A. The name of the file for this adapter is @DEFF.ADF. An explanation of each numbered item begins on page 18.

```
AdapterId 0DEFFh 1

AdapterName "IBM Multiprotocol Communications Adapter" 2

NumBytes 2 3

NamedItem 4

Prompt "Communications Port"

choice "SDLC_1"   pos[0]=XXX1000Xb  io 0380h-038ch  int 3 4
choice "SDLC_2"   pos[0]=XXX1001Xb  io 03a0h-03ach  int 3 4
choice "BISYNC_1" pos[0]=XXX1100Xb  io 0380h-0389h  int 3 4
choice "BISYNC_2" pos[0]=XXX1101Xb  io 03a0h-03a9h  int 3 4
choice "SERIAL_1" pos[0]=XXX0000Xb  io 03f8h-03ffh  int 4
choice "SERIAL_2" pos[0]=XXX0001Xb  io 02f8h-02ffh  int 3
choice "SERIAL_3" pos[0]=XXX0010Xb  io 3220h-3227h  int 3
choice "SERIAL_4" pos[0]=XXX0011Xb  io 3228h-322fh  int 3
choice "SERIAL_5" pos[0]=XXX0100Xb  io 4220h-4227h  int 3
choice "SERIAL_6" pos[0]=XXX0101Xb  io 4228h-422fh  int 3
choice "SERIAL_7" pos[0]=XXX0110Xb  io 5220h-5227h  int 3
choice "SERIAL_8" pos[0]=XXX0111Xb  io 5228h-522fh  int 3

Help

"This port can be assigned as the primary (SDLC1) or
secondary (SDLC2) sdlc port, the primary (BISYNC1) or secondary
(BISYNC2) bisync port, or as a serial port (Serial 1 through
Serial 8).  Use the F5=Previous and the F6=Next keys to
change this assignment.  Conflicting assignments are marked
with an asterisk and must be changed."
```

```
NamedItem 5
Prompt "Arbitration Level for SDLC"

choice "Level_1"   pos[1]=XXXX0001b  arb  1
choice "Level_0"   pos[1]=XXXX0000b  arb  0
choice "Level_2"   pos[1]=XXXX0010b  arb  2
choice "Level_3"   pos[1]=XXXX0011b  arb  3
choice "Level_4"   pos[1]=XXXX0100b  arb  4
choice "Level_5"   pos[1]=XXXX0101b  arb  5
choice "Level_6"   pos[1]=XXXX0110b  arb  6
choice "Level_7"   pos[1]=XXXX0111b  arb  7
choice "Level_8"   pos[1]=XXXX1000b  arb  8
choice "Level_9"   pos[1]=XXXX1001b  arb  9
choice "Level_10"  pos[1]=XXXX1010b  arb 10
choice "Level_11"  pos[1]=XXXX1011b  arb 11
choice "Level_12"  pos[1]=XXXX1100b  arb 12
choice "Level_13"  pos[1]=XXXX1101b  arb 13
choice "Level_14"  pos[1]=XXXX1110b  arb 14

Help

"This assignment need only be changed if it is in conflict
with another assignment.  Conflicting assignments are marked
with an asterisk.  Use the F5=Previous and the F6=Next keys
to change arbitration level assignments.  Using arbitration
levels, this adapter accesses memory directly without
burdening the computer's main microprocessor.  An
arbitration level of 0 has the highest priority, and
increasing levels have corresponding decreased priority"
```

1 The card_id for this adapter is hex 0DEFF. This is an ASCII representation of the ID generated by the adapter. The high byte is followed by the low byte. The card_id is required for all ADFs.

2 The card_name is "IBM Multiprotocol Communications Adapter." The card_name is required for all ADFs.

3 The nbytes value in this file is 2, which indicates the adapter uses the POS byte located at hex 0103 and may use the POS byte at hex 0102.

4 This is the first named_item for the adapter. The title of the field is "Communications Port." The user can toggle between the 12 named_choices. Each named_choice has a unique pos_setting assigned to it in bit locations 1 through 4 of POS byte hex 0102

(pos [0]). Also shown is a resource_setting that corresponds to the pos_setting of the named_choice. The resources allocated in this named_item are I/O addresses and interrupt levels. A help string for this named_item is provided below the last named_choice.

5 This is the second named_item for the adapter. The title of the field is "Arbitration Level for SDLC." The user can toggle between the 15 named_choices. Each named_choice has a unique pos_setting assigned to it in bit locations 0 through 3 of POS byte hex 0103 (pos [1]). Also shown is a resource_setting that corresponds to the pos_setting of the named_choice. The resources allocated in this named_item are arbitration levels. A help string for this named_item is provided below the last named_choice.

Adapter Description Program

The adapter description program (ADP) is used to configure memory space on an adapter that cannot be handled by normal ADF syntax. However, memory adapters that require initialization of more than their POS values should use an initialization program, not an ADP. The ADP is called when the item selected in the ADF contains the appropriate keyword.

Two basic types of memory can be assigned addresses within the memory address space: system memory and nonsystem memory.

System memory is memory that is managed and allocated by the primary operating system. It remains assigned to or fixed in the 4GB physical address space, and its contents can be accessed or modified only by an independent master (bus master, system master, or DMA controller).

Nonsystem memory is memory that is not managed or allocated by the primary operating system. It is made up of memory-mapped I/O devices, memory on an adapter that can be directly modified by the adapter, or memory that can be relocated within the address space, such as bank switched and EMS (expanded memory specifications).

The ADP initializes its POS data in the POS array and updates the parameter block. It must be in EXE program format, should not exceed 10KB, and start its code at offset hex 100. The filename for the ADP is CCardID.ADF (CardID is the adapter ID in hex).

Notes:

1. Memory address space is the only system resource that can be allocated by an ADP.
2. The Set Configuration program does not include support for memory adapters that have translation tables.
3. ADPs should not output directly to the display or prompt the user for input; unpredictable results will occur.
4. To ensure system integrity, ADPs must not write directly to locations in nonvolatile RAM.

Syntax for ADPs

There are three keywords in the ADF syntax that define an ADP: SysMem, Exec, and Vcheck. SysMem is used to assign address space to system memory, and Exec is used to assign address space to nonsystem memory. Vcheck is used to indicate that the ADP will verify a resource choice in the context of the allocated resources for the installed adapters.

System Memory ADPs

The SysMem keyword in the ADF indicates that the adapter contains system memory and that an ADP is used to configure the adapter's POS data and the amount of its system memory.

Adapters should not split their memory between the 1MB to 16MB range and the 16MB to 4GB range unless the memory is contiguous across the 16MB address.

System memory is the first resource allocated by the Set Configuration program. The system memory ADPs are run before other resources are allocated and in the following priority:

1. System board memory
2. Memory address size (24-bit address followed by 32-bit)
3. Memory granularity (larger block size first)
4. Slot number starting with connector 1.

If the ADP is not found by the Set Configuration program, the user is prompted to insert the appropriate diskette into drive A. If the ADP still is not found, the adapter is disabled.

Nonsystem Memory ADPs

An ADP can be used by adapters that want to map nonsystem memory into the address space above 1MB. The memory on these adapters does not get included as part of the top of memory value stored in CMOS RAM. This memory must be mapped starting at the highest addressable location and counting down.

The Exec keyword, included within a NamedItem in the ADF, indicates that the adapter has an ADP. It should be the last Choice of a NamedItem list. The ADP is run at the time that Choice is selected. If the ADP cannot be found by the Set Configuration program, the user is prompted to insert the appropriate diskette into drive A. If the ADP still cannot be found, the adapter is disabled if that Choice is selected. When the Exec keyword is used, POS bits that are modified by the ADP are identified by Xs in each bit position. The following is an example of how the Exec keyword is used.

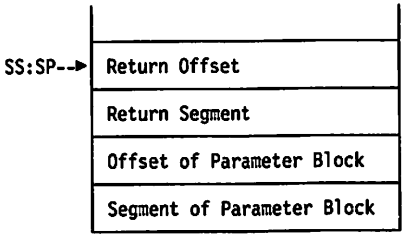
```
NamedItem Prompt "Adapter RAM"
     Choice "C000"
          pos[2]=XXXX0000b
          pos[1]=11000000b  mem 0c0000h-0c7fffh

     Choice "C800"
          pos[2]=XXXX0000b
          pos[1]=11001000b  mem 0c8000h-0cffffh

     Choice "D000"
          pos[2]=XXXX0000b
          pos[1]=11010000b  mem 0d0000h-0d7fffh

     Choice "D800"
          pos[2]=XXXX0000b
          pos[1]=11011000b  mem 0d8000h-0dffffh

     Choice "Extended Memory"
          pos[2]=XXXXXXXXb
          pos[1]=XXXXXXXXb  Exec

     Help
          "Select a memory address space."
```

Program Call

When the Set Configuration program calls the ADP, the address of the ADP parameter block is passed on the stack. This parameter block contains pointers to data structures that are used by the ADP.

The Set Configuration program checks the parameter block before and on return from the ADP; however, the ADP must maintain the integrity of these data structures during its execution.

When the ADP is given control, the stack is set up as follows:



The ADP must preserve the contents of the DS, DI, SI, and BP registers as well as the stack and stack pointers. The ADP ends using a Far Return with the stack set up as shown.

The following figure shows the format of the parameter block.

00H	Function	Slot
02H	Offset to System Memory (< 16MB)	
04H	Segment to System Memory (< 16MB)	
06H	Offset to System Board Memory (< 16MB)	
08H	Segment to System Board Memory(< 16MB)	
0AH	Offset to POS Array	
0CH	Segment to POS Array	
0EH	Offset to POS Mask	
10H	Segment to POS Mask	
12H	Offset to Adapter ID List	
14H	Segment to Adapter ID List	
16H	Offset to ADP Function Call Entry	
18H	Segment to ADP Function Call Entry	
1AH	Offset to Resource Query Parameters	
1CH	Segment to Resource Query Parameters	
1EH	Offset to System Memory (16MB to 4GB)	
20H	Segment to System Memory (16MB to 4GB)	
22H	Offset of System Board Memory (16MB to 4GB)	
24H	Segment of System Board Memory (16MB to 4GB)	

Figure 4. ADP Parameter Block

The following is a description of the parameter block values and the data structures that are pointed to.

ADP Function: This byte specifies the operation that the ADP will perform. The functions are listed on page 28. The ADP uses this byte to return the results of the operation, successful (0) or unsuccessful (not 0). The ADP returns a hex 01 in this byte if the operation was not successfully completed, and returns a hex FF if the function is not supported.

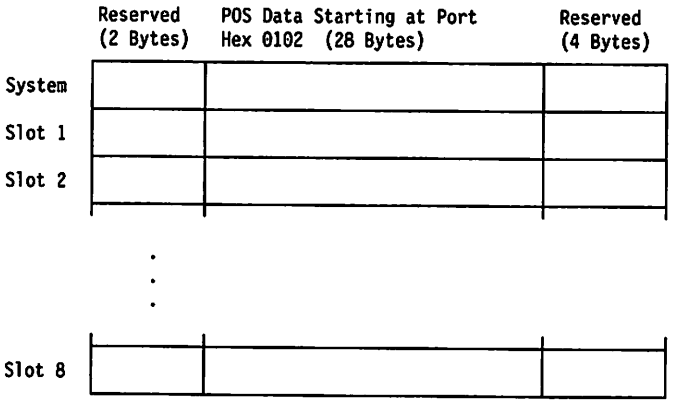
Slot: This is the slot number of the adapter to be configured. The value is not required for all ADP functions.

System Memory Below 16MB: This is a pointer to the current count of all system memory below 16MB. The count is a doubleword value and is the number of 1KB blocks of memory below 16MB already allocated as system memory. This count includes the count in the System Board Memory Below 16MB field. This value is adjusted by ADPs as they map their system memory into the system address space.

The value is initially set to 0 before any configuring is started. An ADP must not increase this value to the point that memory gets mapped beyond the 16MB address.

System Board Memory Below 16MB: This is a pointer to the count of system memory on the system board below 16MB. The count is a doubleword value and is the number of 1KB blocks of system board memory below 16MB. Split memory is included in this count. (Split memory is that part of the first 1MB of system-board memory that is not assigned addresses within the memory-address space below 1MB.) This value is set by the system description program and must not be adjusted by any other program.

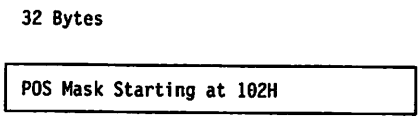
POS Array: This is a pointer to a table of the POS data for the system board and the I/O slots. The ADP should initialize only those locations that correspond to its adapters, and only the four POS data bytes from hex 0102 to 0105. The following is the format of the table and each entry.



POS Mask: This is a pointer to the 32-byte field containing the POS mask for adapter POS information. The POS mask data is initialized by an ADP, and is applied to POS data for all slots with that adapter ID.

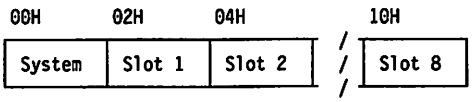
The mask bits are set to 1 for all POS bit positions that are initialized by the ADP. The mask bits are set to 0 for those POS bits that are not used, or that are initialized by statements in the ADF.

The following is the format of the POS mask data pointed to by this field.



The POS mask must be set every time the ADP is called.

Adapter ID List: This is a pointer to a list containing the system and adapter IDs that are currently installed in the system. The adapter IDs are stored as words at the following offsets.



ADP Function Call Entry: This pointer is the program entry point for calls by the ADP to the Set Configuration program when using the Resource Query function calls (see "Resource Query Functions" on page 34). The ADP does a Far Call when calling the Set Configuration program.

Resource Query Parameters: This pointer is the address for the Resource Query parameter block that the ADP uses to pass parameters when calling a Resource Query function. The address for this parameter block is allocated by the Set Configuration program and should not be changed by the ADP.

System Memory Between 16MB and 4GB: This is a pointer to the current count of all system memory between 16MB and 4GB. The count is a doubleword value and is the number of 1KB blocks from 16MB to below 4GB already allocated as system memory. This value is adjusted by ADPs as they map their system memory into the system address space.

The value is initially set to 0 before any configuring is started. An ADP must not increase this value to the point that system memory gets mapped to the 4GB address (must be less than 4GB).

System Board Memory Between 16MB and 4GB: This is a pointer to the count of system memory on the system board between 16MB and 4GB. The count is a doubleword value and is the number of 1KB blocks of system memory on the system board from 16MB to 4GB. This value is set by the system description program and must not be adjusted by any other program.

ADP Functions

When control is passed to an ADP, a function code is also passed. This function code indicates the operation to be performed by the ADP. The ADPs do not need to support all functions. If a function code is passed to an ADP that is not supported, the ADP should return a hex FF in the Function byte.

The following functions are supported by the Set Configuration program.

Function (Hex)	Description
01	Configure the adapter in the given slot.
02	Configure all adapters corresponding to the ADP.
03	Deallocate resources for the adapter in the given slot.
04	Deallocate resources for all adapters corresponding to the ADP.
05	Deduce the configuration of the adapter in the given slot.
06	Disable a block of system memory.
07	Remap the system memory to above the 16MB address.
08	Remap the system memory to below the 16MB address.
09	Configure the system memory on the adapter in the given slot.
0A	Configure the system memory on all adapters corresponding to the ADP.
0B	Deallocate the system memory on the adapter in the given slot.
0C	Deallocate the system memory on all adapters corresponding to the ADP.
0D	Deduce the configuration of the system memory on the adapter in the given slot.
80	Check the validity of an ADF choice

Figure 5. ADP Functions

Configure the Adapter

This function is indicated by 01 in the Function byte. The ADP configures the adapter in the slot specified, and updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

All ADPs defined by the SysMem or Exec keywords must support this function.

Configure All Adapters

This function is indicated by 02 in the Function byte. All adapters associated with that ADP are configured. If more than one adapter is being configured during this function, they are configured in the ascending order of their slot number. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

All ADPs defined by the SysMem keyword must support this function.

Deallocate Resources for an Adapter

This function is indicated by 03 in the Function byte. The resources allocated to the slot specified are deallocated. An ADP can deallocate only those resources it has allocated. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

All ADPs defined by the SysMem or Exec keywords must support this function.

Note: The Set Configuration program can deallocate memory space, previously allocated by an ADP, without calling the ADP.

Deallocate Resources for All Adapters

This function is indicated by 04 in the Function byte. The ADP deallocates all resources for all its adapters. The ADP can affect only those resources and adapters that it changed during allocation. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

All ADPs defined by the SysMem or Exec keywords must support this function.

Deduce Adapter Configuration

This function is indicated by 05 in the Function byte. The ADP interprets the data in the POS array for the slot specified and determines what memory space is allocated to the adapter.

If the ADP was defined by the Exec keyword, it uses the Request Memory Block function to request the address space again. If the address space is not allocated, the ADP leaves the POS data unchanged and returns with an error in the Function byte.

If the ADP was defined by the SysMem keyword, the current top of memory is checked to determine if this adapter was previously assigned the next address space. If not, memory on another adapter may have been deallocated, and the ADP should reconfigure its adapter to the current top of memory. The ADP then updates the current count of system memory in the parameter block before returning.

When the ADP detects a difference between the actual memory on the adapter and its configuration, the ADP calls the Enable Memory Prompt function and handles memory according to the user response.

All ADPs defined by the SysMem or Exec keywords must support this function.

If the Set Configuration program is deducing the configuration of an adapter and a match cannot be found between the data stored in CMOS/NVRAM and any of the ADF choice POS definitions, the adapter will be configured to the first choice in the list and the CMOS/NVRAM will be updated to reflect this. If the adapter is in conflict with any other adapters when configured to the first choice, the adapter will be disabled. The only indication that the user will have of this change is by looking at the Change or View Configuration screen. A message indicating that changes were made will also be displayed when the user exits the Set Configuration menu.

Disable Block of System Memory

This function is indicated by 06 in the Function byte. The ADP disables the smallest amount of memory that it possibly can for the specified slot. The ADP also decreases the current count of memory value affected (either in the below 16MB value or the 16MB to 4GB value), and updates the POS array to reflect the disabled memory.

The ADP can interpret the POS data to determine where its memory was previously mapped. If its memory is not currently mapped as the top of memory, the ADP does not change any values and indicates a successful operation. If its memory is currently mapped as the top of memory, but the ADP cannot disable the memory, the ADP returns a hex 01 in the ADP Function byte indicating the operation was not successfully completed.

The ADP should support multiple calls to disable memory. For each call, an additional block of memory should be disabled until all memory on the adapter is disabled. If all memory has been disabled

and the ADP is called again, the ADP returns an error indicating that all memory for the slot specified is already disabled.

All ADPs defined by the SysMem keyword must support this function.

Remap System Memory to Above 16MB

This function is indicated by 07 in the Function byte. All memory is remapped to the area above 16MB. The ADP updates its POS values in the POS array and also updates both top of memory values to indicate the changes. If its memory is not currently mapped as the top of memory, the ADP does not change any values and indicates a successful operation.

All ADPs defined by the SysMem keyword must support this function; adapters that have specified 24-bit addressing will not receive this request.

Remap System Memory to Below 16MB

This function is indicated by 08 in the Function byte. All memory for this adapter is remapped to the area below 16MB. The ADP updates its POS values in the POS array and also updates both top of memory values to reflect the changes. If its memory is not currently mapped as the top of memory, the ADP does not change any values and indicates a successful operation.

All ADPs defined by the SysMem keyword must support this function.

Configure System Memory on an Adapter

This function is indicated by 09 in the Function byte and is used only when the ADP is defined with both the SysMem and Exec keywords. The ADP configures only the system memory on the adapter in the slot specified. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

Configure System Memory on All Adapters

This function is indicated by hex 0A in the Function byte and is used only when the ADP is defined with both the SysMem and Exec keywords. The ADP configures only the system memory on all adapters associated with the ADP. If more than one adapter is being configured, they are configured in the ascending order of their slot number. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

Deallocate System Memory on an Adapter

This function is indicated by hex 0B in the Function byte and is used only when the ADP is defined with both the SysMem and Exec keywords. The system memory previously allocated to the adapter by the ADP is deallocated. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

Deallocate System Memory on All Adapters

This function is indicated by hex 0C in the Function byte and is used only when the ADP is defined with both the SysMem and Exec keywords. The ADP deallocates system memory that it previously allocated for all its adapters. The ADP updates the appropriate system memory and POS array values in the parameter table before returning to the Set Configuration program.

Deduce System Memory Configuration on an Adapter

This function is indicated by hex 0D in the Function byte and is used only when the ADP is defined with both the SysMem and Exec keywords. The ADP checks the current top of memory to determine if the adapter was previously assigned the next address space. If not, memory on another adapter may have been deallocated and the ADP should reconfigure the starting address of its adapter to the current top of memory. The ADP then updates the current count of system memory in the parameter table before returning.

When an ADP detects a difference between the actual memory on the adapter and its configuration, the ADP uses the Enable Memory by Prompt function and handles memory according to the user response.

Check Validity of Choice

This function is indicated by hex 80 in the function byte. It indicates whether the choice is valid or not for the current system configuration. The parameter block for this function is different than those defined previously. The following figure shows the parameter block for this function.

00H	Function	Reserved as 00
02H	Starting Address of the First I/O Address Range (0 if no I/O Address Range)	
04H	First Interrupt Level (0 if no Interrupt Level)	
06H	First Arbitration Level (Hex 0FFFF if no Arbitration Level)	
08H	Starting Address of the First Memory Address Range (0 if no Memory Address Range)	
0CH	Offset of the Adapter IDs List.	
0EH	Segment of the Adapter IDs List.	
10H	Offset of the ADP Function Call Entry Point	
12H	Segment of the ADP Function Call Entry Point	
14H	Offset of the Resource Query Parameters	
16H	Segment of the Resource Query Parameters	

Figure 6. Check Validity of ADF Choice Function.

If the choice is valid, the ADP returns a 0 in the function byte. If the choice is not valid, the named_item is not included in the list of available settings during the configuration process.

All ADPs that use the Vcheck keyword must support this function.

Resource Query Functions

The Resource Query functions allows the ADP to communicate with the Set Configuration program. The ADP can query the memory resource-allocation table and request a block of addresses for an adapter. Before calling, the ADP loads the specific values into the Resource Query parameter block for that function and restores the DS register if it was changed.

A Resource Query function is initiated by making a far call to the program entry point passed to the ADP in the parameter block. Most Resource Query functions are designed to allocate address space for nonsystem memory. ADPs configuring system memory are generally concerned with only the current top of memory value.

Code (Hex)	Description
00	Request Memory Block
01	Check Memory Block
02	De-allocate Memory Block
03	Enable an Adapter
04	Disable an Adapter
05	Enable Memory with a Prompt
06	Get Number of Slots
07	Get Slot Information

Figure 7. Resource Query Function Calls.

The allocation of nonsystem memory is handled by the Set Configuration program. The ADP simply requests a block of memory within a specified range. The ADP should try to map its memory into the highest block possible.

Note: The size of the block and high end of the range plus 1 must be divisible by 4KB.

The Set Configuration program responds with a return code that indicates whether the request was successful or unsuccessful. If the request is successful, the Set Configuration program has also returned the starting address of the memory block allocated. The ADP then sets the appropriate bits in the POS array.

The Set Configuration program attempts to allocate the highest block of memory within the requested range. If the highest block is not available, the starting address is decreased by the size of the block requested until either a block of memory space is available, or the starting address is below the requested range.

When the ADP returns control to the Set Configuration program, it must indicate in the Function byte whether or not the configuration was successful.

The parameter block for a Resource Query function is defined as follows:

00H	Function	Slot
02H	Start_add_high	
04H	Start_add_low	
06H	End_add_high	
08H	End_add_low	
0AH	Size_high	
0CH	Size_low	

Figure 8. Resource Query Parameter Block

Request Memory Block

This function is used to request a block of memory space for adapters with nonsystem memory. The ADP provides the range where its memory can be mapped and the number of bytes to be mapped.

The ADP can only allocate memory to adapters with its ID. Once a memory block has been allocated, it remains allocated until that ADP deallocates the space, or until automatic configuration is run.

The following figure shows the values for the parameter block passed between the ADP and the Set Configuration program.

```
        Function = 00
        Slot = Slot number of adapter
        Start_addr_high = Beginning of range (High Word)
        Start_addr_low = Beginning of range (Low Word)
        End_addr_high = End of range (High Word)
        End_addr_low = End of range (Low Word)
        Size_high = Requested block size in bytes (High Word)
        Size_low = Requested block size in bytes (Low Word)

    Return:
        Successful:
            Function = 00
            Start_addr_high = Beginning of the block allocated
            Start_addr_low = Beginning of the block allocated
        Unsuccessful:
            Function = Error code (01, 02, 03, 06, 08 or 09)
                    (refer to "Error Codes" on page 40)
            All other values unchanged.
```

Figure 9. Request Memory Block Function

Check Memory Block

This function allows the ADP to get the current memory block assigned to an adapter in a given slot. An ADP can check the allocated memory blocks for any slot.

Because a list of memory blocks can be allocated to an adapter, the ADP must specify which block in the list to use. An ADP can check the entire list by starting at number 0 and checking the next block until an error code is returned.

```
        Function = 01
        Slot = Slot number of adapter
        End_addr_high = Block number

    Return:
        Successful:
            Function = 00
            Start_addr_high = Beginning of block (High Word)
            Start_addr_low = Beginning of block (Low Word)
            Size_high = Allocated block size in bytes (High Word)
            Size_low = Allocated block size in bytes (Low Word)
        Unsuccessful:
            Function = Error code (01, 05 or 09)
                    (refer to "Error Codes" on page 40)
            All other values unchanged.
```

Figure 10. Check Memory Block Function

Deallocate Memory Block

This function allows the ADP to deallocate a block of memory for a given slot. The ADP can deallocate only the memory space assigned to adapters with its ID. The ADP specifies only the beginning address of the memory block.

```
     Function = 02
     Slot = Slot number of adapter
     Start_addr_high = Beginning of memory block (High Word)
     Start_addr_low = Beginning of memory block (Low Word)

Return:
     Successful:
          Function = 00
     Unsuccessful:
          Function = Error code (01, 04 or 09)
                     (refer to "Error Codes" on page 40)
          All other values unchanged.
```

Figure 11. Deallocate Memory Block Function

Enable an Adapter

This function allows the ADP to enable and temporarily configure one of its adapters. Because the data in the POS mask array is written when the adapter is enabled, the ADP must initialize the POS mask before enabling the adapter.

```
     Function = 03
     Slot = Slot number of the adapter

Return:
     Successful:
          Function = 00
     Unsuccessful:
          Function = Error code (01)
                     (refer to "Error Codes" on page 40)
          All other values unchanged.
```

Figure 12. Enable an Adapter Function

Disable an Adapter

This function disables the adapter. Configuration information is not required because only the adapter enable bit is changed.

```
     Function = 04
     Slot = Slot number of the adapter

Return:
     Successful:
          Function = 00
     Unsuccessful:
          Function = Error code (01)
                   (refer to "Error Codes" on page 40)
          All other values unchanged.
```

Figure 13. Disable an Adapter Function

Enable Memory by Prompt

This function allows an ADP to prompt the user for a decision when the amount of memory on the adapter does not match configuration. When this function is called, the Set Configuration program displays a message and waits for the user to type a Y or N. The message asks the user if the memory should be enabled and the adapter reconfigured. If a Y is typed, the Set Configuration program returns a 01 in the Function byte and the ADP enables memory and reconfigures the adapter. If an N is typed, the Set Configuration program returns a 00 in the Function byte, the ADP disables the changed memory and does not change the configuration.

Note: The Set Configuration program also returns a 01 in the Function byte if the slot number specified in the parameter block is not a valid slot.

This function should be used only when the ADP was called with a Deduce function. An example of when this function is used is if memory was added to a configured adapter.

```
     Function = 05
     Slot = Slot number of the adapter

Return:
     Function = 01 - Enable memory
     Function = 00 - Leave memory disabled
        ("Error Codes" on page 40 not used for
           this function.)
```

Figure 14. Enable Memory by Prompt Function

Get Number of Slots

This function allows the ADP to determine the number of channel connectors in the system, which can be used to determine the size of the POS array, adapter ID list, and other arrays.

```
        Function = 06

    Return:
        Function = 00
        Slot = Number of channel connectors in the system
```

Figure 15. Get Number of Slots Function

Get Slot Information

This function requests information about a particular channel connector. The ADP can use this information to determine the address size of the connector and whether or not the connector has a video extension.

```
        Function = 07
        Slot = Slot number to get information about

    Return:
        Successful:
            Function = 00
            Slot = address size of slot
            Start_addr_high:
               bit 0 = 1, slot contains video extension
                       0, slot does not contain video extension
               bits 1 - 15 = reserved (set to 0)
        Unsuccessful:
            Function = Error code (01)
                     (refer to "Error Codes" on page 40)
            All other values unchanged.
```

Figure 16. Get Slot Information Function

Error Codes

The following is a list of the error codes used by the ADP and Set Configuration program. The error codes are the same for the ADP functions and the Resource Query functions.

Error Code	Description
01	The slot number specified was not valid.
02	The size specified is not valid.
03	The resource is already allocated to an adapter.
04	Attempted to deallocate resources that were not allocated.
05	A list value is not valid.
06	The memory range specified is not valid.
07	The function passed by the ADP is not valid.
08	Cannot allocate any more memory blocks to this adapter.
09	All memory for this adapter has already been disabled.
FF	Operation is not supported.

Figure 17. Error Codes

ADP Example

The following is an example of an adapter description program that is defined by the Exec keyword.

```
;-------------------------------------------------------------------;
;                                                                   ;
;  Sample ADP defined by the EXEC keyword.                          ;
;                                                                   ;
;  Adapter specifications:                                          ;
;       Adapter has a 1M buffer of nonsystem memory that can be     ;
;       mapped on any 1M boundary in the area from 2G to 4G - 1.    ;
;                                                                   ;
;       Adapter POS registers:                                      ;
;               register 100h = 77h                                 ;
;               register 101h = 77h                                 ;
;               register 102h =                                     ;
;                       bit 0 = adapter enable bit                  ;
;                       bit 1 = nonsystem memory enable             ;
;               register 103h = low byte of high word of assigned   ;
;                               memory block.                       ;
;               register 104h = high byte of high word of assigned  ;
;                               memory block.                       ;
;                       (Low word of memory block always 0).        ;
;                                                                   ;
;-------------------------------------------------------------------;
.286C
.286p

;-------------------------------------------------------------------;
;  Constants:
;-------------------------------------------------------------------;

adapter_id              equ     7777h   ;ID for this adapter
begin_block_h           equ     8000h   ;high word for start of 2G
begin_block_l           equ     0       ;low word for start of 2G
end_block_h             equ     0ffffh  ;high word for end of 4G - 1
end_block_l             equ     0ffffh  ;low word for end of 4G - 1
size_h                  equ     10h     ;high word for size of 1M
size_l                  equ     0       ;low word for size of 1M
mem_enable_mask         equ     2       ;mask for memory enable bit in POS 102h
not_mem_enable          equ     0fdh    ;complement of mem_enable_mask

max_pos_bytes           equ     32      ;number of POS bytes in pos array
pos_array_size          equ     34      ;number of total bytes in pos array

;ADP functions
configure               equ     1       ;configure an adapter
deallocate              equ     3       ;deallocate resources for an adapter
deallocate_all          equ     4       ;deallocate resources for all adapters
deduce                  equ     5       ;deduce the adapter configuration
```

```
;ADP function calls - resource query
request_mem_block       equ     0       ;request a block of nonsystem memory
check_mem_block         equ     1       ;check memory block allocated to adapter
deallocate_mem_block    equ     2       ;de-allocate a block of nonsystem memory

;ADP parameter block
adp_pb          struc
function        db      ?               ;ADP function
slot            db      ?               ;slot being configured
top_mem_os      dw      ?               ;pointer to top of system memory below
top_mem_seg     dw      ?               ;       16M
top_pmem_os     dw      ?               ;pointer to top of planar system memory
top_pmem_seg    dw      ?               ;       below 16M
pos_array_os    dw      ?               ;pointer to POS array
pos_array_seg   dw      ?
pos_mask_os     dw      ?               ;pointer to POS mask
pos_mask_seg    dw      ?
id_list_os      dw      ?               ;pointer to list of adapter IDs
id_list_seg     dw      ?
res_query_os    dw      ?               ;resource query entry point
res_query_seg   dw      ?
adp_pb_os       dw      ?               ;pointer to ADP function call parameter
adp_pb_seg      dw      ?               ;       block
top_mem32_os    dw      ?               ;pointer to top of system memory from
top_mem32_seg   dw      ?               ;       16M to below 4G
top_pmem32_os   dw      ?               ;pointer to top of planar system memory
top_pmem32_seg  dw      ?               ;       from 16M to below 4G
adp_pb          ends

;Resource Query parameter block
res_query       struc
rqfunction      db      ?               ;ADP function call
rqslot          db      ?               ;slot to perform function on
start_addr_high dw      ?               ;high word of starting address
start_addr_low  dw      ?               ;low word of starting address
end_addr_high   dw      ?               ;high word of ending address
end_addr_low    dw      ?               ;low word of ending address
size_high       dw      ?               ;high word of memory block size
size_low        dw      ?               ;low word of memory block size
res_query       ends
```

```
;-------------------------------------------------------------------;
;  Begin program:
;-------------------------------------------------------------------;

code    segment para    public 'code'
start   proc    far
        assume  cs:code,ds:code,ss:code
        org     100h                    ;program starts at offset 100h

        push    bp                      ;save base pointer on stack
        mov     bp,sp                   ;initialize base pointer to stack pointer
        push    es                      ;save registers
        push    ds
        push    si
        push    di
        jmp     prog                    ;jump around data area

;-------------------------------------------------------------------;
;  Data area:
;-------------------------------------------------------------------;

data_seg        dw      ?               ;holds value of DS for Setup program


prog:   mov     cs:data_seg,ds          ;save value of DS from Setup program
        mov     ax,[bp+8]               ;get segment of ADP parameter block
        mov     ds,ax                   ;put ADP parameter block segment in DS
        mov     di,[bp+6]               ;put offset of ADP parameter block in DI

;-------------------------------------------------------------------;
;  Determine what ADP function is to be performed.
;-------------------------------------------------------------------;

        mov     al,[di].function        ;get ADP function
        cmp     al,configure            ;is function configure?
        je      do_configure            ;yes, branch
        cmp     al,deallocate           ;is function deallocate?
        je      do_deallocate           ;yes, branch
        cmp     al,deallocate_all       ;is function deallocate all?
        je      do_deallocate_all       ;yes, branch
        cmp     al,deduce               ;is function deduce?
        jne     bad_command
        jmp     do_deduce               ;yes, branch
bad_command:
        mov     al,0ffh                 ;function not supported return code
        jmp     exit                    ;exit the ADP
```

```
;-------------------------------------------------------------------;
;  Configure the adapter.
;-------------------------------------------------------------------;

do_configure:
        call    get_rq_pb               ;point ES:SI to resource query param blk
        mov     es:[si].rqfunction,request_mem_block    ;function to perform
        mov     al,[di].slot            ;get slot of adapter being configured
        mov     es:[si].rqslot,al                       ;slot to configure
        mov     es:[si].start_addr_high,begin_block_h   ;beginning of allowable
        mov     es:[si].start_addr_low,begin_block_l    ;       range
        mov     es:[si].end_addr_high,end_block_h       ;end of allowable range
        mov     es:[si].end_addr_low,end_block_l
        mov     es:[si].size_high,size_h                ;size of block required
        mov     es:[si].size_low,size_l
        call    rqfunction_call         ;do the resource query function call
        cmp     es:[si].rqfunction,0    ;was the function successful?
        je      set_pos_bits            ;yes, set the appropriate POS bits
        jmp     unsuccessful

set_pos_bits:
        mov     bx,es:[si].start_addr_high              ;get assigned address
        mov     al,[di].slot            ;slot value to get pos array pointer for
        call    point_to_pos            ;point ES:SI to pos array
        mov     es:[si+3],bl            ;set POS value for register 103h
        mov     es:[si+4],bh            ;set POS value for register 104h
        jmp     successful

;-------------------------------------------------------------------;
;  De-allocate resources for the adapter.
;-------------------------------------------------------------------;

do_deallocate:
        mov     al,[di].slot            ;get slot of adapter being configured
        call    deallocate_mem          ;call proc to de-allocate the memory
        call    point_to_pos            ;get pointer to pos array for adapter
        and     byte ptr es:[si+2],not_mem_enable       ;clear memory enable bit
        mov     al,0                    ;successful error code
        jmp     exit                    ;exit ADP
```

```
;-------------------------------------------------------------------;
;  De-allocate resources for all adapters.
;-------------------------------------------------------------------;

do_deallocate_all:
        mov     cx,8                    ;max number of adapter slots per system
        mov     ax,[di].id_list_seg     ;get segment of list of adapter IDs
        mov     es,ax
        mov     si,[di].id_list_os      ;get offset of list of adapter IDs
        mov     al,1                    ;first slot to check

deallocate_lup:
        add     si,2                    ;point to next ID in list (skip planar)
        cmp     es:[si],adapter_id      ;does current slot have this adapter?
        jne     no_adapter              ;no, skip dealloction of memory
        call    deallocate_mem          ;call proc to deallocate the memory
        call    point_to_pos            ;get pointer to pos array for adapter
        and     byte ptr es:[si+2],not_mem_enable       ;clear memory enable bit

no_adapter:
        inc     al                      ;next slot to check
        loop    deallocate_lup

        mov     al,0                    ;successful return code
        jmp     exit                    ;exit ADP

;-------------------------------------------------------------------;
;  Deduce current configuration of adapter.
;-------------------------------------------------------------------;

do_deduce:
        mov     al,[di].slot
        call    point_to_pos            ;point ES:SI to pos array
        mov     al,es:[si+2]            ;read value of POS register 102h
        and     al,mem_enable_mask      ;see if the memory is enabled
        jz      unsuccessful            ;if mem enable bit not set, branch

        mov     bl,es:[si+3]            ;read value of POS register 103h
        mov     bh,es:[si+4]            ;read value of POS register 104h
        call    get_rq_pb               ;point ES:SI to resource query param blk
        mov     es:[si].rqfunction,request_mem_block    ;function to perform
        mov     al,[di].slot            ;get slot of adapter being deduced
        mov     es:[si].rqslot,al                       ;slot to deduce
        mov     es:[si].start_addr_high,bx              ;beginning of allocated
        mov     es:[si].start_addr_low,begin_block_l    ;       range
        add     bx,size_h - 1           ;determine end value for this request
        mov     es:[si].end_addr_high,bx                ;end of allocated range
        mov     es:[si].end_addr_low,end_block_l
        mov     es:[si].size_high,size_h                ;size of block required
        mov     es:[si].size_low,size_l
        call    rqfunction_call         ;do the resource query function call
        cmp     es:[si].rqfunction,0    ;was the function successful?
        je      successful              ;yes, branch
        jmp     unsuccessful
```

```
;-------------------------------------------------------------------;
;  ADP function was successful.
;-------------------------------------------------------------------;

successful:
        mov     al,[di].slot
        call    point_to_pos            ;get pointer to pos array
        or      byte ptr es:[si+2],mem_enable_mask      ;set memory enable bit
        mov     al,0                    ;successful return code for ADP function
        jmp     exit                    ;exit the ADP

;-------------------------------------------------------------------;
;  ADP function was unsuccessful.
;-------------------------------------------------------------------;

unsuccessful:
        mov     al,[di].slot
        call    point_to_pos            ;get pointer to pos array
        and     byte ptr es:[si+2],not_mem_enable       ;clear memory enable bit
        mov     al,1                    ;unsuccessful return code for ADP function

;-------------------------------------------------------------------;
;  Exit the ADP.
;-------------------------------------------------------------------;

exit:   mov     [di].function,al        ;save return code of ADP function
        call    set_pos_mask            ;initialize the pos mask bits
        pop     di                      ;restore registers
        pop     si
        pop     ds
        pop     es
        pop     bp
        ret
start   endp
```

```
;-----------------------------------------------------------------------;
;  DEALLOCATE_MEM
;       Perform deallocate ADP function on the slot defined in the
;       AL register.
;-----------------------------------------------------------------------;

deallocate_mem  proc    near
        push    ax
        push    es
        push    cx
        push    si
        mov     bl,al                   ;save slot number in BL register

del_begin:
        call    get_rq_pb               ;point ES:SI to resource query param blk
        mov     es:[si].rqfunction,check_mem_block      ;function to perform
        mov     es:[si].rqslot,bl       ;slot to check
        mov     es:[si].end_addr_high,0 ;always check first block in list
        call    rqfunction_call         ;do the resource query function call
        cmp     es:[si].rqfunction,0    ;is there a memory block assigned?
        jne     del_done                ;no, done de-allocating memory

        call    get_rq_pb               ;point ES:SI to resource query param blk
        mov     es:[si].rqfunction,deallocate_mem_block ;function to perform
        call    rqfunction_call         ;do the resource query function call
        jmp     del_begin

del_done:
        pop     si
        pop     cx
        pop     es
        pop     ax
        ret
deallocate_mem  endp

;-----------------------------------------------------------------------;
;  GET_RQ_PB
;       Initializes ES:SI to point to the ADP function call
;       (resource query) parameter block.
;-----------------------------------------------------------------------;

get_rq_pb       proc    near
        push    ax
        mov     ax,[di].adp_pb_seg      ;put the segment of the ADP function
        mov     es,ax                   ;       call parameter block in ES
        mov     si,[di].adp_pb_os       ;put the offset in SI
        pop     ax
        ret
get_rq_pb       endp
```

```
;-------------------------------------------------------------------;
;  RQFUNCTION_CALL
;       Makes an ADP function call.
;-------------------------------------------------------------------;

rqfunction_call proc    near
        push    ds                      ;save registers
        push    es
        push    si
        push    di
        push    bx
        push    cx

        mov     bx,di                   ;set BX to start of ADP parameter block
        add     bx,res_query_os         ;ES:BX points to ADP function call entry
        mov     ax,cs:data_seg          ;restore the Setup data segment
        mov     ds,ax
        call    dword ptr es:[bx]       ;make the ADP function call

        pop     cx                      ;restore the registers
        pop     bx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
rqfunction_call endp

;-------------------------------------------------------------------;
;  POINT_TO_POS
;       Initializes ES:SI to point to the pos array for the slot
;       number defined in the AL register.
;-------------------------------------------------------------------;

point_to_pos    proc    near
        push    ax
        push    bx

        mov     bl,al                   ;slot number to get pos array pointer for
        mov     ax,[di].pos_array_seg   ;put the pointer to the beginning of the
        mov     es,ax                   ;       pos array into ES
        mov     si,[di].pos_array_os    ;put pointer to beginning in SI
        mov     ax,pos_array_size       ;get the size of each adapter pos array
        mul     bl                      ;multiply the size by the slot number
        add     si,ax                   ;set SI to the start of the pos array
                                        ;       for the specified slot
        pop     bx
        pop     ax
        ret
point_to_pos    endp
```

```
;-------------------------------------------------------------------;
;  SET_POS_MASK
;       Initializes the bits in the POS mask.
;-------------------------------------------------------------------;

set_pos_mask    proc    near
        mov     ax,[di].pos_mask_seg    ;put the segment of the pos mask into
        mov     es,ax                   ;       ES
        mov     si,[di].pos_mask_os     ;put segment of pos mask in SI
        mov     cx,max_pos_bytes        ;set CX to the number of POS bytes
        push    si
lup2:   mov     byte ptr es:[si],0      ;set all pos mask values to 0
        inc     si
        loop    lup2
        pop     si
        mov     byte ptr es:[si],mem_enable_mask        ;set mem enable bit for
                                                        ;   POS register 102h
        mov     byte ptr es:[si+1],0ffh ;set all bits for POS register 103h
        mov     byte ptr es:[si+2],0ffh ;set all bits for POS register 104h
        ret
set_pos_mask    endp

code    ends
        end     start
```

Initialization Programs

An initialization program is a memory image program that is loaded onto track 0, head 0 of drive hex 80. The master boot record executes the initialization programs before booting the operating system from the fixed disk (the initialization program is bypassed if the operating system is booted from the diskette drive). It is an alternate method of initializing an adapter that does not require the use of ROM or jumpers and switches on the adapter.

Note: These programs are run only when booting from the drive hex 80. They should not be used if the function of the program is critical to the adapter, or if the system would be adversely affected if the program is not run.

The Set Configuration program, selected from the Main Menu of the Reference Diskette, installs the initialization programs onto drive hex 80. (Automatic Configuration does not install these programs.) The Set Configuration program determines that an adapter has an initialization program by checking the adapter ADF. An adapter with a corresponding initialization program has the statement, InitProg <attribute>, in its ADF. The InitProg statement is placed after the FixedResources statement and before the NamedItem statement.

The attribute is a value from 0 to 127 that selects the order in which the initialization programs are run; the lowest value is run first. Attributes from 30 to 80 are available; attributes outside of this range are reserved. An example of the use of this attribute is to ensure that a specific adapter is initialized before another if both are installed in the same system.

The following rules apply to all initialization programs:

- The initialization program is always loaded beginning at offset hex 7C00. Any data variables referenced in the code may need their offsets adjusted to hex 7C00.
- The initialization program must begin with a 20 byte header. Execution of the initialization program begins on the 21st byte.
- The 16th byte of the program header is the revision level of the initialization program. The value in this byte should be incremented if a new revision of the program is released. When Set Configuration is run, the installation process checks this byte to ensure that the program with the latest revision level is installed. The other bytes in the header are set by the Set Configuration program.
- The initialization program cannot be more than 1024 bytes in size, including the 20 byte header.
- While executing, the initialization program should not assume that memory is available beyond its own programming size.
- The initialization program is shipped as a DOS file so that it can be merged onto the Reference Diskette in the same way as the ADFs. The name of the file is based on the ID of the adapter, and is in the form ICardID.ADF (CardID is the adapter ID in hex).
- When an initialization program finishes executing, it performs a Near Return to pass control back to the calling program. The stack will then be in the same state it was when the initialization program was called.

A revision to an initialization program is installed the same way as the original is installed. The revision to the program has a higher value in the revision level byte than previous versions. The Set Configuration program compares the revision levels of initialization programs installed on the fixed disk drive with those on the Reference Diskette. If the program on the Reference Diskette has a greater revision level than the one on the fixed disk drive, it updates the program on the fixed disk drive.

Index

A

adapter configuration call 29
adapter configuration, order of 4
adapter description program (ADP) 20
  error codes 40
  example 41
  function call entry 27
  function codes 28
  memory above 16MB 27
  nonsystem memory 22
  POS mask 26
  program call 23
  query parameters 27
  resource query 34
  syntax 21
  system memory 21
adapter identification 2
adapter init programs 49
adapter location 39
adapterId keyword 8
adaptername keyword 8
address keyword 14
arb keyword 13
array, POS 26
automatic configuration 1, 3

B

backup configuration utility 6
battery failure 3
begin_device parameter 15

C

call, ADP function 23
change configuration utility 5
check memory block 36
check validity of choice 33
choice keyword 11
CMOS RAM, card ID bytes 2
configuration determination call 29
configuration error 3
configure system memory 31
configure the adapters 28
copy an option diskette utility 6
CRC error 3

D

deallocate memory block 37
deallocate resources 29
deallocate system memory 32
deduce configuration 29
deduce system memory configuration 32
device keyword 15
disable adapter 38
disable block of memory 30
diskettes, reference and option 1

E

enable adapter 37
enable memory by prompt 38
entry point, function call 27
error codes 40
example, ADP 41
exec keyword 14

F

file naming 1
fixedresources keyword 8
function call entry point, ADP 27
function call, ADP 23
functions, ADP 28
  check validity 33
  configure system memory 31
  configure the adapters 28
  deallocate resources 29
  deallocate system memory 32
  deduce configuration 29
  deduce system memory 32

© Copyright IBM Corp. 1989, 1991

functions, ADP *(continued)*
disable memory 30
remap memory 31
functions, query 34
check memory block 36
deallocate memory block 37
enable memory prompt 38
enable/disable adapter 37
get slot data 39
request memory block 35

G

get slot information 39

H

help keyword 12
help text 5

I

implement_id parameter 15
information on adapter slots 39
initialization programs 49
initprog keyword 9
input keyword 11
int keyword 13
io keyword 13
items parameter 9

L

location, adapter 39
lock keyword 9

M

masking POS 26
mem keyword 14
memory above 16MB 27
memory and ADPs 21
memory below 16MB 25
memory call, disable 30
memory call, remap 31
memory prompt 38
memory, system and nonsystem 20

N

nameditem keyword 10
naming convention, file 1
nodma keyword 15
nonsystem memory 20, 22
nonvolatile RAM, card ID bytes 2
numbytes keyword 8

O

option diskette 2

P

parameter table 25
pio keyword 14
POS array 26
pos keyword 12
POS mask 26
POST error message files 3
program call, ADP 23
prompt keyword 11
prompt to enable memory 38

Q

query parameters 27

R

reference diskette 1
remap memory 31
request memory block 35
resource query functions 34
check memory block 36
deallocate memory block 37
enable memory prompt 38
enable/disable adapter 37
error codes 40
get slot data 39
request memory block 35
resource query parameters 27
resource_setting parameter 13
restore configuration utility 6
rs422 keyword 15

S

set configuration program 2
setup, system board 1
sharedarb keyword 13
syntax for ADPs 21
sysmem keyword 9
system board memory above 16MB 27
system board memory below 16MB 25
system board setup 1
system configuration utilities 1
system memory 20
system memory above 16MB 27
system memory ADP 21
system memory assignment 32
system memory below 16MB 25
system memory call, configure 31
system memory, deallocate 32
system resources 2

V

validity check 12, 33
vcheck keyword 12
view configuration 5

Notes:

First Edition (January, 1991)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

© Copyright International Business Machines Corporation 1991. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Special Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Notwithstanding the notice of the preceding paragraph regarding rights and licenses, the purchaser of this Technical Reference is granted certain rights as specified on the following page entitled "License Statement."

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM Personal System/2 PS/2 Micro Channel

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Intel is a trademark of the Intel Corporation.
Motorola is a trademark of Motorola, Inc.

© Copyright IBM Corp. 1991

LICENSE STATEMENT

By purchase of this Subsystem Control Block I/O Architecture Technical Reference (Technical Reference) you are authorized under IBM's copyrights in the Technical Reference:

1. To reproduce copies (which are not complete copies of the Technical Reference)
2. To prepare derivative works
3. To distribute copies.

Such copies may be in the form of hardware, software or a combination thereof as well as documents. No license is granted, however, to copy any IBM computer code.

You are eligible for a limited license under patents owned by IBM that cover inventions the practice of which is not avoidable in implementing, in device driver computer code, control blocks and elements complying with the Technical Reference at Part 2, Chapter 1, Sections titled "Command Control Blocks," "Termination Status Blocks," "Commands"; Part 2, Chapter 4; Part 3, Chapter 1 Section titled "Control Elements"; and Part 3, Chapter 4. Such patent license is non-exclusive, royalty free and nontransferable and permits you to make, use and sell such device driver computer code. To the extent hardware or software infringes a patent absent device driver computer code licensed hereunder it is not relieved of such infringement by the addition of such code. To qualify for this patent license you must purchase a copy of the Technical Reference, license IBM and its subsidiaries under the same terms and conditions for patents you own that apply to such device driver computer code or improvements thereof and activate the license set forth below.

You need take no action to activate the copyright authorization specified above. It is activated automatically by your purchase of this Technical Reference. You, of course, must make your own independent assessment regarding needs under any patents. Should you wish to activate the patent license specified above you may do so by notifying IBM to that effect in a letter to the IBM Director of Commercial Relations, IBM Corporation, 2000 Purchase Street, Purchase NY 10577.

Preface

The Technical Reference library is intended for those who develop hardware and software products for IBM* Personal Computers and IBM Personal System/2* products. Users should understand computer architecture and programming concepts.

This technical reference provides a description of hardware and software requirements and design considerations about the SCB architecture; it should be used with the following publications:

IBM Personal System/2 Hardware Interface Technical Reference

IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference.

This manual consists of the following sections:

Part 1, "Subsystem Control Block Architecture Overview," lists the basic design consideration of a system using the SCB architecture and describes the overall concepts of the architecture. It includes a brief description of the available modes and lists design requirements to consider in selecting the mode to use.

Part 2 describes the Locate mode.

Part 3 describes the Move mode.

The appendixes contain examples of the data structures used in specific modes.

Warning: The term "Reserved" describes certain bits and data areas that should not be changed. Use of reserved areas can cause compatibility problems. Programs and hardware writing to reserved areas should ensure that the areas are set to 0; programs and hardware reading these areas should ensure that the areas are treated as "don't care."

---

* IBM, Personal System/2, and Micro Channel are trademarks of the International Business Machines Corporation.

© Copyright IBM Corp. 1991

Notes:

Contents

Chapter 1. Introduction 1-1
Architecture Applications 1-3
Subsystem Control Block Characteristics 1-5
System Overview 1-6
Bus-Master Characteristics 1-8
System Resources 1-9
I/O Address Space 1-9
Memory Address Space 1-9
Private or Local Memory 1-10
Shared Memory 1-10
Memory Organization 1-12

Chapter 2. SCB Architecture Overview 2-1
Logical Levels 2-2
Physical Level 2-2
Delivery Level 2-3
Processing Level 2-4
Client 2-6
Server 2-6
Management Entity 2-6
Delivery-Service Structure 2-9
Delivery-Service Levels 2-11
Delivery Services 2-12
Delivery Modes 2-14
Locate Mode Delivery 2-14
Move Mode Delivery 2-15
Control Areas 2-18
Shared Memory 2-19
Structures 2-20
Control Blocks 2-20
Termination Status Blocks 2-20
Indirect Lists 2-20
Control Elements 2-21

Chapter 3. Subsystem Control Block Capabilities 3-1
Locate Mode 3-1
Move Mode 3-3
Selecting the Appropriate Mode 3-5
When to Use the Locate Mode 3-5

© Copyright IBM Corp. 1991

When to Use the Move Mode 3-6
When to Use the Combined Modes 3-9

Index X-1

Figures

1-1. User System 1-6
1-2. Example of Memory 1-11
1-3. Byte Notation 1-12
1-4. Word Notation 1-12
1-5. Doubleword Notation 1-12
1-6. Memory Organization 1-13
1-7. Word Storage Format 1-13
1-8. Doubleword Storage Format 1-13
2-1. Physical-Level Structure 2-3
2-2. Delivery-Level Structure 2-4
2-3. Processing-Level Structure 2-5
2-4. Delivery-Management Services 2-7
2-5. System-Management Services Structure 2-8
2-6. Overview of the Delivery Support 2-10
2-7. Delivery-Service Structure 2-11
2-8. Generic Delivery Service 2-13
2-9. Control-Block Delivery Example—Locate Mode 2-15
2-10. Control-Element Delivery Example—Move Mode . . . 2-17

© Copyright IBM Corp. 1991

Notes:

Chapter 1. Introduction

In the personal computing and workstation environments, more and more adapters are becoming functional subsystems that provide more than just device attachment.

The bus master in Micro Channel-based systems is capable of transferring data on the system channel independent of the system processor. This allows adapters, such as LAN interfaces and SCSI controllers, to use bus-master capabilities to perform tasks independently, thereby freeing the system processor to perform other tasks and improving the overall throughput within the system.

The Subsystem Control Block (SCB) architecture enhances the potential of the bus master by defining services and conventions that deliver command information and data to the adapter. The architecture provides command chaining, data chaining, signalling, and synchronization of command and command status; it supports system-to-adapter and adapter-to-adapter (peer-to-peer) operations.

This architecture defines the logical protocols and control structures used to transfer command and control information, data, and status information between the system processor and an adapter, or directly between adapters. The architecture separates the delivery of control information and data to increase performance, raise the level of functional capability, and provide design flexibility.

The SCB architecture is based on three levels: the processing level, the delivery level, and the physical level.

Entities at the processing level can be characterized by the roles they perform. *Client* entities are entities that make requests of *server* entities. A server entity performs the work necessary to satisfy the request and return a reply to the client entity. In some cases, an entity might assume the role of both client and server. In this case, the entity is called an *agent*. An agent takes a request from a client entity and requests the services provided by other server entities to complete the request.

The delivery level provides the set of services and protocols used to deliver requests and replies to the specific functional entities for the

© Copyright IBM Corp. 1991

system processor or an adapter. The architecture consists of two modes of delivery: the Locate mode and the Move mode.

The physical level defines the physical interface for accessing the data and control information in locations within the system address space for I/O and memory. The physical level supports the driving and receiving of buses and signals between adapters on the channel and to and from the system processor. The specific implementation of this level is determined primarily by the design of the system hardware.

Because the interaction between levels is at a logical level, enhancements to an adapter or the base system (such as increasing data rates and changing the data-transfer procedures) will not cause changes to the processing-level support.

Architecture Applications

The SCB architecture is designed to have diverse application and to be used in the following areas:

- Traditional I/O-device protocols.

  These protocols typically have a single system processor, which requests an adapter to perform work on its behalf. With these protocols, requests flow from entities in the base system to entities in the adapter. The adapter can support the attachment of one or more devices.

  In this environment, the Locate mode and the Move mode can be used. In the Locate mode, the address of control information is passed to the adapter; the adapter then uses the address to locate the control block and fetch it into its own storage area for execution. The reply to the system must be synchronized with the request.

  In the Move mode, a copy of the control information is passed directly to the adapter. The reply does not have to be synchronized with the request.

- Peer-to-peer protocols.

  In peer-to-peer protocols, requests can flow from entities in one adapter directly to entities in another adapter, as well as from entities in the base system to entities in an adapter.

  For these environments, the Move mode must be used. The control information is physically moved between the requesting and responding entities through two prearranged paths in memory, called pipes. The Move mode is especially useful for controllers that have high arrival rates of work requests or that run in an asynchronous manner. This mode is also useful in supporting the distribution of process-to-process type functions between the base system and adapters.

- Communication protocols.

  Communication protocols require routing to network servers within an adapter, and they require the handling of replies, which can arrive out of sequence and interspersed with requests. They can also require the ability to handle the movement of large amounts of data at very high speeds.

For communication environments, the method of control delivery used would be the Move mode.

- Response-time-critical applications.

  The Move mode is ideal for application environments where response time is critical, because it allows command and data to flow to and from an adapter, can operate with or without interrupts, and can process multiple requests and replies asynchronously.

Subsystem Control Block Characteristics

The following are some capabilities of the SCB architecture that enhance system and adapter performance in the Micro Channel and bus-master environments:

- Provides a programming model for use with Micro Channel-based systems
  - Provides greater flexibility of design
  - Provides high levels of function distribution
  - Provides for signalling between bus masters
  - Provides a means of chaining commands and data
  - Defines request and reply protocols
    - Provides a means of correlating requests to replies
    - Provides source and destination identification
    - Allows multiple outstanding requests
    - Allows unsolicited operations
  - Allows data and control information to be delivered together or separately
- Supports bus masters on the channel
  - Provides a processor-independent architecture
  - Supports full-duplex or half-duplex operation
  - Supports adapter-to-adapter operation
  - Supports synchronous and asynchronous operations
- Improves performance and throughput
  - Reduces interrupt overhead
  - Reduces the number of separate memory accesses
  - Reduces overhead in passing control information and data
  - Optimizes burst or streaming data capabilities.

System Overview

Systems in the personal computing and workstation environment have evolved to a point where a user system can have adapters operating as simple I/O slaves or as bus masters. These adapters communicate with the base system, using the channel. Adapters with bus-master capabilities can also communicate directly with each other on this same channel.

These adapters can be a standard part of the system or an option that can be added later to make up a user system. An example of the physical structure for a user system is shown in the following figure.

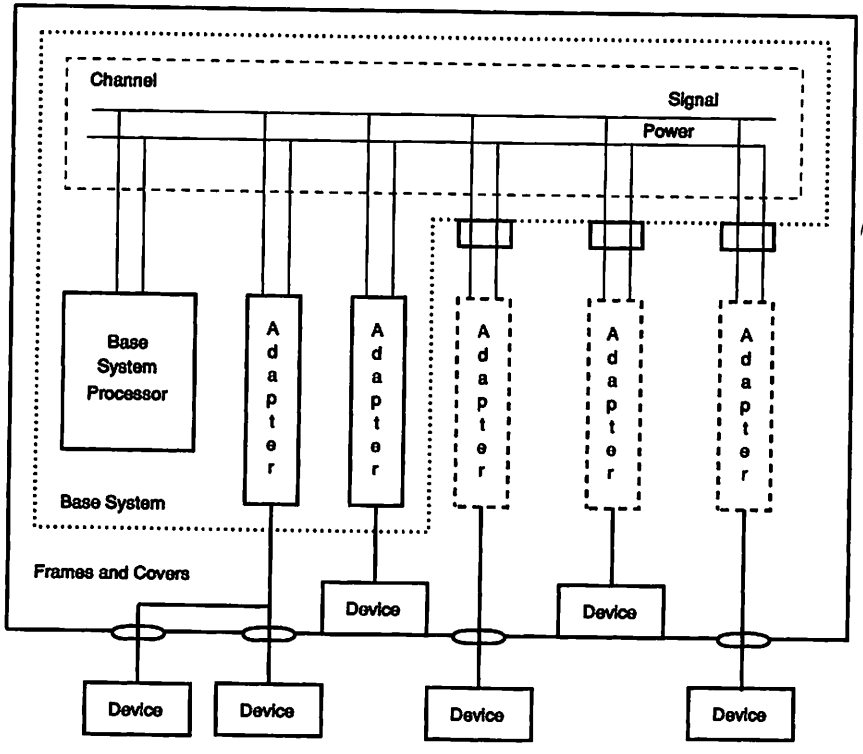


Figure 1-1. User System

In bus-master environments, a high-level, control-block-oriented interface is established between the base system and the adapter or directly between the different adapters.

Bus-Master Characteristics

The following lists some of the characteristics of bus-master adapters:

- An adapter may need to communicate directly with another adapter.
- A single adapter can support the attachment of more than one device. These devices can be of different types and might need to operate asynchronously to the delivery of the control information.
- Adapters with several devices attached can support different forms of function distribution. In one adapter, the function might be distributed using the traditional I/O-device protocol, where the support in the base system manages the devices attached to an adapter.

  In another adapter, the function might be distributed using the peer-to-peer protocol, where the support in the base system and the adapter act as peers, and the adapter itself manages the devices attached to it.

System Resources

This section describes the meaning of the terms byte, word, and doubleword and defines the rules on memory organization. In addition, it describes the use of I/O address space, memory address space, private or local memory, and shared memory.

Because the SCB architecture must work with a variety of system designs, certain aspects of system resources must be defined within the context of the architecture.

Different system hardware designs access I/O address space in different ways. Some designs separate the I/O and memory address space, some have I/O and memory in the same address space, and some designs have separate address spaces but allow I/O to be mapped into the memory address space.

The SCB architecture operates in any of these environments; however, to do that, it places some requirements on adapters and system masters. The architecture requires that all addresses be byte-addressable and defines the I/O addresses used by a given type of adapter.

I/O Address Space

Systems based on Micro Channel architecture have an I/O address space of 64KB.[1] The I/O address space is selected when the signal that selects the address space is in the I/O state. The I/O addresses are mapped into I/O ports or registers, which are assigned to I/O devices, adapters, and system-control functions (such as timers and memory controls). The width of an I/O port defaults to 1 byte; however, it can be a word (2 bytes wide) or a doubleword (4 bytes wide).

Memory Address Space

Systems based on Micro Channel architecture have an address space

---

1 KB = 1024 bytes

of 4GB.[2] The memory address space is selected when the signal that selects the address space is in the memory state. The memory address space contains the memory that the system master can access for instructions and data and use for the transfer of data.

Private or Local Memory

Systems based on Micro Channel architecture can have memory that is not part of either the memory or the I/O address space; this memory is called *local memory*. An example of this private or local memory is shown in Figure 1-2 on page 1-11. In the example, all memory on Adapter C and a portion of the memory on Adapter B is outside the 4GB memory address space; this memory is part of a private bus and is local to that adapter.

Shared Memory

Shared memory is that memory in a system that can be directly accessed by all masters on the channel. This includes all memory within the 4GB memory address space, including system board memory and channel memory. In the example, all the memory on Adapter A and a portion of the memory on Adapter B are within the 4GB memory address space and can be shared by all masters on the channel.

---

[2] GB = 1,073,741,824 bytes

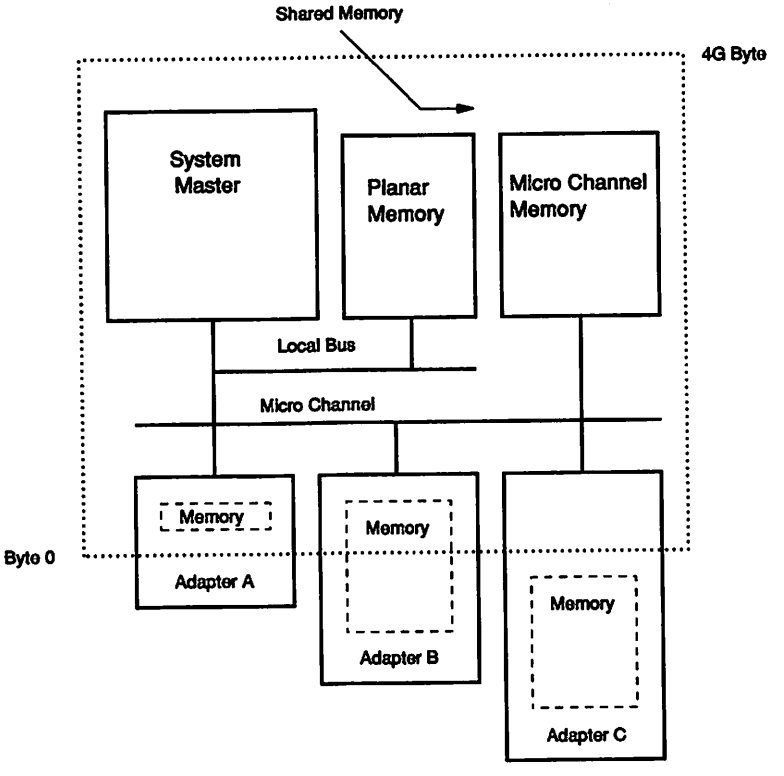


Figure 1-2. Example of Memory

Memory Organization

Since the architecture must work with a variety of system and adapter hardware and software designs in a heterogeneous environment, all data must follow the same mapping and organization conventions to prevent miscommunication between the different components (system units and adapters).

Whether in memory or I/O address space, a *byte* is 8 contiguous bits and must be considered as a single object; the bits are numbered from 0 to 7. Bit 0 is the least-significant bit.

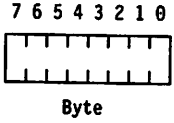


Figure 1-3. Byte Notation

Whether in memory or I/O address space, a *word* is always 16 bits and must be considered as a single object; the bits are numbered from 0 to 15. Bit 0 is the least-significant bit and is in byte 0, which is the least-significant byte (low byte).

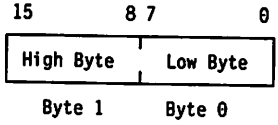


Figure 1-4. Word Notation

Whether in memory or I/O address space, a *doubleword* is always 32 bits and must be considered as a single object; the bits are numbered from 0 to 31. Bit 0 is the least-significant bit and is in byte 0, which is the least-significant byte (low byte).

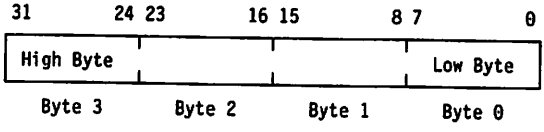


Figure 1-5. Doubleword Notation

When stored in shared memory, all data must follow the same storage conventions. The following shows the mapping and organization of shared memory.

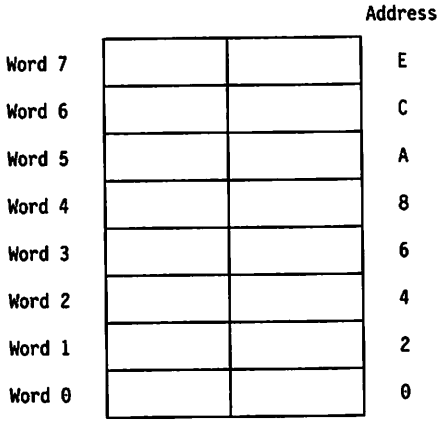


Figure 1-6. Memory Organization

In shared memory, a word must be stored and read as a single object and must be aligned on even address boundaries with the high byte (the byte containing the most-significant bit) in the low-byte position, as shown in the following figure.

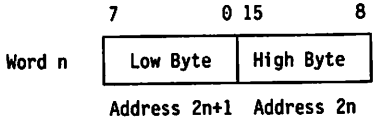


Figure 1-7. Word Storage Format

In shared memory, a doubleword is stored and read in the following format.

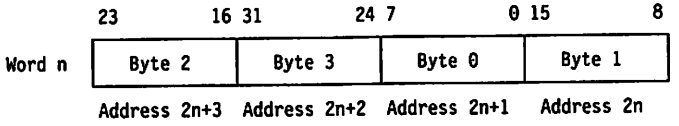


Figure 1-8. Doubleword Storage Format

Notes:

Chapter 2. SCB Architecture Overview

The SCB architecture defines a method of transferring control-related information and data to an adapter that maximizes the throughput and minimizes the time that adapters and the system processor spend managing the transfer. The control-related information is in the form of requests to an adapter and replies from an adapter and includes the asynchronous notification from or to an adapter.

By packaging control-related information in blocks (including commands, options, and data pointers), the architecture allows the individual adapter to control and manage the transfer with minimal or no intervention from the system processor. Each adapter can manage several tasks concurrently, allowing several devices to be operated at the same time by one adapter.

© Copyright IBM Corp. 1991

Logical Levels

Each adapter and system unit is logically structured into three levels: the physical level, the delivery level, and the processing level.

The *physical level* defines a set of services and protocols for accessing control areas in I/O address space, for transferring data and control information into and out of shared memory, and for signalling between the system master and adapters.

The *delivery level* defines a set of services and protocols for managing the delivery of one or more control blocks or elements between the system master and an adapter, or between adapters.

The *processing level* directly supports the exchange of requests and replies between a pair of cooperating entities. One entity, the client, sends requests to the other entity, the server, and receives replies in response.

Physical Level

Services at the physical level are used to pass data and control information at locations within the I/O and memory address space and to support driving and receiving signals between adapters on the channel.

The services provided depend on the mode of delivery.

Push The Push service moves data or control information from a location in the caller's memory to a location in shared memory or I/O address space.

Pull The Pull service moves information from a location in shared memory or I/O address space to a location in the caller's memory.

Signal The Signal service alerts the logic in the system master or adapter that some information needs to be processed.

These services are used by support logic at the delivery level in the base system and adapters to access data and control information in memory or I/O address space and to signal the logic in the base system or another adapter.

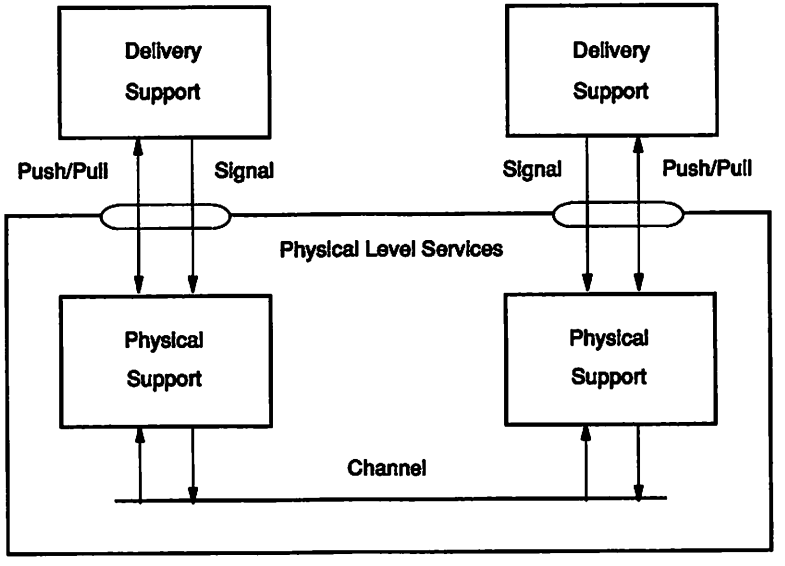


Figure 2-1. Physical-Level Structure

Delivery Level

The delivery level provides a set of services and protocols used to support the delivery of requests from the client to the server and replies from the server to the client. The following figure shows the delivery-level structure.

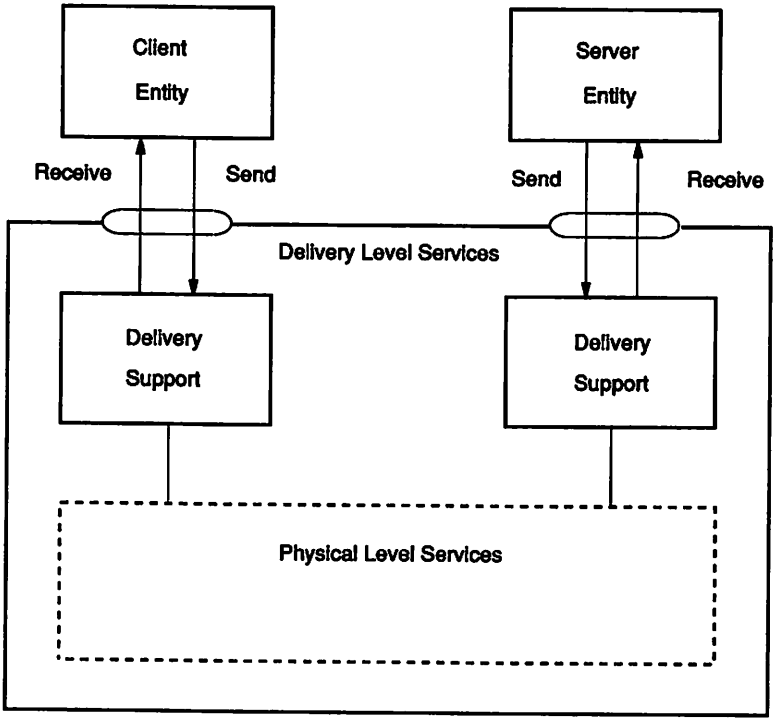


Figure 2-2. Delivery-Level Structure

The delivery level provides the interface between the processing level and the physical level. The interface between the delivery level and the processing level is in the form of a set of primitives for exchanging control-related information, status information, and data. The interface between the delivery level and the physical level is in the form of a set of primitives for accessing the control areas in I/O and memory address space and for signalling between units.

Processing Level

The entity represents the basic functional element supported by the architecture. The client and server operate as an entity pair. Usually, the client is located in one unit and the server in another, although both can be in the same unit.

The following figure shows the processing-level structure. The protocol between entity pairs is concerned with the format and content of the requests and replies. Requests are always from the client, and replies are always from the server. The processing level uses the services provided by the delivery level to manage the exchange of requests and replies between the client and server.

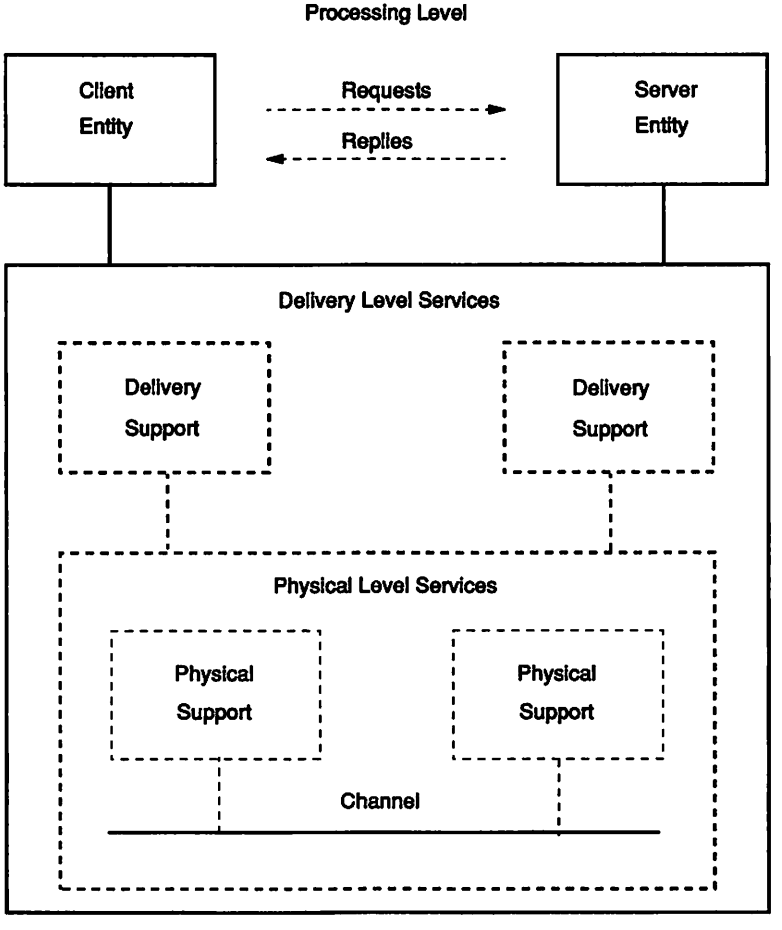


Figure 2-3. Processing-Level Structure

Client

The client entity is the entity that uses or requests services provided by another entity. An example of a client is a program that is used to initiate the reading or writing of data to or from a disk drive.

Server

The server is the entity that provides services to another entity. An example of a server is the SCSI adapter.

Management Entity

The SCB architecture requires that an entity in the base system and in each adapter provide delivery-management support for that unit. This management entity is shared by all entities in that unit. The following figure shows the unit-level management entity relative to the delivery and physical levels.

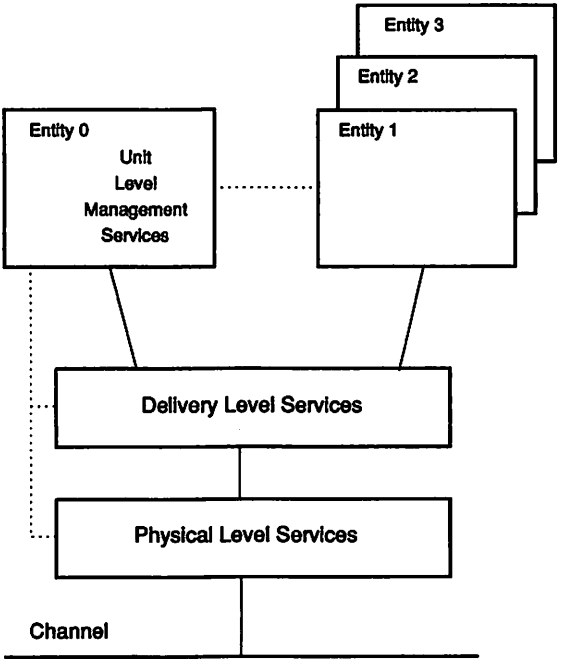


Figure 2-4. Delivery-Management Services

The SCB architecture requires that one entity in the system act as the focal point for all the unit-level management entities in the individual units. This entity is called the *system-level management entity*. The system-level management entity can physically reside in any unit (see Figure 2-5 on page 2-8).

The unit-level management entities in the individual units (system or adapter) communicate with the system-level management entity. The services provided by the delivery level are used to exchange requests and replies.

This management structure supports management services of various types and is not limited to the delivery service. In other words, the management entity can be used for entity-level reporting and testing, as well as for delivery-level and physical-level management.

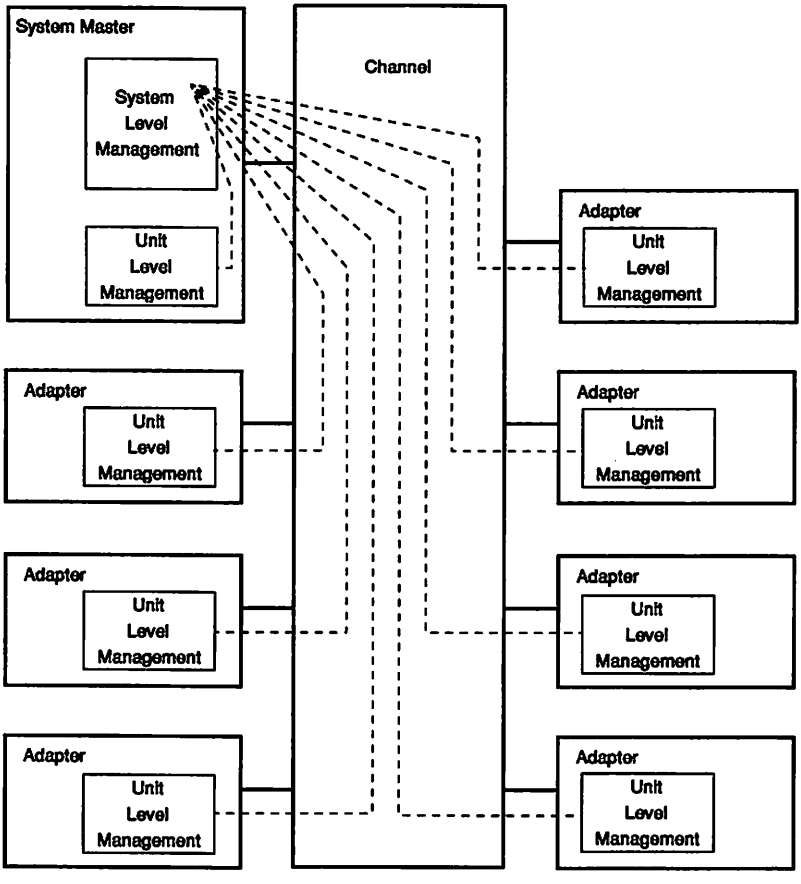


Figure 2-5. System-Management Services Structure

The management structure represents a form of client-server support between the system-level management entity and other unit-level management entities. The management services for a system are hierarchical, even though the support for entity-to-entity delivery supports peer-to-peer protocols.

Delivery-Service Structure

The SCB architecture defines protocols that support the physical delivery of command- and control-related information between a pair of entities. These delivery services are used with Micro Channel-based systems to enhance the operation of adapters operating as bus masters and to separate the physical implementations of Micro Channel procedures from the logical operations on the adapter.

The terms used in describing this architecture are similar to those in the Micro Channel architecture. The various adapters that have bus-master or I/O-slave functions are referred to as adapters, regardless of whether they are packaged as adapters that plug into a channel connector or are physically located on the system board. The system processor and its supporting logic are referred to as the system unit.

The architecture separates the delivery of control information from the delivery of data and separates the delivery of control information from the processing of control information (see Figure 2-6 on page 2-10).

The delivery service provides delivery of the control information and is used by an entity in either the system unit or an adapter to communicate the information needed to perform an operation to an entity in another adapter. These entities build and process the control information in the form of control blocks. The delivery service is responsible for delivering control blocks between cooperating entities.

The information in a control block identifies whether the operation specified in the control block also involves the transfer of data between the two entities. The delivery of data is independent of the delivery of command and control information, and the individual entities are responsible for initiating and completing its delivery.

Also, configuration information is required during system initialization to tailor the control and data-delivery services to support a particular operational environment.

The following figure shows an overview of the control and data-delivery support. It also shows that configuration information is needed to support the operations for both control and data delivery.

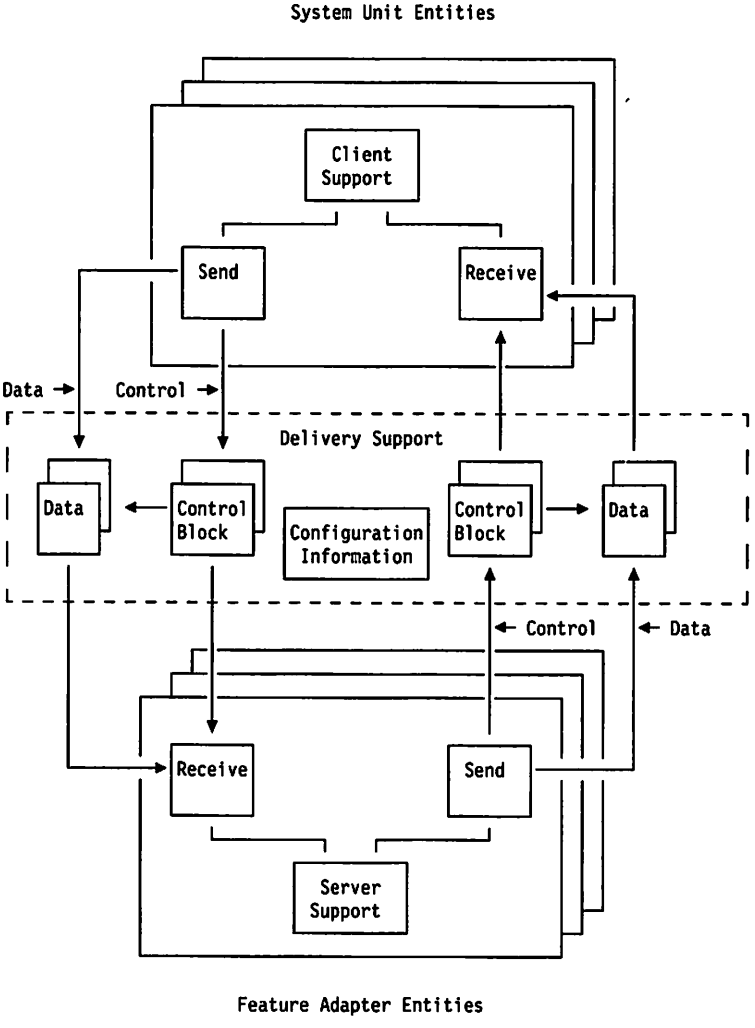


Figure 2-6. Overview of the Delivery Support

Delivery-Service Levels

The following figure shows how the delivery services in different units interact to provide the overall architecture.

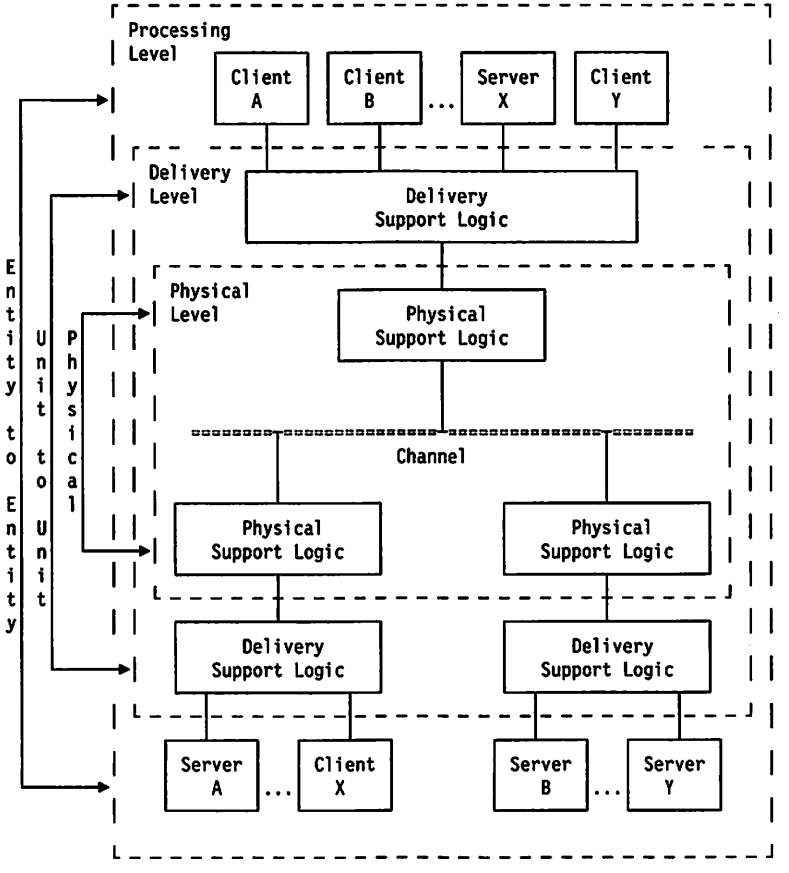


Figure 2-7. Delivery-Service Structure

Because of the structure of the delivery service, the various entities at the processing level can share a common delivery-level support. Additionally, a common delivery-level support can be mapped to different physical-level support if an adapter can be used with different system hardware.

The delivery-level protocols use the I/O address space and shared memory to pass command and control information between units.

The processing-level protocols can also use shared memory to pass data (pointed to by control blocks).

Delivery Services

In a simple form of control delivery, the control information is passed between a client in the system unit and a server in an adapter. In more sophisticated forms, the client or server can be either in the system unit or in the adapter (the client in one location always makes requests of the server in the other location).

The delivery service itself is distributed among all adapters and the system master. The part of the delivery service that is local to each is the delivery agent. The delivery agents in the various adapters communicate with each other using delivery protocols that are built on top of physical-level services, which interface with the channel. (The channel provides access to memory and I/O space in the system unit and the adapters, using various Micro Channel procedures.) A simple view of the delivery service is shown in the following figure.

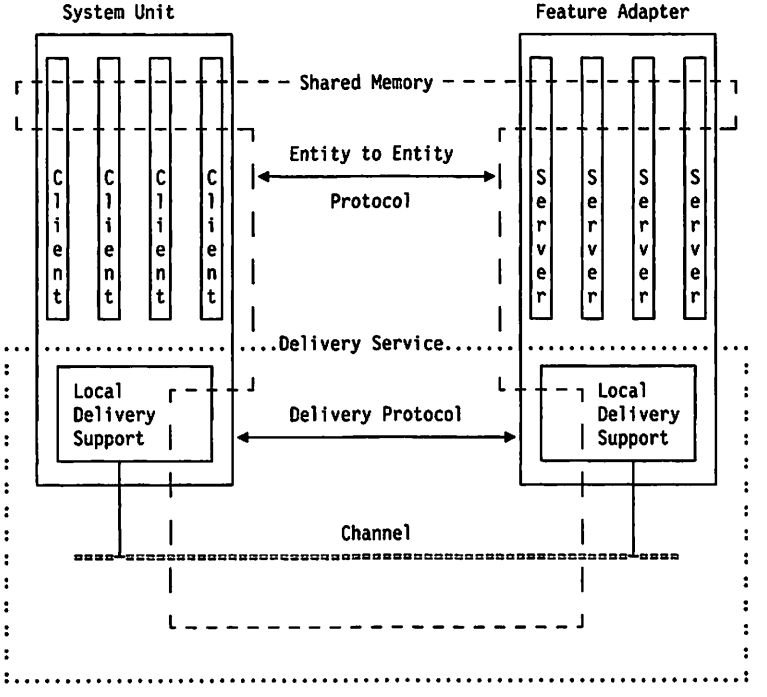


Figure 2-8. Generic Delivery Service

The delivery service supports the delivery of control blocks between pairs of entities (a client and a server). These entities build and interpret the control blocks. The entities themselves determine the actual protocol used between client and server; the overall operation at the processing level determines the services needed from the delivery level and the protocols needed to support the distribution of these services.

The delivery service is logically structured into processing-level and physical-level support. The processing-level support provides services for delivering control blocks between entity pairs. The physical-level support provides access to I/O and memory address space and provides a means for signalling between the adapters and the system unit. The physical-level support contains both support logic and Micro Channel-related hardware.

Delivery Modes

The SCB architecture defines two methods of delivering command and control information: the Locate mode and the Move mode. The following provides a brief description of the operations of each mode and the underlying control structures.

Locate Mode Delivery

In the Locate mode form of control-block delivery, the control structure is a relatively fixed format. The structure allows command and control information, status information, and pointers to data blocks to be passed from a client in the system unit to a server in the adapter. The data block pointed to by the control block can contain data or a list of data blocks (indirect list).

To send a request using control blocks, the system unit passes the physical address of the control block and a device identifier to the adapter through the control areas in I/O address space.

The operation of the Locate mode is serial and requires that the entity pairs complete one request before another request is sent to that entity. A request can be a single control block or a group of control blocks that have been chained together. The operation of the Locate mode is shown in the following figure.

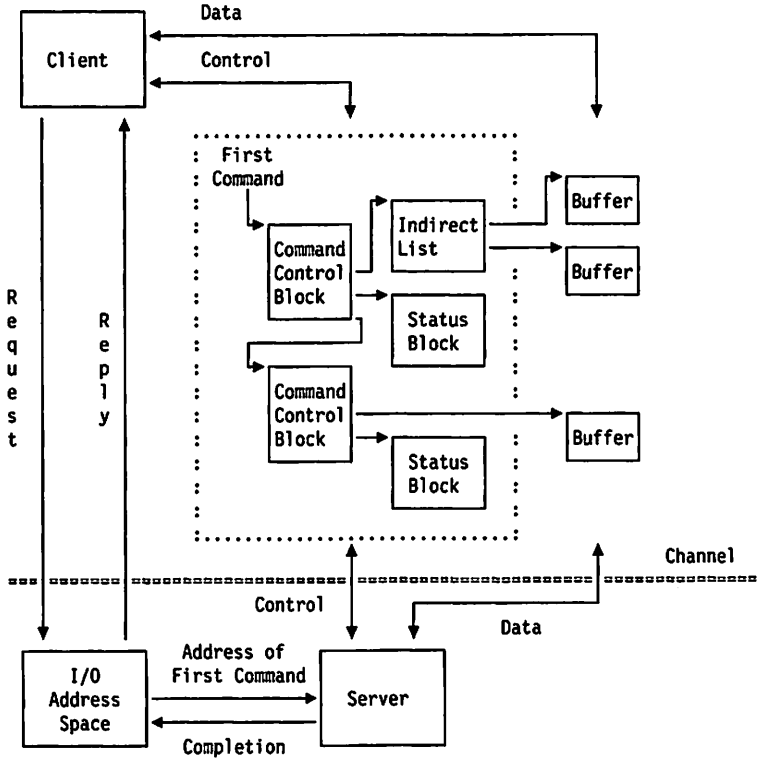


Figure 2-9. Control-Block Delivery Example—Locate Mode

Move Mode Delivery

The Move mode supports a control delivery using control elements to deliver control-related information between the client and the server. The control elements are variable in length and can contain requests, replies, error notifications, or event notifications for a specific entity or for different entities.

The entity-level operation of the Move mode is similar to that of the Locate mode (the client builds requests, requests are delivered to the server, the server builds replies, and replies are delivered to the client). However, in the Move mode, clients can be in the system unit or in adapters, which means a client in an adapter can send requests directly to a server in another adapter without having to go through the system unit. Additionally, the server can be in the system unit, which means that the system unit can receive requests from an

adapter. These capabilities provide a peer-to-peer relationship between adapters and the system unit.

The Move mode uses control elements instead of control blocks. The control element is used by the client to deliver the request, and it is used by the server to deliver the reply.

The control elements are moved between entity pairs using a pair of delivery pipes; each pipe behaves as a first-in-first-out queue. Each pipe allows for the delivery of control elements in only one direction; together, the two pipes provide full-duplex operation and allow multiple control elements to be delivered and processed asynchronously by each entity.

The operation of the Move mode is shown in the following figure.

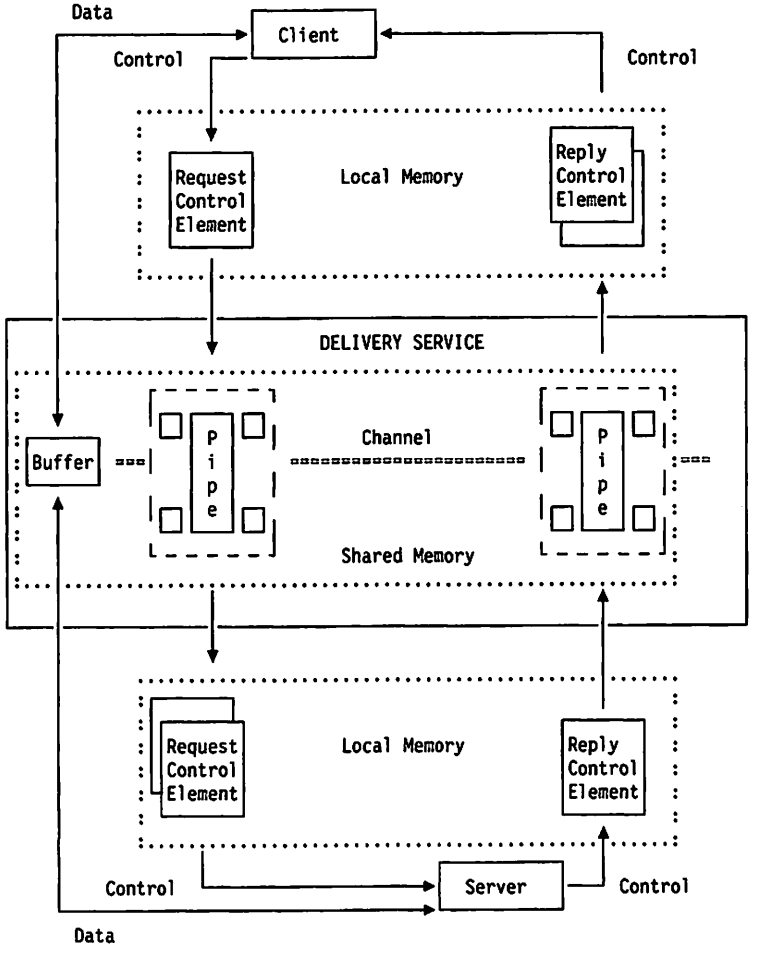


Figure 2-10. Control-Element Delivery Example — Move Mode

Control Areas

The SCB architecture defines and identifies specific control areas that are used when interfacing with the adapter or system unit.

Ideally, there should be only one physical interface used to interface with any adapter on the channel; however, conflicting factors (such as differences in expected performance, implementation cost, complexity of implementation, and ever-improving technology) lead to the need for the architecture to support alternative interfaces.

In the Locate mode, the control areas are a set of ports in the I/O address space that are used to control and report the status of the physical interface and to deliver the control and status information between entities.

The memory address-space address of the control block containing the command is passed from the entity in the system unit to the entity in the adapter, using a port in the I/O address space. The entity in the adapter uses this memory address-space address to fetch the control block and perform the indicated operation. Upon completion, the status is put into the status block, and the entity in the system unit is notified, using a port in the I/O address space.

In the Move mode, the control areas are a set of ports in the I/O address space and a pair of delivery pipes in shared memory. Shared memory is located in the memory address space. The I/O ports are used to control and report the status of the physical interface; the delivery pipes are used to deliver the control and status information.

The control elements carrying the requests and replies are moved between entity pairs, using a pair of delivery pipes maintained in shared memory. The delivery pipe behaves as a first-in-first-out queue, which makes the control elements appear to be contiguous. The pipe allows multiple control elements to be grouped into a single request.

Using delivery pipes allows multiple control elements to be delivered and processed asynchronously by each entity.

Shared Memory

The SCB architecture defines and uses areas in shared memory to maintain control structures used to support the delivery of command control information, and data between the system unit and adapters or directly between peer adapters.

The mapping and organization of shared-memory space is described in "System Resources" on page 1-9. This memory space is directly accessible by the system unit and adapters with bus-master capabilities.

In the Locate mode, shared memory is used to hold control blocks, indirect lists, termination status blocks, and data.

In the Move mode, shared memory is used as an area to deliver control elements (request, reply, error, and event) through the two delivery pipes, to manage the delivery through these pipes, and as a signalling area between the system unit and adapters. It is also used to hold data.

Structures

The Locate mode uses the command control block, the termination status block, and indirect lists; the Move mode uses the control element. All of these structures are variable-length structures in shared memory that are passed between entity pairs.

Control Blocks

Control blocks are used to pass commands and control information from a client to a server. Control blocks contain the commands and parameters to be passed to the server in an adapter. The parameters describe the particular options for the operation to be performed.

Some guidelines and rules for control blocks are:

- Control blocks must be aligned on doubleword boundaries.
- Control blocks must be placed in shared memory in the format defined for shared memory.
- Addresses in the control blocks must be physical addresses.
- Control blocks can be chained together.
- Entities need to read only those fields in the control blocks that they use.

Termination Status Blocks

Termination status blocks are used to report status information for each control block processed. The termination status block is used to report error and exception status that occurs during control-block processing. A field in the control block points to the termination status block.

Indirect Lists

An indirect list is a variable-length list that is used in the Locate mode and consists of a series of address and count fields used for data chaining. The address fields and count fields are 4-byte fields that specify the location and size of the data blocks. Both the location of the indirect list and its length are specified in the control block.

Control Elements

The Move mode uses control elements to exchange control information between a client and a server. Control elements are like control blocks; however, they differ in that:

- Control elements are variable in length.
- Control elements are self-describing.
- Control elements provide a means of specifying the destination, identifying the source, and indicating the type and urgency of the control information they contain.
- Control elements can contain data as well as control information.
- Control elements contain information for correlating requests with replies.
- Control elements can be processed asynchronously.

The following are some rules for control elements:

- Control elements are aligned on doubleword boundaries.
- Control elements must be placed in shared memory in the format defined for shared memory.
- Addresses in the control elements must be physical addresses.

Chapter 3. Subsystem Control Block Capabilities

This section provides some criteria to help in deciding which form of the Subsystem Control Block Logical I/O architecture an implementation chooses to support.

Locate Mode

The Locate mode provides support for:

- Multiple devices per adapter.

  The adapters can support multiple entities (devices and other resources). Some examples of entities are small computer system interface (SCSI) devices, LAN connections, X.25 virtual circuits, Integrated Services Digital Network (ISDN) channels, communication lines, and processes. The Locate mode supports the delivery of requests to specific entities through a device identification number (ID).

- Adapter management.

  The adapter can perform operations that involve the function of the entire subsystem and its attached devices. The management entity is assigned device identification number 0 and receives all adapter management-type information.

- Requests to devices.

  To use a device or resource, a program (client) sends requests in the form of control blocks to the device ID that represents the device or resource (server) and receives replies for those requests.

© Copyright IBM Corp. 1991

- Command and data chaining and detailed status.

  The control structure defined for the Locate mode provides for an immediate-command type of request, a request that has one command control block and status block and a request made up of multiple command control blocks chained in a specific order and treated as one logical request. Command control blocks can point to data directly or via an indirect-list control block.

  The structure also provides for handling status information in case of exceptions during the processing of a request. The status is placed in a termination status block. In order to handle termination at any point in a chain, a termination status block is pointed to by each command control block in a chain.

- Use of direct memory access (DMA).

  The Locate mode defines the interface to transfer both the control structure for a request and the data associated with the request between the system unit and an adapter. The transfer uses DMA operations managed from the feature adapter's delivery support and enity support.

- Interrupts.

  The Locate mode allows each request to define when and under what conditions interrupts are generated. During normal, error-free operation, an interrupt is optionally generated after each request is completed. Additional interrupts can be requested to synchronize the delivery of a chain of control blocks at intermediate points within the request.

  The flow of the interrupt processing in a system is from the hardware to the operating-system kernel to the delivery-interrupt handler to the delivery-request delivery support. The interrupt processing uses status information in the I/O space, in the local state control areas, and in the termination status blocks to determine the cause of the interrupt.

Move Mode

The Move mode provides support for:

- Request and reply extensions.

  Request and reply extensions to the control element provide for delivery of event and error information, in addition to delivery of request and reply information. Unlike the control elements for requests, errors, and replies, the control element for events can flow in either direction.

- Shared memory.

  Shared memory allows for the use of memory in adapters as well as memory in the system unit. This allows the control elements to be in either the system unit or the adapter and allows for more flexibility in how the control structure is built and moved to the destination entity. (The destination entity is the entity receiving the control element. The server is the destination entity for requests, and the client is the destination entity for replies.)

- Variable-length requests and replies.

  The Move mode provides a control structure that allows a request to be made up of a set of variable-length control elements. At the interface to the entity, these control elements appear to be contiguous. This allows for chaining of control elements within a request. It also allows for a minimum-size control element for passing simple requests and replies.

  Variable-length control elements address the need to have smaller control structures, to be able to pass a variable number of request parameters, and to have a simple way to support a number of different subsets. Some client/server entity pairs might choose to use complex combinations of control elements while others use a simple set. They are all implemented from the same control-element structure and use the same set of delivery support primitives.

- Lists of requests and replies.

  Figure 2-10 on page 2-17 shows an example of control elements flowing from a client to a server and another flowing from a server to a client. Both flow on the shared-memory delivery pipes.

The delivery pipes defined for the Move mode control-element delivery service have the following attributes:

- Full duplex.
- Multiplexing of entity pairs.

  Each pipe can have control elements for multiple entity pairs in the same pipe. The control structure provides for source and destination identification in the control elements to allow a set of control elements for different entity pairs to be delivered in the same pipe.
- Intermixing of requests and replies.

  The control structure supports the intermixing of requests and replies as well as other control-element types (error and event) in the same pipe.
- Continuous running.

  The control structure permits the delivery of control elements in a continuous flow. Mechanisms are defined to provide a common way for entities to suspend the delivery of control elements at the entity-to-entity level, to notify the destination entity that one or more control elements are available, and to provide for synchronization between the entities in the source and the destination units.
- Peer-to-peer support.

  An entity in a system unit or feature adapter can send requests directly to, and receive replies directly from, any other system unit or feature adapter attached to the channel without having to go through a third party. This means an entity in a feature adapter can send requests and receive replies directly from an entity in another feature adapter without having to go through a system unit.

Selecting the Appropriate Mode

Three architecture designs are available:

- Locate mode.

  This refers to both commands and delivery as defined in the Locate mode.

- Move mode.

  This refers to control elements and delivery as defined in the Move mode. It excludes control elements used to deliver commands in the form of control blocks.

- Combined modes.

  This refers to using some of the features defined for the Locate mode in combination with features defined for the Move mode.

When to Use the Locate Mode

Locate mode commands and delivery are most useful when the following conditions exist:

- Command flow is from a single system unit.

  In the single system-unit configuration, command flow is from the system unit to the subsystem. Full-duplex operation is not needed.

- Command requests and replies do not require high speed.

  The rate at which requests are sent to the subsystem does not require high-speed transfer to maintain a timely response. The rate at which interrupts or replies are generated by devices attached to the subsystem, per command or command chain, is low to moderate.

- The command and reply format is fixed in size.

  There is no special need to deal with a wide range of command and reply sizes. Control information comes in a smaller range of sizes.

- Command chains are not modified dynamically.

  Command chains do not need to be dynamically modified as they are executed by a device, such as with the Notify and Wait facilities of the Move mode.

- The number of devices attached is small.

  The number of devices attached to the subsystem is small, typically in the range of 1 to 15 devices.

- Peer-to-peer operations are not needed.

  Requests and replies are not sent directly to another subsystem or adapter on the channel. All work requests and replies flow from the system unit to a particular subsystem.

When to Use the Move Mode

The Move mode should be selected over the Locate mode, unless compatibility with the Locate mode command set is required. This recommendation is based upon the following considerations, which were developed by contrasting the Move mode and the Locate mode:

- The Move mode is more flexible.

  The Move mode allows greater flexibility in the following areas:

  - Variable-length control elements can be defined.

    A Length field is included in control-element headers at a standard location to simplify the handling of variable-length control elements.

  - Control elements can contain immediate data.

    Control elements are defined to allow reading and writing of data from or to control elements. This is useful when small amounts of data need to be intermixed with control flows, because a separate data operation is not needed.

  - Implementations can develop their own control elements.

    Control elements have a type code, which allows an implementation to define its own control elements. By defining their own control elements, designers can match their designs to specific implementation needs.

- Servers can send multiple replies from the same request.

  A server in the Move mode can send back multiple replies or events from a single request, which is useful for operations that are performed in stages.

- Location of delivery space is flexible.

  Move mode architecture has the ability to locate the memory for delivery pipes in either the destination or source unit. This gives greater flexibility in adapter design.

- The Move mode offers better potential performance.

  Move mode architecture offers better potential performance than does Locate mode architecture, for the following reasons:

  - More parallel execution between units is allowed.

    Enqueue and dequeue functions are defined that allow requests to be queued for processing. This allows for more parallel execution between units.

  - Client serialization is not required.

    In the Move mode, a client thread does not need to serialize all access to a subsystem after placing a control element in the delivery pipe. The pipe is free flowing, and as soon as the element is placed in the pipe, the client is free to perform other work. This allows more parallel processing to occur.

  - Multiple control elements can be processed on an interrupt.

    In the Move mode, clients and servers can process control elements without requiring individual interrupts for each one. This is an important performance benefit because processing an interrupt usually carries a heavy penalty.

  - Several requests can be handled at the same time.

    A Move mode server can be defined to handle multiple work requests at the same time. The client does not have to wait for the server to complete a request before submitting other requests. Requests can be queued at the server.

  - Less data is required per control element.

    Generally, Move mode architecture requires less data to be transferred to process a unit of work. This gives better performance.

- Correlation of commands is supported.

  The Move mode provides better support for programs to correlate a reply to a specific command and program thread. This is done through a Correlation ID field, which is included in all control elements, and which can be used for programming-defined purposes. Fewer instructions are used per reply.

- Handling is expedited.

  The Move mode provides an expedited-handling flag in control elements. This allows an implementation to get performance- or response-critical elements to the attention of the server.

- Control-element execution is optimized.

  A Move mode server is allowed to optimize the processing order of control elements outside of command chains. This gives a potential for improved performance.

- Full-duplex operation is allowed.

  The Move mode entities can process inbound and outbound requests independently. This gives a greater potential for parallel execution.

- Real-time command chains are allowed.

  The Move mode use of notify and wait flags allows the programming of command chains that are more real-time responsive. This improves the response time of a system using the Move mode.

- The Move mode provides improved peer support.

  Move mode architecture supports improved peer capability when compared to the Locate mode architecture, for the following reasons:

  - The Move mode supports replies to peer sources.

    Move mode control elements contain the source address of the peer unit in the control-element header. This allows a reply to be sent directly to the peer unit that originated the request.

- The Move mode supports requests and replies on a single pipe.

  In the Move mode, a single delivery pipe can contain request and reply control elements. This is needed in a peer-to-peer environment where work flows depend on transaction type. That is, a unit can be a server for one form of work request, and it can be a client for others.

When to Use the Combined Modes

No detailed description of combined modes is presented in this book. Each implementation must define an appropriate combination of Locate mode and Move mode features.

Modes can be combined, for example, for upward compatibility from adapters that implement Locate mode commands and wish to provide an implementation bridge so that Locate mode commands can be used in a better, more responsive way. This is done using Move mode command delivery with the Locate mode command set. In this case, the following improvements are realized:

- Performance is improved over the Locate mode, for the following reasons:
  - Less serialization is needed at the client.

    Less synchronization is needed at the client to send a request to a subsystem than is required in the pure Locate mode. The busy time in the sending unit is reduced using Move mode delivery in the combined modes.
  - Less serialization occurs on replies.

    The use of a Move mode delivery pipe instead of a single Interrupt Status port means that less serialization of reply processing occurs at both the client and the server. This means more parallel execution is possible.

- More data is given with replies.

  The use of Move mode delivery means that a correlation ID, as well as the physical address of a control-block command can be supplied with each reply. The additional data makes it easier to determine which specific command and execution thread should handle the reply. Fewer instructions per reply means better performance.

- Control-element execution is optimized.

  A server using Move mode delivery is allowed to optimize the processing order of control elements outside of command chains. This gives a potential for improved performance.

- Dequeue synchronization with enqueue is minimized.

  An entity using Move mode delivery dequeues control elements from a delivery pipe with a minimum amount of synchronization between itself and entities placing control elements in the pipe. This allows for more parallel execution between units.

- Multiple control elements can be processed on an interrupt.

  In Move mode delivery, clients and servers can process control elements without requiring individual interrupts for each one. This is an important performance benefit because processing an interrupt usually carries a heavy penalty, especially in processors that are pipelined.

- Handling is expedited.

  Move mode delivery provides an expedited-handling flag in control elements. This allows an implementation to get performance- or response-critical elements to the attention of the server.

- Full-duplex operation is allowed.

  Entities using Move mode delivery can process inbound and outbound requests independently. This gives greater potential for parallel execution.

- Real-time command chains are allowed.

  The use of notify and wait flags in control elements allows the programming of command chains that are more real-time responsive. This improves the response times of a system using Move mode delivery.

- Functionality might be improved in the following areas:

  - Peer support is better.

    An adapter is able to serve multiple clients without going through a single system unit because of the peer nature of the Move mode delivery service. This can be a performance boost in a LAN bridge or multimedia system where DASD data, for example, needs to be processed by a peer adapter and does not need to be processed by a system unit.

  - Expansion is easier.

    It would be easier to support the command queuing proposed by the SCSI II standard, given the free-running nature and additional data provided by the Move mode delivery service.

Notes:

Index

A

adapter 2-9
adapter management 3-1
agent 1-1
architecture applications
  communication protocol 1-3
  I/O-device protocol 1-3
  peer-to-peer protocol 1-3
  response-time-critical application 1-4

B

bus-master characteristics 1-8
byte 1-12

C

capabilities, SCB 1-5
characteristics, bus-master 1-8
client entity 1-1, 2-6
combined modes, when to use 3-9
communication protocol 1-3
control block 2-20
control element 2-15, 2-16, 2-21
control-block delivery 2-14

D

data chaining 3-2
delivery agent 2-12
delivery level 1-1, 2-3
delivery modes 2-2
  pull 2-2
  push 2-2
  signal 2-2
delivery pipe 2-16, 2-18
delivery pipe, attributes of 3-4
delivery service 2-9, 2-14
design requirements, system 1-9
devices, requests to 3-1
direct memory access 3-2
DMA 3-2
doubleword 1-9, 1-12

E

entities
  agent 1-1
  client 1-1, 2-6
  management 2-6
  server 1-1, 2-6
extensions, request and reply 3-3

F

full duplex 3-4

I

indirect list 2-20
interrupt 3-2
I/O address space 1-9, 2-18
I/O port 2-18
I/O-device protocol 1-3

L

least-significant bit 1-12
least-significant byte 1-12
local memory 1-10
locate mode 1-2, 1-3, 2-14
locate mode, when to use 3-5
logical levels 2-2
  delivery level 2-3
  physical level 2-2
  processing level 2-4

© Copyright IBM Corp. 1991

low byte 1-12

M

management entity 2-6
  system-level 2-7
  unit-level 2-6
memory address space 1-9
memory address-space address 2-18
memory organization 1-12
move mode 1-2, 1-3, 2-15
move mode, when to use 3-6
multiplexing 3-4

O

organization, memory 1-12

P

peer-to-peer protocol 1-3
physical level 1-2, 2-2
physical-level support 2-13
pipe, delivery 2-16, 2-18
private memory 1-10
private or local memory 1-10
processing level 2-4
processing-level support 2-13
pull 2-2
push 2-2

R

request and reply extensions 3-3
requests and replies, variable-length 3-3
requests to devices 3-1
requirements, system resource 1-9
response-time-critical application 1-4

S

SCB capabilities 1-5
server entity 1-1, 2-6
shared memory 1-10, 1-13, 2-18, 2-19, 3-3
  doubleword storage format 1-13
  word storage format 1-13
signal 2-2
structures
  control block 2-20
  control element 2-21
  indirect list 2-20
  termination status block 2-20
system overview 1-6
system requirements 1-9
system resources
  I/O address space 1-9
  memory address space 1-9
  private or local memory 1-10
  shared memory 1-10
system unit 2-9
system-level management entity 2-7

T

termination status block 2-20

U

unit-level management entity 2-6

V

variable-length requests and replies 3-3

W

word 1-9, 1-12

Locate Mode

Chapter 1. Overview 1-1
Operational Characteristics 1-3
Command and Control-Block Delivery 1-6
Control-Block Structure 1-10
Termination Status Block Structure 1-11
Indirect-List Structure 1-12
Delivery Structure 1-13
I/O Address Space 1-15
Physical Interface Description 1-15
Port Descriptions 1-18
Command Port 1-18
Attention Port 1-18
Subsystem Control Port 1-20
Interrupt Status Port 1-21
Command Busy/Status Port 1-24
Device Interrupt Identifier Port 1-28
Memory Address Space 1-29
Control Structures 1-29
Command Control Blocks 1-29
Command Word 1-31
Enable Word 1 1-33
Memory Address 1 1-39
Memory Address 2 1-39
Byte Count 1 1-40
Termination Status Block Address 1-40
Chain Address 1 1-40
Chain Address 2 1-41
Device Dependent Size 1-41
Chain ID 1-41
Enable Word 2 1-41
Byte Count 2 1-42
Termination Status Blocks 1-43
End Status Word 1 1-46
End Status Word 2 1-50
Residual Byte Count 1 1-50
Residual Buffer Address 1 1-51
Residual Byte Count 2 1-51
Residual Buffer Address 2 1-52
Device Dependent Data Size 1-53

© Copyright IBM Corp. 1991

Device Dependent Data Area 1-53
Indirect-List Structure 1-53
Commands 1-55
Control-Block Commands 1-55
Initialize Device 1-55
Options 1-57
No Operation 1-57
Read 1-58
Read Completion Status 1-60
Read Configuration 1-63
Run Diagnostic Test 1-70
Write 1-71
Immediate Commands 1-72
No Operation (Noop) 1-76
Reset Device 1-77
Reset Interrupt Status Port 1-79
Reset Control Block Interrupt 1-80
Reset Subsystem (Software Controlled) 1-82
Resume 1-86
Run Immediate Diagnostic Test 1-87
Suspend 1-88
Reset Subsystem (Hardware Controlled) 1-90

Chapter 2. Physical Level 2-1
Structure 2-1
System-Master Hardware Support 2-3
Subsystem Hardware Support 2-3
Support Logic 2-4
Memory Address Space 2-4
I/O Address Space 2-4
Physical-Level Services 2-6
Push and Pull 2-6
Interrupt 2-8
Data Delivery 2-9
Physical-Level Protocols 2-9

Chapter 3. Delivery Level 3-1
Structure 3-1
Immediate-Command Delivery 3-4
Control-Block Command Delivery 3-6
Delivery-Level Services 3-7
Immediate-Command Delivery Service 3-7
Hardware-Controlled Command Delivery Service 3-8

Control-Block Delivery Service 3-9
Delivery-Level Protocols 3-10
Command-Delivery Protocol 3-10
Signalling Protocol 3-13
Interrupt Identification 3-14
Physical and Logical Interrupts 3-14
Physical-Interrupt Enablement 3-15
Interrupt Queuing within the Subsystem 3-16
Interrupt Types 3-16
Non-DIIP Interrupts (Simple Form) 3-16
DIIP Interrupts (Multiple Form) 3-26
Resetting Interrupts 3-38

Chapter 4. Processing Level 4-1
Command Chaining 4-2
Conditional Command Chaining 4-5
Command-Chaining Options 4-7
Control-Block Execution 4-9
Data Chaining 4-10
Reading with a Base Control Block and Indirect Lists . . . 4-12
Reading with an Extended Control Block and Indirect Lists 4-14
Writing with a Base Control Block and Indirect Lists 4-14
Writing with an Extended Control Block and Indirect Lists 4-16
Writing Physical Data Blocks and Indirect Lists 4-16

Chapter 5. Design Considerations 5-1
Locate Mode Setup and Initialization 5-1
Locate Mode Configuration 5-2
Locate Mode Ports 5-2
Device Interrupt Identifier Port 5-2
Obtaining Configuration Information 5-3
Exception and Error Handling 5-4
Device Level 5-4
Subsystem Level 5-6
Exception Conditions 5-9
Short-Length Exception 5-9
Long-Length Exception 5-11
Specification Exception 5-13
Device-Overrun Exception 5-14
Major Error on Data Transfer 5-16
Locate Mode Residual Data Values 5-19
Residual Byte Count 5-19
Residual Buffer Address 5-20

Number of Bytes Transferred 5-21
Summary – Residual Buffer Address and Residual Byte Count 5-22

Index X-1

Figures

1-1. Control-Block Delivery Structure Example 1-1
1-2. Request-Delivery Flow 1-7
1-3. Control Block Format 1-11
1-4. Termination Status Block Format 1-12
1-5. Indirect-List Format 1-12
1-6. Locate Mode Delivery Structure 1-14
1-7. I/O Address Map—Type 1 1-16
1-8. I/O Address Map—Type 2 1-17
1-9. Command Port 1-18
1-10. Attention Port 1-19
1-11. Attention Codes 1-19
1-12. Subsystem Control Port 1-20
1-13. Interrupt Status Port 1-22
1-14. Interrupt-Identifier Codes 1-22
1-15. Command Busy/Status Port 1-25
1-16. Command Busy/Status Port—Status Bit Encoding . . . 1-25
1-17. Busy-Bit And Reject-Bit Settings 1-27
1-18. Device Interrupt Identifier Port 1-28
1-19. Command Control Block Structure 1-31
1-20. Command Word Format 1-31
1-21. Architected Op Codes 1-32
1-22. Enable Word 1 Format 1-33
1-23. Memory-Addressing-Control Encoding 1-36
1-24. Chain-Condition-Specification Encoding 1-37
1-25. Chaining-Bit Settings 1-38
1-26. Enable Word 2 Format (Extended Command Control Block) 1-41
1-27. Termination Status Blocks 1-43
1-28. Base Termination Status Block 1-45
1-29. Extended Termination Status Block 1-45
1-29. Extended Termination Status Block 1-45
1-30. End Status Word 1 Format 1-46
1-31. TSB-Available Bit Encoding 1-48
1-32. End Status Word 2 Format 1-50
1-33. Indirect-List Format 1-54
1-34. Initialize Device Control Block 1-56
1-35. No-Operation Control Block 1-58
1-36. Read Control Block 1-59
1-37. Read Completion-Status Control Block 1-61

© Copyright IBM Corp. 1991

1-38. Command for Completion Status Field Encoding 1-61
1-39. Completion-Status Data Block for Immediate Commands 1-62
1-40. Completion-Status Data Block for Base Control Blocks 1-63
1-41. Read Configuration Control Block 1-64
1-42. Read Configuration Data Block 1-65
1-43. Time-Field Specifier 1-67
1-44. Run-Diagnostic-Test Control Block 1-70
1-45. Write Control Block 1-71
1-46. Immediate-Command Formats 1-73
1-47. Operation-Code Bits—Format 1 1-75
1-48. No-Operation Command Structure 1-76
1-49. Reset Device Command Structure 1-77
1-50. Reset Interrupt Status Port Command Structure 1-79
1-51. Reset Control Block Interrupt Command Structure 1-81
1-52. Reset Subsystem Command Structure 1-84
1-53. Resume Command Structure 1-86
1-54. Run Immediate Diagnostic Test Command Structure . 1-87
1-55. Suspend Command Structure 1-89
1-56. Subsystem Control Port (Hardware-Controlled Subsystem Reset) 1-91
2-1. Locate Mode Physical Level 2-2
3-1. Overall Delivery-Level Structure 3-2
3-2. Command Port—Immediate Commands 3-4
3-3. 8-Bit Attention Port—Immediate Commands 3-5
3-4. Command Port—Control-Block Commands 3-6
3-5. 8-Bit Attention Port—Control-Block Commands 3-6
3-6. Command-Delivery Flow 3-11
4-1. Control Blocks without Command Chaining 4-3
4-2. Control Blocks with Command Chaining 4-4
4-3. Command Chaining with Conditional and Chain No Error Enabled 4-6
4-4. Indirect List 4-11
4-5. Control-Block Command with Indirect List 4-12
5-1. Residual Status Data after Control-Block Execution .. 5-22

Chapter 1. Overview

The Locate mode defines a control structure that allows the coupling of commands, status, and data to represent a request from a client to a server. A client delivers requests, one at a time, in one of two forms: as an immediate command or as a command control block. A client initiates a request by passing either an immediate command or the address of a command control block to a server through a set of I/O ports in I/O address space. The following figure shows an example of the structure of control-block delivery.

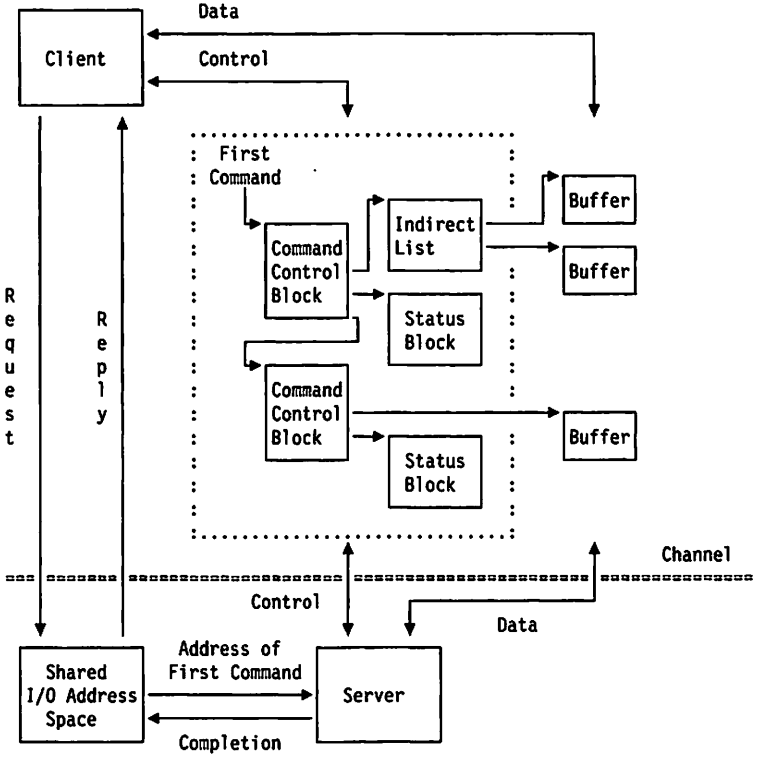


Figure 1-1. Control-Block Delivery Structure Example

A server uses the information passed in the I/O ports to determine whether the request is an immediate command or the address of a control block containing a command.

© Copyright IBM Corp. 1991

If the request is in the form of an immediate command, the server uses the information contained in the set of I/O ports directly and returns a reply to the client through the same I/O ports and an interrupt. If the request is in the form of a command-control-block address, the server uses the information contained in the I/O ports to fetch the command control block. Upon completion of the command, the server puts the completion status into the status block and returns a reply to the client, using the I/O ports and an interrupt.

The following information explains the operational characteristics and the flow of the delivery support. This information is necessary to understand the overall operation of the Locate mode.

Operational Characteristics

The operational characteristics of the Locate mode are as follows:

- Unidirectional delivery.

  The flow of requests is from a client in a system master to a server in a subsystem. The server cannot send requests to a client.

- Multiple devices attached to a subsystem.

  A subsystem can support more than one device attached to it. Requests are directed to a specific entity, which supports an attached device. All entities within a subsystem use a single set of delivery resources in the subsystem to communicate with the system master.

- Shared delivery control areas.

  A subsystem has a single set of control areas in I/O space that are used by the system master to send requests to any of its devices and to receive status from any of its devices. The Command port and the Attention port are used to deliver requests. Sending a request to a device on the subsystem makes these shared delivery-control areas busy until the request is accepted by the subsystem. The busy indication is provided in the Command Busy/Status port. A rejected request causes the subsystem delivery space to be put into the reject state. In the reject state, the system-master delivery support needs to issue a special subsystem command to clear it before additional requests can be sent to any device on the subsystem.

  Delivery support in the system manages the subsystem's control areas as a set of serially-reusable resources used for the delivery of one request at a time.

- Device use.

  A device executing a request does not accept a new request until any current request, other than an immediate command such as the Reset command or the Suspend command, has been completed. Once a request has started, a device indicates busy-reject status if it receives another request (unless it has been issued a Suspend or Reset command) and puts the subsystem's delivery space into a reject state. To avoid this, the

entity in the system master must manage devices on subsystems as serially-reusable resources. Interrupts sent by a completed (or staged) request can be used to pace requests to a device.

- Signalling of request or command completion through interrupts.

  After a request or a particular command is completed by the device or subsystem, the subsystem optionally signals the system master of the completion by presenting an interrupt request to the system master. Interrupt status from the subsystem is stored in the Interrupt Status port, which supports only one interrupt at a time. When an interrupt is presented to the system master, another interrupt cannot be presented until the Interrupt Status port is cleared with a command from the system master. Other devices must enqueue their interrupts within the subsystem until the port is cleared.

- Handling interrupts from an adapter.

  Request-delivery support in the subsystem services requests to all of the devices attached to the subsystem. Interrupts from the devices are handled by a single shared-interrupt-request handler and serialized through a single set of subsystem interrupt resources.

  The system interrupt-handling logic that receives the interrupt from the subsystem interrupt handler must determine which instance of request-delivery support in the system should be notified of the interrupt. This can be done in the system by using a layered approach to interrupt handling consisting of shared first-level support and instances of second-level support that are unique to request delivery for a specific device.

  First-level support determines which subsystem has an interrupt. If an interrupt is present, it clears the interrupt and notifies the instance of the request delivery, which is waiting for an interrupt from this device. It also disables any further interrupts from this device until the device is enabled by the second-level support. The delivery support uses certain immediate commands to disable and enable device-level interrupts.

  For certain versions of Locate mode operation, the interrupt can indicate multiple device-level interrupts through the Device Interrupt Identifier port. For these, the first-level support notifies each instance that has a device-level interrupt.

The instance of the request delivery (the second level of support) typically has its execution blocked and is waiting for an interrupt from a device to which it has issued a request. When the second-level support has been notified, the first-level support is free to service another interrupt from a subsystem.

Second-level support in a request-delivery instance handles request-completion interrupts from the specific device attached to a subsystem. It determines which command control blocks have been completed and returns this information to the send interface.

Command and Control-Block Delivery

The following figure shows the flow of requests from the system master to a subsystem.

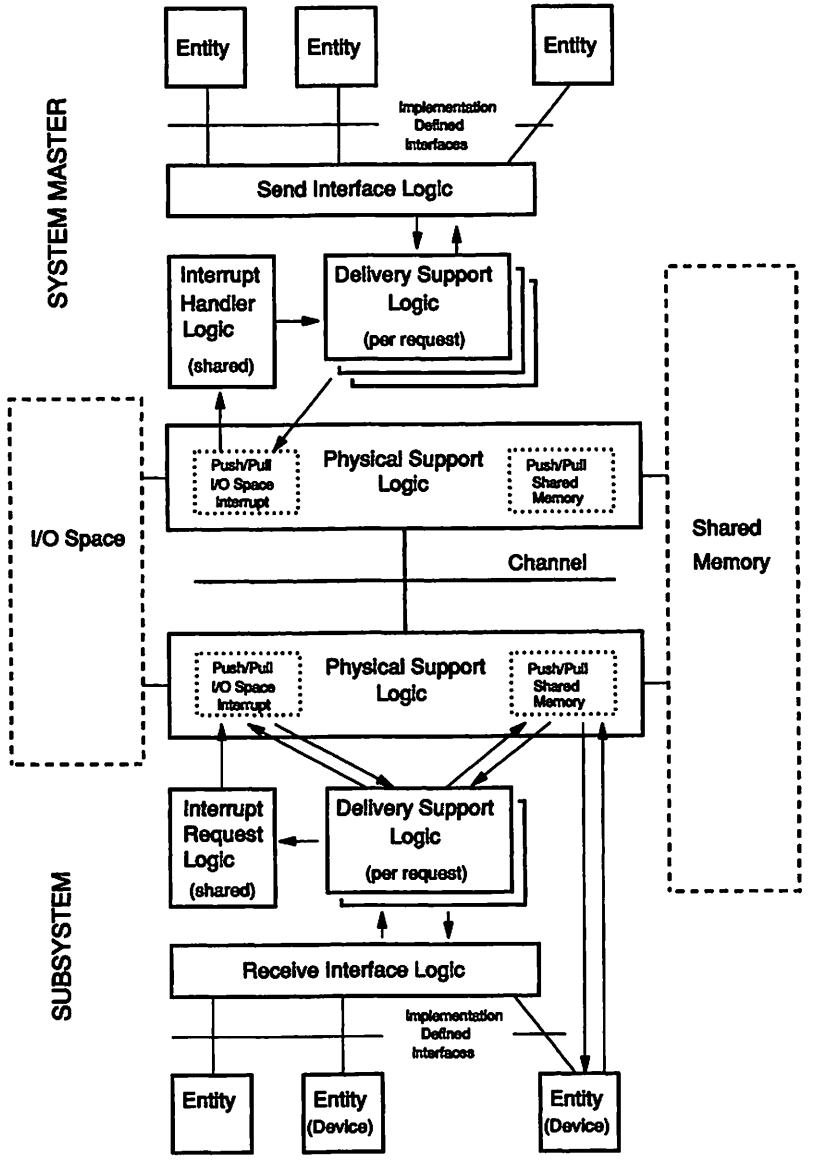


Figure 1-2. Request-Delivery Flow

The following information describes the flow through the various parts of a client in the system master sending a request to a server in a subsystem.

1. The send interface handles the translation of the local form of send, and its parameters, from the client into the request control block, as seen in shared memory.
2. The request delivery has an instance for each entity sending a request to an entity in the subsystem. The following functions are provided by the request-delivery support:
   - Management of access to the control areas in I/O address space shared among all request-delivery flows to the same subsystem
   - Coordination of the request flow with interrupt handling for a particular subsystem or device.
3. After the request delivery obtains access to control areas, it performs the read and write operations to the control areas to pass control parameters to the request delivery in the subsystem.
4. When all parameters have been written to the control areas, the request delivery writes a specific value to the Attention port, causing an interrupt of the subsystem.
5. When this interrupt occurs in the subsystem, the request-delivery support in the subsystem passes the request to the processing-level support, through the receive interface.
6. When the processing level completes the request, it can request that the subsystem schedule an interrupt to the system master.
7. When the system-master interrupt occurs, the interrupt handler in the system master determines which subsystem and device are interrupting and notifies the corresponding request-delivery instance.
8. The request-delivery support determines the completion status of the request and returns that information to the send interface.
9. After being notified that the request was completed, the client can issue the next request or continue to process the notification for other control blocks in a chain.

Because multiple subsystems of the same type or of different types can be used in the system, the base address for the I/O space of each subsystem must be defined during configuration. The I/O ports used by a Locate mode type of subsystem are defined in "I/O Address Space" on page 1-15.

The following are general descriptions of the control areas used by Locate mode delivery. A port is a byte or set of contiguous bytes in I/O space.

- Subsystem Control port

  The Subsystem Control port is used to directly control the subsystem-to-system interface.

- Request ports

  There are five ports that are used in the delivery of requests to a device and the delivery of replies (interrupts) from a device.

  - The Command port is used to pass either the address of a control block or an immediate command.
  - The Attention port contains an attention code and a device identifier. The attention code informs the subsystem that the Command port contains either the address of a control block or an immediate command. The device identifier indicates which device on the subsystem the request is directed to.
  - The Interrupt Status port is used to identify the interrupting device and to indicate any exception that occurred during command processing.
  - The Command Busy/Status port is used in the subsystem to serialize access to the shared logic of the control-block delivery service. This port contains the following indicators:
    - Busy—indicates that the subsystem is busy (using the shared logic) and will therefore ignore submitted commands.
    - Interrupt Valid—indicates that the contents of the Interrupt Status port are valid and that the subsystem has an interrupt pending.
    - Reject—indicates that the subsystem has rejected a request (a Reset Reject command is needed to clear a reject and allow the subsystem to resume accepting requests).
    - Status—indicates the reason for the reject.

- The Device Interrupt Identifier port is an optional method of reporting control-block interrupts. When this port is used, only the interrupt status for immediate commands is reported through the Interrupt Status port. All other interrupts are reported through the Device Interrupt Identifier port.

  This optional port allows the interrupt handler to process multiple control-block interrupts through a single system interrupt. For more information on interrupt handling, refer to "Signalling Protocol" on page 3-13.

Control-Block Structure

The following figure shows the structure and content of a typical command control block. There are two formats for command control blocks: base and extended. Both formats contain all the fields shown in solid lines. The remaining fields (shown in dashed lines) are present only in the extended control-block structure. The device-dependent area is in both formats; however, the actual location of the device-dependent area within each control block depends on whether the base or the extended control block is used.

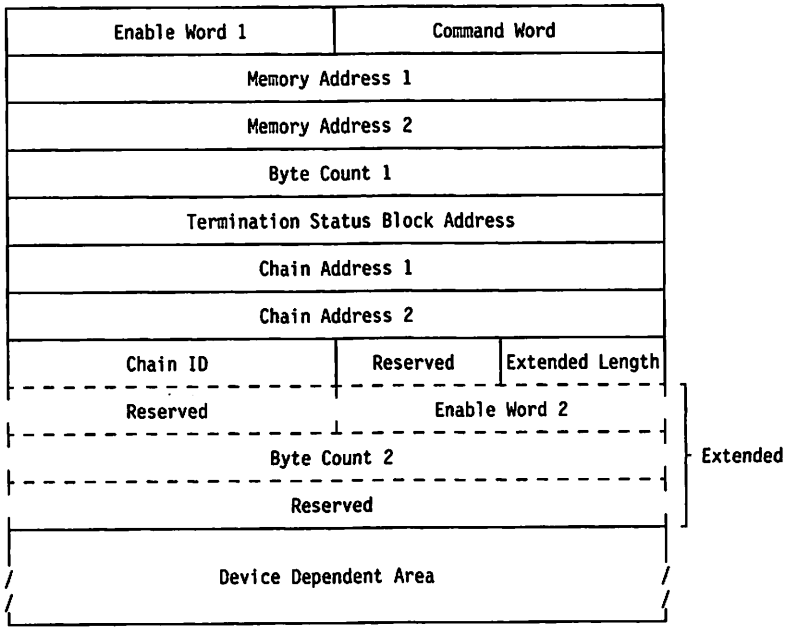


Figure 1-3. Control Block Format

Termination Status Block Structure

In addition to the completion indication in the Interrupt Status port, status information can be reported for each command control block, using the termination status block. A termination status block is always stored if an exception occurs during control-block processing; optionally, it can be stored for each control block processed. A field in the command control block points to the area in shared memory where the termination status block is stored. The format of the termination status block is shown in the following figure.

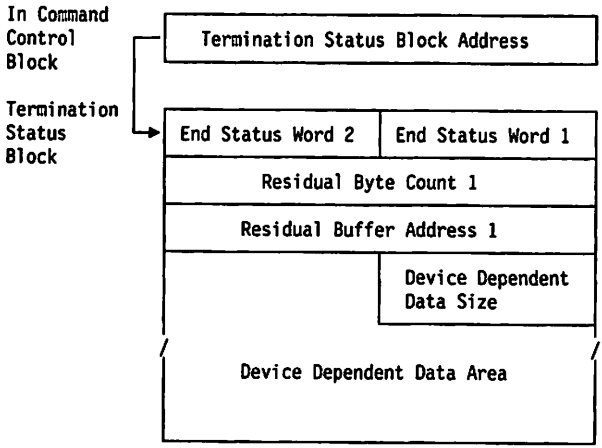


Figure 1-4. Termination Status Block Format

Indirect-List Structure

An indirect list is a variable-length list consisting of the address-count pairs used to support data chaining. Both the location of the indirect list and its length are specified in the control block. Within the indirect list, the Address field and the Count field are each doubleword fields. The format and content of an indirect list is shown in the following figure.

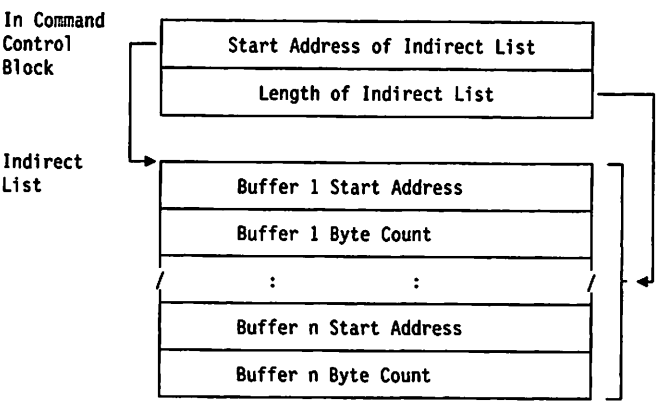


Figure 1-5. Indirect-List Format

Delivery Structure

The following figure shows how the various levels relate to each other and to the control areas in I/O address space and control blocks in shared memory. It also shows the send and receive interfaces between the delivery level and the entity level. There is no unit-to-unit protocol associated with this support.

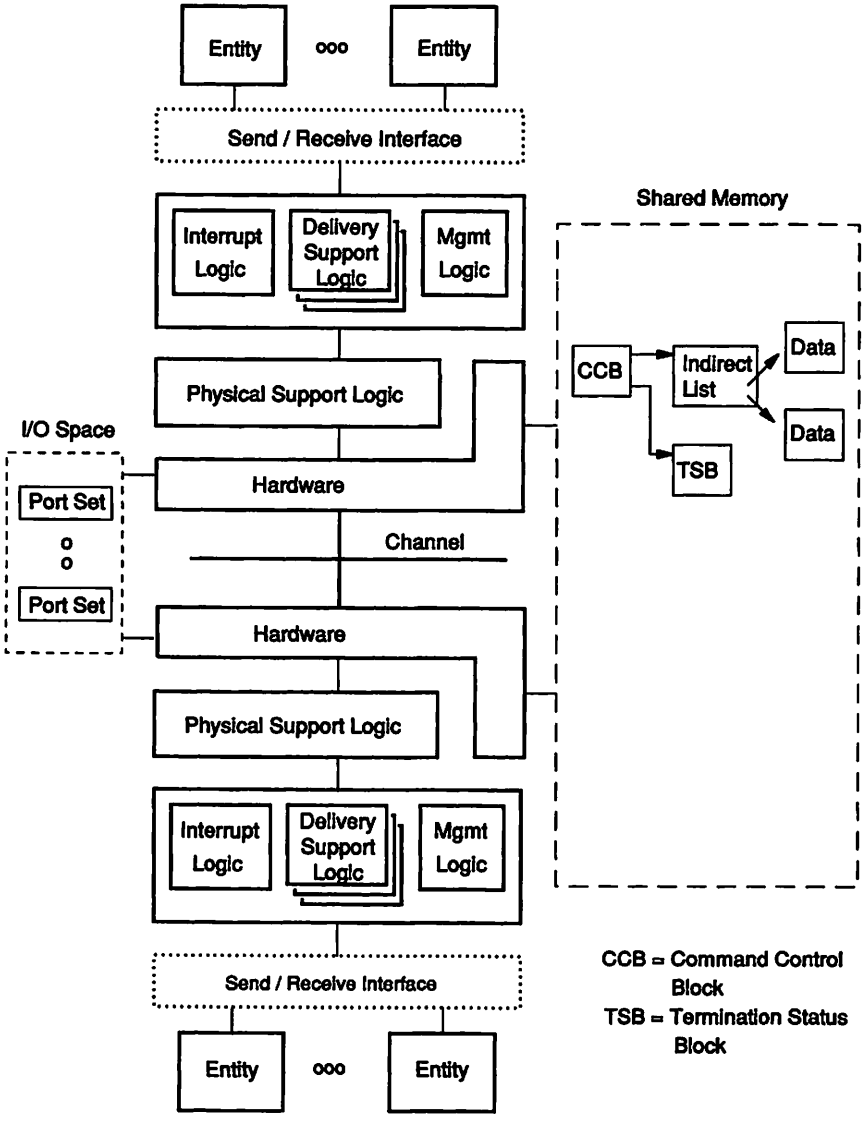


Figure 1-6. Locate Mode Delivery Structure

I/O Address Space

The SCB architecture defines support for two different physical interfaces: Type 1 and Type 2. The registers used in these interfaces perform the same functions; the difference between the interfaces is the location and size of required ports.

The interface used by a subsystem is determined by the number of physical devices that can be attached to it. When 16 or more devices are supported, the Type 2 interface is used.

Physical Interface Description

Both Type 1 and Type 2 physical interfaces use the following control areas in I/O address space:

- Command port
- Attention port
- Subsystem Control port
- Interrupt Status port
- Command Busy/Status port
- Device Interrupt Identifier port.

Because multiple subsystems of the same type can be present on the channel, the addresses within the I/O address space are shown as offsets from the base I/O address. The base I/O address used by each subsystem is determined during system configuration.

The following figure shows the Type 1 interface.

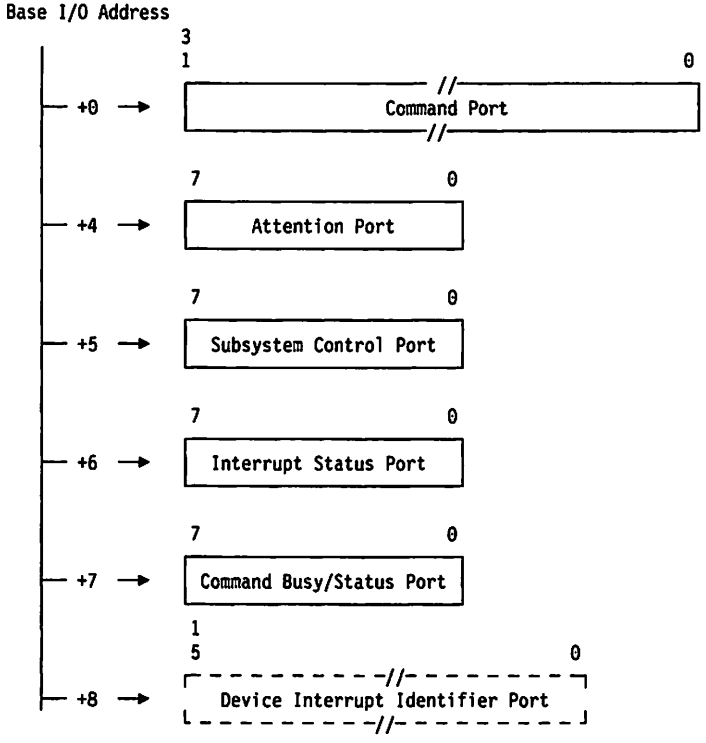


Figure 1-7. I/O Address Map – Type 1

The following figure shows the Type 2 interface.

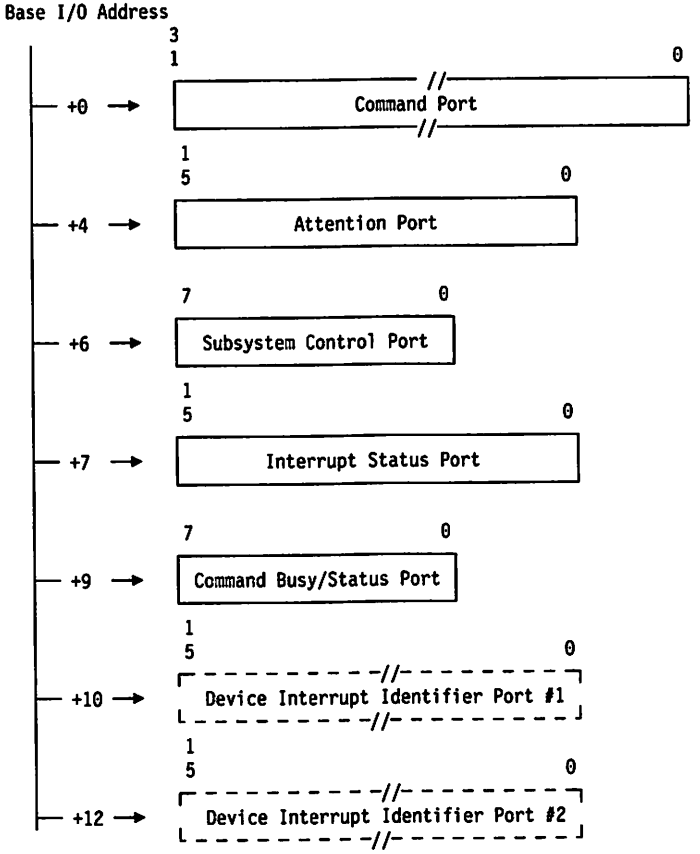


Figure 1-8. I/O Address Map—Type 2

Port Descriptions

The following information describes the ports for the Type 1 and Type 2 interfaces.

Command Port

This 32-bit port is used to deliver a 32-bit immediate command or the physical address of a control block to a subsystem. The immediate command, or the control block pointed to by the address in the Command port, specifies an operation to be performed by the subsystem or the device.

The format and content of the Command port are shown in the following figure.

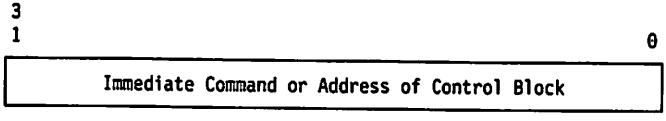


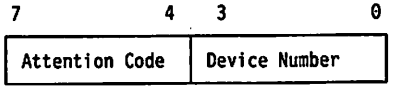
Figure 1-9. Command Port

Attention Port

This port signals the subsystem or device associated with the subsystem. The signal can interrupt and notify the subsystem that the Command port contains either an immediate command or the address of a control block. This port is also used to reset an interrupt and can be used to issue a software reset.

The size of the Attention port is determined by the number of devices supported; however, the two fields that make up this port serve the same function, regardless of the number of devices. The Attention port is an 8-bit port if there are fewer than 16 devices supported by the subsystem; it is a 16-bit port if 16 or more devices are supported.

Fewer than 16 Devices



16 Devices or More

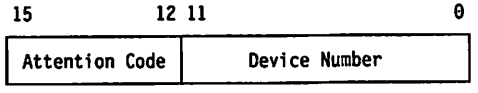


Figure 1-10. Attention Port

Attention Code Field: The attention codes identify the specific operations for a device. These codes typically specify an operation that is a single-purpose signal to the device. For example, attention code hex 1 specifies that the Command port is loaded with an immediate command, and attention code hex 3 specifies that the port contains the address of a control block. The following is a summary of the valid attention codes, their names, and a brief description of each.

Code	Name	Description
0	Reset command	Requests the subsystem to perform a device reset for the specified device.
1	Immediate command	Requests the subsystem to execute the command contained in the Command port.
2	Reserved	
3	Start Control Block command	Requests the subsystem to process the control block pointed to by the address in the Command port.
4	Device dependent	See note.
5-C	Reserved	
D	Move mode delivery	Used to signal a request for Move mode command delivery.
E	End of interrupt	Requests the subsystem to perform one of the two interrupt-resetting commands. This is a hybrid command that requires both the subsystem hardware and software to clear the interrupts presented to the system unit.
F	Device dependent	See note.

Note: The device-dependent codes should not duplicate a function that is already defined by the architecture. Although the command or control block associated with these attention codes is device dependent, implementers must not redefine the basic architecture.

Figure 1-11. Attention Codes

Device Number Field: This field specifies which device receives the attention code. The Type 1 interface can select one of 16 devices from 0 to 15; the Type 2 interface can select one of 4096 devices from 0 to 4095. Device 0 is reserved for the subsystem and is used to direct commands or control blocks to the subsystem, rather than to any attached device, allowing the subsystem to act as the focal point for all devices.

Subsystem Control Port

This 8-bit port provides direct control of the subsystem hardware. The following is a description of the Subsystem Control port.

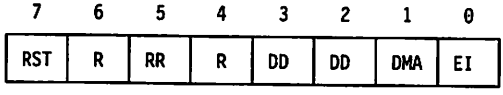


Figure 1-12. Subsystem Control Port

Bit 7 The subsystem-reset (RST) bit provides a hardware-controlled reset of the subsystem and all attached devices. The bit does not affect programmable-option-select data. See "Reset Subsystem (Hardware Controlled)" on page 1-90 for more details on the use of the subsystem-reset bit.

Bit 6 This bit is reserved.

Bit 5 The reset-reject (RR) bit restores the subsystem to a state in which it can receive commands. When this bit is set to 1, the reject state is cleared, and the busy bit and the reject bit in the Command Busy/Status port are reset to 0. If the subsystem is not in the reject state, setting this bit to 1 has no effect. The reject state is always cleared after a reset of the subsystem or after a power-on reset. (See "Command Busy/Status Port" on page 1-24 and "Delivery-Level Protocols" on page 3-10 for more details on the reject state.)

Bit 4 This bit is reserved.

Bits 3, 2 These device-dependent bits can be used by subsystems to meet their specific needs. If the bits are used, they must be defined in the technical documentation for the specific subsystem.

Bit 1 The enable-DMA (DMA) bit controls enabling and disabling of the DMA operations. When this bit is set to 1, DMA operations are enabled. When this bit is set to 0, DMA operations are disabled. DMA operations are always disabled after a reset of the subsystem or after a power-on reset.

Bit 0 The enable-subsystem-interrupts (EI) bit controls physical interrupts to the system master that are generated by the subsystem. When this bit is set to 1, the interrupts are enabled. When this bit is set to 0, the interrupts are disabled. Interrupts are always disabled after a reset of the subsystem or after a power-on reset. (See "Signalling Protocol" on page 3-13 for a description of interrupts.)

Interrupt Status Port

This port is used by a subsystem to present interrupt information for the subsystem or device to the system master. It can be an 8-bit port or a 16-bit port, depending on the number of devices supported by the subsystem.

Because this port can contain interrupt information for only one command at a time, an interrupt for another command cannot be presented to the system master through the Interrupt Status port until the previous interrupt has been reset by the Reset Interrupt Status Port command.

When a subsystem does not support the Device Interrupt Identifier ports, all interrupt information is presented through this port. When the Device Interrupt Identifier port is supported, only the interrupt information for immediate commands or for commands with hardware failures are presented through this port.

When the Device Interrupt Identifier port is supported, interrupt information for control block commands is presented in the Device Interrupt Identifier ports and in the termination status block associated with each control block. See "Device Interrupt Identifier Port" on page 1-28 for more information.

The format and content of the Interrupt Status port are shown in the following figure, followed by a description of the individual fields.

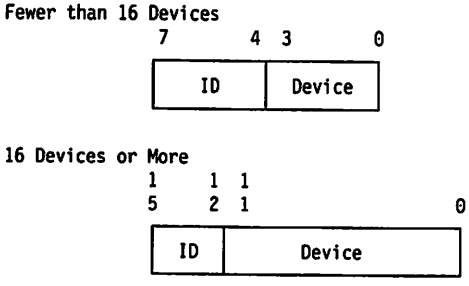


Figure 1-13. Interrupt Status Port

Interrupt Identifier Field: The Interrupt Identifier (ID) field is encoded with a value that identifies the cause of the interrupt. If the interrupt is the result of a control-block command, more detailed information on the cause is available in the termination status block for that control block.

The following figure contains a summary of all the valid interrupt-identifier codes.

Hex	Interrupt Definition
0	Reset completed without error
1	Control-block command completed without error
2–4	Reserved
5	Device dependent
6	Inform interrupt
7	Hardware failure – immediate command
8	Hardware failure – control-block command
9	Reserved
A	Immediate command completed without error
B	Reserved
C	Control-block command completed with error
D	Immediate command completed with error
E	Reserved
F	Device dependent

Figure 1-14. Interrupt-Identifier Codes

The following is a description of all the valid interrupt-identifier codes.

Code Definition

0 The subsystem or device has completed a reset operation without error.

1 The device or subsystem has completed the control-block operation without error. If the specific control block enables storing of the termination status block, additional status is stored in the termination status block.

This code is not used when the Device Interrupt Identifier port is supported.

2–4 These codes are reserved.

5 This code is device dependent.

6 This code can be used to indicate a condition that is not associated with any specific command.

7 A hardware failure occurred that prohibits the completion of the immediate command or hardware-controlled command.

8 While a control-block command was being processed, a hardware failure caused one of the following:

- The subsystem or device attempted to read the control block, and the read operation failed.
- The subsystem attempted to store a termination status block, and the store operation failed.

If the condition occurs while the device is processing a chained control block, the chain is terminated. If the hardware failure does not prohibit the device from responding to commands, the device can store command-completion status internally. The contents of the termination status block are not valid; however, additional status can be obtained with a Read Completion Status command, depending upon the specific hardware failure.

9 This code is reserved.

A The subsystem has completed the immediate or hardware-controlled command without an error.

B This code is reserved.

C A specific control block has been completed, and an error occurred. The subsystem or device has stored the status. See "Termination Status Blocks" on page 1-43 for details on the termination status block and "Signalling Protocol" on page 3-13 for details on interrupt status and completion.

This interrupt-identifier code is not used when the Device Interrupt Identifier port is supported.

D An immediate or hardware-controlled command was completed with a nonhardware failure, or the device failed the diagnostic test. (See "Run Immediate Diagnostic Test" on page 1-87.)

If the device failed the diagnostic test, it can provide additional information about the reason for the failure through a Read Completion Status command or a Run Diagnostic Control Block command.

The subsystem or device might need to be reset before any diagnostic data can be obtained by a Run Immediate Diagnostic Test command.

E This code is reserved.

F This code is device dependent.

Detailed explanations of interrupt handling are provided in "Signalling Protocol" on page 3-13.

Device Number Field: The Device Number (Device) field identifies the device that presented the interrupt status.

Command Busy/Status Port

This 8-bit port has two functions:

- After an immediate command or command-control-block address is submitted, the Command Busy/Status port is read to determine the status of the command or control block. The system must allow the subsystem a predefined period of time before attempting to read this port.
- It also indicates whether the subsystem has a valid interrupt value present in the Interrupt Status port. The Interrupt Status port should not be read unless the Command Busy/Status port indicates that the interrupt is valid.

The Command Busy/Status port is shown in the following figure and is followed by a description of the individual bits.

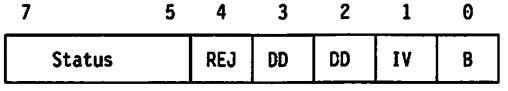


Figure 1-15. Command Busy/Status Port

Bits 7 – 5 The status bits indicate the reason that the command or control block has been rejected. If the reject bit is 0, the status bits are undefined and should be ignored. (See "Delivery-Level Protocols" on page 3-10 for more details on command and control-block submission.)

The encoded values and meanings of the status bits are shown in the following table.

Code	Meaning
0 0 0	**Reserved**
0 0 1	**Device Not Available** The subsystem determined that a device cannot perform the command or process the control block. The condition is cleared by the Reset Device or Reset Subsystem command.
0 1 0	**Invalid Command** The subsystem does not recognize an attention code in the Attention port, or an immediate command is not a valid command.
0 1 1	**Device Busy** A device is busy executing a command or processing a control block and cannot accept a new command or control block.
1 0 0	**Device Control-Block Execution Suspended** The device has been put into a suspended state. It can be reset by the Resume, Reset Device, or Reset Subsystem command.
1 0 1	**Invalid Device Number** The Device Number field of the Attention port contains an invalid value.
1 1 0	**Reserved**
1 1 1	**Device Limits Reached** The device's internal capacity to accept another command has reached its limit. The condition will be cleared as the device completes the execution of commands and the system accepts and resets interrupts from the device. If the condition persists, it can also be cleared by a Reset Device or a Reset Subsystem command. (See "Signalling Protocol" on page 3-13 for details.)

Figure 1-16. Command Busy/Status Port – Status Bit Encoding

Bit 4 The reject (REJ) bit indicates that the subsystem has rejected a command or control block submitted through the Command ports and Attention ports. The reason is encoded in the status bits in the Command Busy/Status port. See Figure 1-17 on page 1-27 for details about this bit.

Bits 3, 2 These bits are device dependent.

Bit 1 The interrupt-valid (IV) bit is set to 1 whenever the subsystem writes an interrupt value into the Interrupt Status port. When this bit is set to 1, the Interrupt Status port contains a valid interrupt identifier and device number. When this bit is set to 0, the Interrupt Status port does not contain a valid interrupt value. (For details on interrupts, see "Signalling Protocol" on page 3-13.)

Bit 0 The busy (B) bit indicates whether the Command port and Attention port are currently being used or whether a hardware-controlled reset is in progress. The following table shows encoding of the busy bit and the reject bit.

Busy Bit	Reject Bit	Description
0	0	Writing to the Command port and Attention port is not blocked.
0	1	Reserved.
1	0	Writing to the Command port and Attention port is blocked. The maximum time for this condition is specified in the configuration data.
1	1	Writing to the Command port and Attention port is blocked because the previous command has been rejected. This condition is cleared by setting the reset-reject bit in the Subsystem Control port to 1.

Figure 1-17. Busy-Bit And Reject-Bit Settings

Device Interrupt Identifier Port

This optional port indicates which devices have control-block logical interrupts pending. When a bit is set to 1, it indicates that the device associated with the bit has at least one control-block logical interrupt waiting to be serviced. The interrupt status associated with the control block has been stored in the termination status block. When a bit is set to 0, it indicates that no control-block logical interrupts have been posted for that device. Bits can be reset with the Reset Control Block Interrupt command.

The following figure shows the format and content of the Device Interrupt Identifier port.

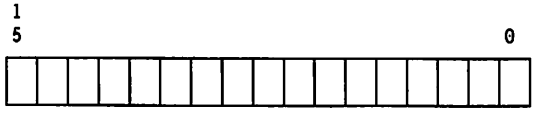


Figure 1-18. Device Interrupt Identifier Port

Each bit, starting with bit 0, has a one-to-one correspondence with a physical device. For example, bit 0 represents the subsystem (device 0); bit 1 represents device 1. When the subsystem supports more than 15 devices, the devices are assigned to additional Device Interrupt Identifier ports consecutively, starting at bit-position 0. For example, a subsystem that has 17 attached devices requires two Device Interrupt Identifier ports; bits 0 and 1 in the second Device Interrupt Identifier port are assigned to devices 16 and 17, respectively.

The Read Configuration command can be used to determine the number of Device Interrupt Identifier ports supported by a subsystem. (See "Read Configuration" on page 1-63.) For more information about the use of the Device Interrupt Identifier ports, see "DIIP Interrupts (Multiple Form)" on page 3-26.

Memory Address Space

In the Locate mode, shared memory is used to hold command control blocks, indirect lists, termination status blocks, and data. Each is described in detail later. The organization of shared memory is described in the System Resource description of the in Part 1, SCB Architecture Overview.

Control Structures

The Locate mode uses the following control structures: the command control block, the termination status block, and the indirect list.

Command Control Blocks

In the Locate mode, command control blocks are used to exchange control and status information between a client and a server. Command control blocks are variable-length areas in the system's memory address space containing the commands and arguments passed to a subsystem. These arguments describe the operation to be performed.

There are two formats for command control blocks: base and extended. The base command control block can use only one memory operand at a time and can increment only the memory address associated with the operation.

The extended command control block can use two memory operands in a single operation and can increment or decrement each memory address independently. The additional fields in the extended command control block used to support the additional functions are Enable Word 2 and Byte Count 2.

Some important characteristics of the command control block are:

- Command control blocks can be chained to other command control blocks.
- After a command control block is sent to a subsystem, it is read-only to a system master and to a device or subsystem. Command control blocks can be modified by the system master before being sent or after the device or subsystem has completed the task.
- Command control blocks must be aligned on a doubleword memory boundary.
- Extended termination status blocks must be used with extended command control blocks, and base termination status blocks must be used with base command control blocks.
- All memory addresses contained within a command control block must be specified as physical addresses.
- A device or subsystem in a feature adapter is required to read only those command-control-block fields that it needs.

The following figure shows the format of the base and extended command control blocks.

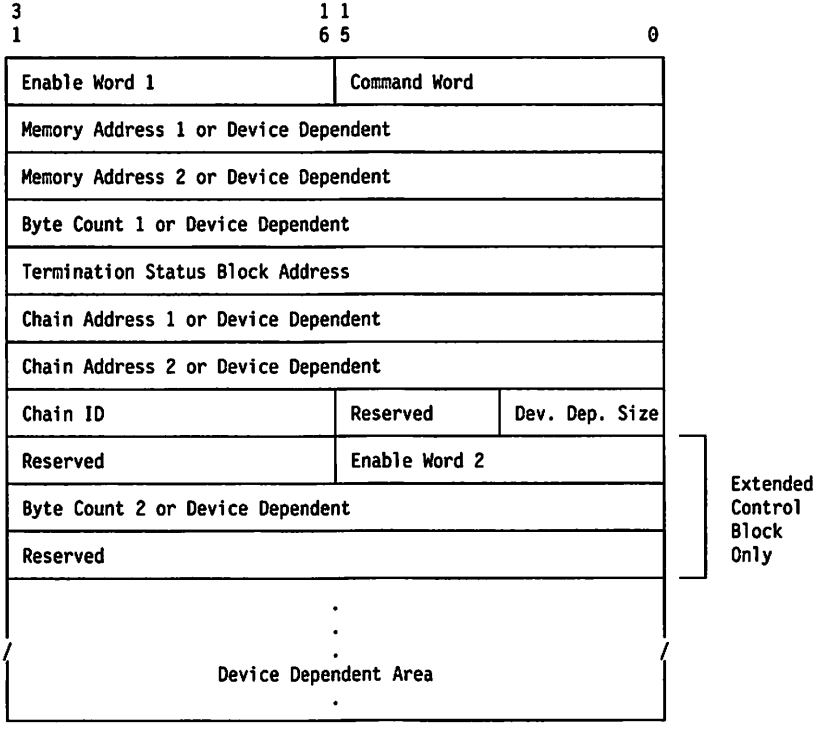


Figure 1-19. Command Control Block Structure

Command Word

The Command Word identifies the operation to be performed by a subsystem or device. The following figure shows the format of the Command Word and a description of each field.

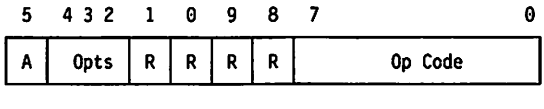


Figure 1-20. Command Word Format

Bit 15 The architected (A) bit indicates whether the command specified in the Op Code field is an architecture-defined command or one that is specific to the device. When this bit is set to 1, the Op Code field contains an operation code defined by the architecture. When this bit is set to 0, the operation code is device dependent. (See "Control-Block Commands" on page 1-55 for definitions of architected commands.)

Bits 14–12 The Options (Opts) field is used to modify or qualify the operation performed for each of the architected commands. The meaning of each bit depends on the command (see the specific command for the meaning and use of these bits).

Bits 11–8 These bits are reserved.

Bits 7–0 The Op Code field is used to identify the specific command to be performed. The value used can be for commands defined by the architecture, or it can be for a command that is defined by and specific to that subsystem or device. Bit 15, the architected bit, indicates whether the op-code value is architecture defined or device dependent.

For commands defined by the architecture, the op-code values are as follows.

Value	Meaning
0000 0000	No Operation
0000 0001	Read
0000 0010	Write
0000 0101	Run Diagnostic Test
0000 0110	Initialize Device
0000 0111	Read Completion Status
0000 1010	Read Configuration
All architected values not shown are reserved.	

Figure 1-21. Architected Op Codes

Enable Word 1

Enable Word 1 defines parameters to be used with this command control block. The following is the format of Enable Word 1 and a description of each field. DD indicates that the bit position is device dependent and that the bit can be used by individual subsystems and devices for specific needs.

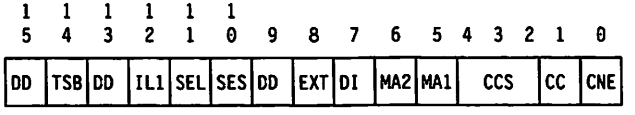


Figure 1-22. Enable Word 1 Format

Bit 15 This bit (DD) can be used for applications that are device dependent.

Bit 14 The conditional termination-status-block (TSB) bit specifies whether status is stored for this command control block. When this bit is set to 0, a termination status block is stored for this command control block. When this bit is set to 1, the termination status block is stored only when an error or exception occurs while the command control block is being executed. (Whenever an error or exception occurs, a termination status block is stored, regardless of whether the condition is suppressed.)

Note: Independent of the setting of this bit, End Status Word 1 in the termination status block is stored whenever the Device Interrupt Identifier port is supported and an interrupt is generated at the completion of the command control block (that is, the disable-interrupt bit is set to 0).

Bit 13 This bit and bits 15 and 9 can be used for device-specific parameters.

Bit 12 The indirect-list-1 (IL1) bit specifies whether an indirect list (used to chain data) is being used with this command control block. If this bit is set to 1, an indirect list is being used (an indirect list must begin on a doubleword boundary). If this bit is set to 0, an indirect

list is not being used. Refer to "Data Chaining" on page 4-10 for additional information on using indirect lists.

Figure 1-23 on page 1-36 describes the use of, and restrictions on the use of, the indirect-list-1 bit.

Bit 11 The suppress-exception-long (SEL) bit specifies whether a long-length exception is to be treated as a major error. (Refer to "Long-Length Exception" on page 5-11 for more information on long-length exceptions.)

If this bit is set to 1, a long-length exception condition is not treated as a major error. If this bit is set to 0, the exception is treated as a major error.

When a long-length exception occurs, the long-length-exception bit in End Status Word 1 is set to 1. The residual byte count and residual buffer address are stored in the termination status block. A single residual byte count and residual buffer address are stored in the appropriate field if only one of the Memory Address fields is specified in the command control block. When both Memory Address fields are used, two sets of residual byte counts and residual buffer addresses are stored.

Bit 10 The suppress-exception-short (SES) bit specifies whether a short-length exception is treated as a major error. Refer to "Short-Length Exception" on page 5-9 for more details on short-length exceptions.

If the suppress-exception-short bit is set to 1, a short-length-exception condition is suppressed and is not treated as a major error. If the suppress-exception-short bit is set to 0, the exception condition is treated as a major error.

When a short-length exception occurs, the short-length-exception bit in End Status Word 1 is set to 1. The residual byte count and residual buffer address are stored in the termination status block. A single residual byte count and residual buffer address are stored in the appropriate field if only one of the Memory Address fields is specified in the command control block. When both Memory Address fields are used, two

sets of residual byte counts and residual buffer addresses are stored.

Bit 9 This bit is device dependent.

Bit 8 The extended-structure (EXT) bit indicates whether this command control block is a base or extended command control block. When this bit is set to 1, the command control block is an extended command control block. When this bit is set to 0, the command control block is a base command control block.

Bit 7 The disable-interrupt (DI) bit specifies whether the subsystem generates an interrupt after the normal processing of this command control block is completed or only when an error or exception occurs.

When this bit is set to 1, no interrupt is generated upon completion of the command control block. When this bit is set to 0, an interrupt is generated upon completion of the command control block.

If an unsuppressible error or exception occurs during control-block processing, an interrupt is generated, regardless of the setting of this bit.

Bit 6 The memory-address-2-selected (MA2) bit indicates whether the Memory Address 2 field contains a physical address to be used in the operation specified. If this bit is set to 1, the Memory Address 2 field contains an address, and the field operates as defined by the architecture. If this bit is set to 0, the Memory Address 2 field can be used as a device-dependent field.

For the base command control block, the Memory Address 1 field and the Memory Address 2 field are mutually exclusive because only one memory operand can be specified. The extended command control block, however, can use operations that specify two memory operands.

Figure 1-23 on page 1-36 describes the use of, and restrictions on the use of, the memory-address-2-selected bit.

Bit 5 The memory-address-1-selected (MA1) bit indicates whether the Memory Address 1 field contains a physical address to be used in the operation specified. If the bit is set to 1, the Memory Address 1 field contains an address, and the field operates as defined by the architecture. If the bit is set to 0, the Memory Address 1 field can be used as a device-dependent field.

For the base command control block, the Memory Address 1 field and the Memory Address 2 field are mutually exclusive because only one memory operand can be specified. The extended command control block can use operations that specify two memory operands.

The following figure describes the use of, and restrictions on the use of, the memory-address-1-selected bit, the memory-address-2-selected bit, the extended-structure bit, and the indirect-list-1 bit. For a description of how to use indirect lists, see "Data Chaining" on page 4-10.

IL1	EXT	MA2	MA1	Description
X	X	0	0	Neither Memory Address nor Byte Count is selected. Both fields are device dependent.
0	0	0	1	Memory Address 1 and Byte Count 1 are selected. The address can only be incremented.
0	0	1	0	Memory Address 2 and Byte Count 1 are selected. The address can only be incremented.
X	0	1	1	Invalid combination. Two memory operands require extended format.
0	1	0	1	Memory Address 1 and Byte Count 1 are selected. The address can be incremented or decremented.
X	1	1	0	Memory Address 2 and Byte Count 2 are selected. The address can be incremented or decremented.
0	1	1	1	Memory Address 1, Byte Count 1, Memory Address 2, and Byte Count 2 are selected. Any of these addresses can be incremented or decremented.
1	0	0	1	Memory Address 1 and Byte Count 1 are selected. The address points to an indirect list; the buffer addresses are incremented.
1	0	1	0	Memory Address 2 and Byte Count 1 are selected. The address points to an indirect list; the buffer addresses are incremented.
1	1	0	1	Memory Address 1 and Byte Count 1 are selected. The address points to an indirect list; the buffer addresses are incremented or decremented.
1	1	1	1	Memory Address 1 and Byte Count 1 are selected. The address points to an indirect list; the buffer addresses are incremented or decremented. Memory Address 2 and Byte Count 2 are selected. The address can be incremented or decremented.

Notes:

1. X means "don't care."
2. If two memory operands are being used, the operation code determines whether one memory operand is the source of the data and one is the target, or whether both operands act as sources or targets.

Figure 1-23. Memory-Addressing-Control Encoding

Bits 4–2 The chain-condition-specification (CCS) bits specify the conditions that must exist for chaining to continue. The conditions under which chaining will continue are listed in the following figure.

Value	Condition
000	An error or exception occurred that was not suppressed, other than specification errors, hardware failures while reading a command control block, or hardware failures while storing a termination status block.
001	The command control block is completed, and the associated termination status block indicates that additional device-dependent data is available.
010-111	These condition codes are reserved.

Figure 1-24. Chain-Condition-Specification Encoding

When the conditional-chaining bit is set to 0, these bits are ignored. See Figure 1-25 for the setting of conditional-chaining and other related bits.

Bit 1 The conditional-chaining (CC) bit specifies whether conditional chaining is enabled or disabled. If this bit is set to 1, conditional chaining is enabled. If this bit is set to 0, conditional chaining is disabled.

See Figure 1-25 for the setting of this and other related bits.

Bit 0 The chain-no-error (CNE) bit indicates whether unconditional chaining is enabled. If this bit is set to 1, unconditional chaining is enabled. If this bit is set to 0, unconditional chaining is disabled.

The following figure shows the relationship among the chaining bits: chain-no-error, conditional-chaining, and chain-condition-specifications.

CNE	CC	Met Chain Conditions	Subsystem Chain Action
0	0	X	No chaining active.
0	1	No	Halt chaining.
X	1	Yes	Chain to Chain Address 2.
1	0	X	If no error or error is suppressed, chain to Chain Address 1; otherwise, halt chaining.
1	1	No	If no error or error is suppressed, chain to Chain Address 1; otherwise, halt chaining.
Note: X means "don't care."			

Figure 1-25. Chaining-Bit Settings

Memory Address 1

The Memory Address 1 field is a 32-bit field. If the memory-address-1-selected bit is set to 1, this field contains the physical address of the source or target memory address to be used in this operation. If the memory-address-1-selected bit is set to 0, the field can be used for device-dependent parameters.

When the Memory Address 1 field is selected and an indirect list is being used, this field points to an area in memory that contains a list of addresses and byte counts used as the source or destination of the data transfer. (See "Data Chaining" on page 4-10 for additional information on indirect lists.)

When the Memory Address 1 field is selected and an indirect list is not being used, this field contains the physical address of the buffer in memory that is used as the source or destination of the data transfer.

Memory Address 2

The Memory Address 2 field is a 32-bit field. If the memory-address-2-selected bit is set to 1, this field contains the physical address of the source or target memory address to be used in this operation. If the memory-address-2-selected bit is set to 0, this field can be used for device-dependent parameters.

When the Memory Address 2 field is selected and an indirect list is being used, this field points to an area in memory that contains a list of addresses and byte counts used as the source or destination of the data transfer. (See "Data Chaining" on page 4-10 for additional information on indirect lists.)

When the Memory Address 2 field is selected and an indirect list is not being used, this field contains the physical address of the buffer in memory that is used as the source or destination of the data transfer.

Byte Count 1

The Byte Count 1 field is a 32-bit field that contains the number of bytes to be transferred. The value contained in this field depends on whether an indirect list is used. (See Figure 1-23 on page 1-36 for bit settings that control the use of this field.)

If an indirect list is not being used, this field contains the number of bytes to be transferred. If the count specified is 0, no data is transferred.

If an indirect list is being used, this field contains the number of bytes in the indirect list, not the number of bytes to be transferred. The byte count must be a multiple of 8 because each indirect-list entry consists of 8-byte elements (a 32-bit physical address and a 32-bit byte count). The sum of all byte counts in the indirect list is the number of bytes to be transferred. If the sum of all byte counts is 0, no data is transferred.

This field can be used for device-dependent parameters if neither Memory Address field is used or if, in the extended format, the Memory Address 1 field is not used.

Termination Status Block Address

The Termination Status Block Address field is a 32-bit field. It contains the physical address of a doubleword-aligned memory area that is used by the subsystem to store the termination status block for this command control block. Refer to "Termination Status Blocks" on page 1-43 for a description of the termination status block.

Chain Address 1

The Chain Address 1 field is a 32-bit field that contains the physical address of the next command control block to be processed by a device or subsystem. All command control blocks are aligned on doubleword addresses.

If the chain-no-error bit is set to 0, the Chain Address 1 field can be used for device-dependent parameters. (See Figure 1-25 on page 1-38 for bit settings that control the use of this field.)

Chain Address 2

The Chain Address 2 field is a 32-bit field that contains the physical address of the next command control block to be processed by a subsystem or device when conditional chaining is enabled. The Chain Address 2 field is used when conditional chaining is enabled and the command control block is completed in a manner that matches the condition set in the chain-condition-specification bits.

If the conditional-chaining bit is set to 0, the Chain Address 2 field can be used for device-dependent parameters. (See Figure 1-25 on page 1-38 for bit settings that control the use of this field.)

Device Dependent Size

The Device Dependent Size field is an 8-bit field containing a count of the number of bytes in the device-dependent area of the command control block.

Chain ID

This field is reserved.

Enable Word 2

The Enable Word 2 field is a 16-bit field and is available only in the extended command control block. The field specifies additional parameters to be used when the extended command control block is processed. The following figure describes the Enable Word 2 fields.

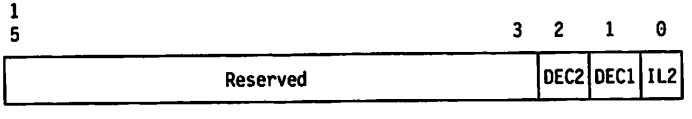


Figure 1-26. Enable Word 2 Format (Extended Command Control Block)

Bits 15–3 These bits are reserved.

Bit 2 The decrement-memory-address-2 (DEC2) bit specifies whether the memory address pointed to by the Memory Address 2 field is to be decremented or incremented when Memory Address 2 is selected. When the Memory Address 2 field is not selected, this bit has no effect.

If this bit is set to 1, the buffer addresses are decremented as data is transferred. If this bit is set to 0, the buffer addresses are incremented.

Bit 1 The decrement-memory-address-1 (DEC1) bit specifies whether the memory address pointed to by the Memory Address 1 field is to be decremented or incremented when Memory Address 1 is selected. When the Memory Address 1 field is not selected, this bit has no effect.

If this bit is set to 1, the buffer addresses are decremented as data is transferred. If this bit is set to 0, the buffer addresses are incremented.

Bit 0 The indirect-list-2 (IL2) bit specifies whether an indirect list is being used with the Memory Address 2 field. (Details on indirect lists are described in "Data Chaining" on page 4-10)

Note: This bit is valid only when both the extended-structure and the memory-address-2-selected bits in Enable Word 1 are set to 1.

When this bit is set to 1 and the memory-address-2-selected bit is set to 1, the Memory Address 2 field contains the address of an indirect list. An indirect list must begin on a doubleword boundary.

When the indirect-list-2 bit is set to 0, the Memory Address 2 field does not point to an indirect list.

Byte Count 2

The Byte Count 2 field is a 32-bit field and is available only in the extended command control block. When the Memory Address 2 field is selected, the content of this field depends on whether an indirect list is being used. When the Memory Address 2 field is not selected, this field can be used for device-dependent applications.

When an indirect list is not being used, this field contains the number of bytes in the source or destination buffer to which the Memory Address 2 field points that are to be transferred. If the count specified is 0, no data is transferred.

When an indirect list is being used, this field contains the number of bytes in the indirect list, not the number of bytes to be transferred. The byte count must be a multiple of 8 because each indirect-list entry consists of an 8-byte element (a 32-bit physical address and a 32-bit byte count). The sum of all byte counts in the indirect list is the number of bytes to be transferred. If the sum of all byte counts is 0, no data is transferred.

Termination Status Blocks

Termination status blocks are areas in shared memory that are used by a device or subsystem to report the ending status of a command control block to the software in the system master. The location of a termination status block is contained in the TSB Address field of the command control block in which it is used.

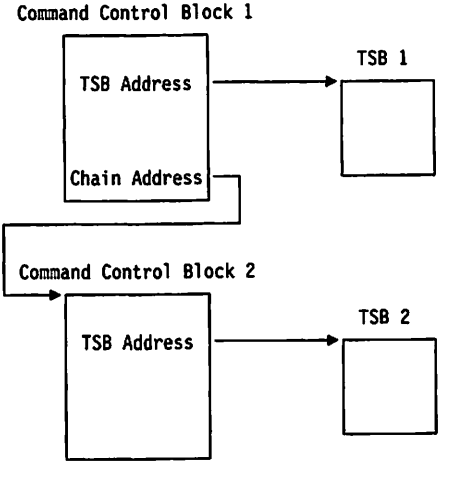


Figure 1-27. Termination Status Blocks

A termination status block is structured so that the status information it contains can be read and interpreted by software in the system master, regardless of the control-block command processed.

The area allocated to the termination status block must not be modified by the system master after its associated command control block has been delivered to the subsystem. The system software should initialize the End Status Word 1 field in the termination status block to 0 prior to delivering the command control block to the subsystem. This allows the software to do a simple nonzero check of the End Status Word 1 field in the termination status block to determine whether the termination status block was stored upon command-control-block completion.

Any command control block in a chain can suppress the storing of the termination status block for normal completion of the command control block. However, the termination status block is always stored when a command control block is suspended, terminated by an error, reports a suppressed error, or follows a conditional chain. When accompanied by an interrupt to the system master, a termination status block allows a program to monitor a device's progress through a chain of command control blocks.

The Subsystem Control Block architecture supports two formats of termination status blocks: base and extended. The difference between them is the addition of two fields for the extended format: the Residual Byte Count 2 field and the Residual Buffer Address 2 field.

The base format of the termination status block is used by a device or subsystem to report the ending status of a base command control block. A base termination status block is always 32 bytes in length.

The structure of the base termination status block is shown in the following figure.

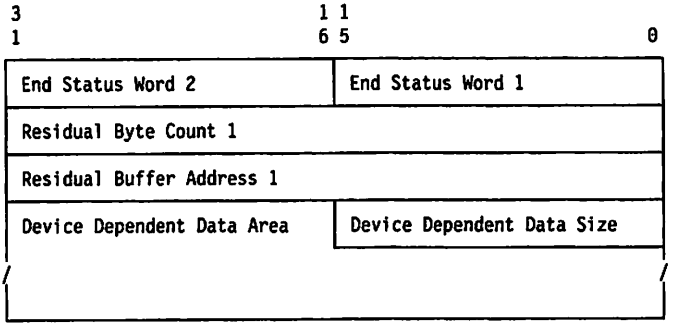


Figure 1-28. Base Termination Status Block

The extended format of the termination status block is used to report the ending status of an extended command control block. An extended termination status block is always 40 bytes in length. The following figure shows the structure of the extended termination status block and a description of each of the individual fields.

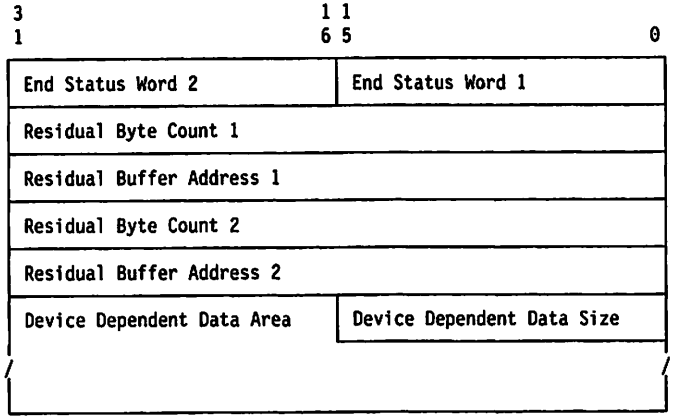


Figure 1-29. Extended Termination Status Block

End Status Word 1

End Status Word 1 is a 16-bit structured field that indicates how a command in the associated command control block was processed. Information is placed in this field by the device or a subsystem when:

- The command control block was completed with an error (including suppressed errors).
- The command control block was completed without error but had requested that a termination status block be stored unconditionally (the TSB bit in Enable Word 1 is 0).
- The command control block was suspended by a Suspend command.
- The device addressed in the Attention port supports the Device Interrupt Identifier port (refer to "Signalling Protocol" on page 3-13 for a description), and an interrupt was requested upon normal completion of the command control block.

When the termination status block is written, End Status Word 1 must be the last portion of the status information stored; the update is done prior to requesting a system interrupt. The system should reset this field to 0 prior to delivering a command control block to a device or subsystem. The system can then test this field for a nonzero value to determine whether the termination status block has been stored.

The format of End Status Word 1 is shown in the following figure. It is followed by a description of the content, meaning, and use of each of its individual bits.

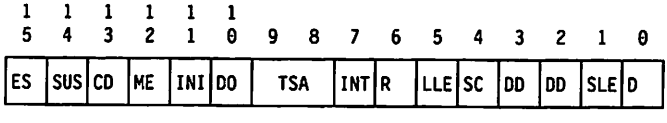


Figure 1-30. End Status Word 1 Format

Bit 15 The extended-status (ES) bit indicates how the End Status Word 2 field is being used. When this bit is 1, the End Status Word 2 field contains status information as defined by the architecture. When this bit is 0, End Status Word 2 can be used as a device-dependent field.

Bit 14 The suspended (SUS) bit indicates that a Suspend command has been received during the execution of this command control block. The device has entered the suspend state after completing the execution of this control block.

When this bit is 1, the processing of the associated command control block has been completed; however, the device is in the suspended state and will not allow execution of any more command control blocks. When this bit is 0, the processing of control blocks has not been suspended.

Bit 13 The chain-direction (CD) bit indicates whether the address in the Chain Address 2 field of the current command control block has been used to obtain the next command control block in the chain.

When this bit is 1, the Chain Address 2 field has been used as the pointer to the next command control block in the chain. When this bit is 0, the Chain Address 2 field has not been used.

Bit 12 The major-error/exception (ME) bit is used to indicate whether a major error or exception has occurred in the processing of the associated command control block.

When this bit is 1, the command in the associated command control block has been completed unsuccessfully. The interrupt-requested bit is also set to 1 because an interrupt must be requested when a major error or exception is detected.

When this bit is set to 0, the command in the command control block has been completed without major errors or exceptions.

Bit 11 The device-not-initialized (INI) bit indicates whether a device has completed initialization.

When this bit is 1, the device has not completed the initialization sequence. When this bit is 0, the device has completed the initialization sequence.

Bit 10 The device-overrun (DO) bit indicates that a device overrun has been detected.

When this bit is 1, the device has lost data because of an overrun. The device also sets the major-error/exception and interrupt-requested bits to 1. When this bit is set to 0, the device has not detected an overrun.

See "Device-Overrun Exception" on page 5-14 for a definition of the condition and the actions taken.

Bits 9, 8 The TSB-available (TSA) bits contain an encoded value that indicates the amount of status information stored in the termination status block and whether more status information is available. The value and meaning of these bits are as shown in the following figure.

Value	Meaning
00	Status information has been placed in End Status Word 1.
01	All fields, except for the Device Dependent Data Size and the Device Dependent Area, contain status information.
10	All fields, including the Device Dependent Area, contain status information. The Device Dependent Data Size field specifies the number of bytes of device-dependent data stored. Up to 18 bytes of device-dependent data can be stored in a termination status block. If the value in the Device Dependent Data Size field exceeds 18, the Read Completion Status command must be used to retrieve this additional data. See "Read Completion Status" on page 1-60 for more details on Read Completion Status.
11	All fields of the termination status block, except for the Residual Buffer Address and Residual Byte Count fields, contain status information. The value of the Device Dependent Data Size field specifies the number of bytes of device-dependent data stored.

Figure 1-31. TSB-Available Bit Encoding

Bit 7 The interrupt-requested (INT) bit indicates whether an interrupt has been requested for the associated command control block. When this bit is set to 0, an interrupt has not been requested.

When this bit is 1, an interrupt has been requested for the completed command control block.

Bit 6 This bit is reserved.

Bit 5 The long-length-exception (LLE) bit indicates whether a long-length exception has been detected.

See "Long-Length Exception" on page 5-11 for the definition of the condition and the actions taken when the condition occurs. When this bit is set to 1, a long-length exception has occurred. When this bit is set to 0, a long-length exception has not occurred.

Bit 4 The specification-check (SC) bit indicates that an invalid field has been detected in the command control block. Checking the command control block for valid fields is optional.

When this bit is 1, checking has been performed, and an invalid field has been found. The operation is terminated, and no data is transferred. When this bit is 0, either the fields have not been checked or an invalid field was not detected.

See "Specification Exception" on page 5-13 for definitions of the conditions and the actions to be taken when a condition is detected.

Bits 3, 2 These bits (DD) are device dependent and contain information meaningful only to the specific device implementation.

Bit 1 The short-length-exception (SLE) bit indicates whether a short-length exception has been detected. When this bit is set to 1, a short-length exception has occurred. When this bit is set to 0, a short-length exception has not occurred. See "Short-Length Exception" on page 5-9 for the definition of the condition and the actions taken when the condition is detected.

Bit 0 The done (D) bit indicates whether the command in the associated command control block was completed without error.

When this bit is 1, the associated command control block has been completed without an error or exception. When this bit is 0, other bits defined in End Status Word 1 must be checked to determine the status of the associated command control block.

End Status Word 2

The End Status Word 2 field is a 16-bit structured field used to hold additional status information. This field can also be used as a device-dependent field, depending on the setting of the extended-status bit in End Status Word 1.

The architected format of End Status Word 2 is shown in the following figure and is followed by a description of the individual bits.

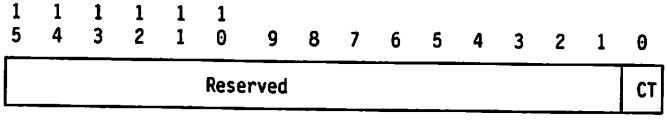


Figure 1-32. End Status Word 2 Format

Bits 15–1 These bits are reserved.

Bit 0 The command-type (CT) bit is always set to 0.

Residual Byte Count 1

The Residual Byte Count 1 field is a 32-bit field that is aligned on a doubleword boundary and contains the number of bytes that remain to be read into, or written from, the buffer identified by the address in the Residual Buffer Address 1 field.

This field must be provided when either a short-length or long-length exception condition is indicated. If neither the short-length-exception bit nor the long-length-exception bit in End Status Word 1 is used, this field is optional. For more detail see "Residual Byte Count" on page 5-19.

Residual Buffer Address 1

The Residual Buffer Address 1 field is a 32-bit field that is aligned on a doubleword boundary and identifies the buffer last read from, or written to, in the associated command control block.

If the associated command control block buffer is not a data chain element in an indirect list, this field contains the address of the buffer (Memory Address 1 or Memory Address 2) as indicated by bits in Enable Word 1 of the command control block.

If the associated command control block buffer is a data chain element in an indirect list, this field contains the address of the buffer in the data-chain element that was most recently used to transfer data.

This field must be provided when either a short-length or long-length exception condition is indicated. If neither the short-length-exception bit nor the long-length-exception bit in End Status Word 1 is used, this field is optional. For more detail see "Residual Buffer Address" on page 5-20.

In the extended termination status block, this field pertains only to the Memory Address 1 field of the command-control-block structure. It is used to identify the data transfer buffer that was located through Memory Address 1.

Note: Software in the system master can use the Residual Buffer Address 1 and Residual Buffer Count 1 fields to determine how far a command has progressed toward completion. The encoded value of the TSB Available (TSA) field in End Status Word 1 indicates whether these two fields have been stored by the device. For more details, see "Number of Bytes Transferred" on page 5-21.

Residual Byte Count 2

The Residual Byte Count 2 field is present only in an extended termination status block. It is a 32-bit field that is aligned on a doubleword boundary and contains the number of bytes that remain to be read into, or written from, the buffer identified in the Residual Buffer Address 2 field.

This field must be provided when either a short-length-exception or long-length-exception condition is indicated by the short-length-exception (SLE) bit or the long-length-exception (LLE) bit in End Status Word 1. If neither the short-length-exception (SLE) bit nor the long-length-exception (LLE) bit is used, this field is optional and does not need to be reported. Refer to "Residual Byte Count" on page 5-19 for a detailed description of this field.

Residual Buffer Address 2

The Residual Buffer Address 2 field is present only in an extended termination status block. It is a 32-bit doubleword-aligned field identifying the buffer last written to or read from in the associated command control block.

If the associated command-control-block buffer is not a data-chain element in an indirect list, this field contains the address of the buffer (Memory Address 2).

If the associated command-control-block buffer is a data-chain element in an indirect list, this field contains the address of the buffer in the data-chain element that was most recently used to transfer data.

This field must be provided when either a short-length-exception or long-length-exception condition is indicated by the short-length-exception (SLE) bit or the long-length-exception (LLE) bit in End Status Word 1. If neither the short-length-exception (SLE) bit nor the long-length-exception (LLE) bit is used, this field is optional and does not need to be reported. Refer to "Residual Buffer Address" on page 5-20 for a detailed description of this field.

Note: Software in the system master can use the Residual Buffer Address 1, Residual Buffer Count 1, Residual Buffer Address 2, and Residual Buffer Count 2 fields to determine how far a command has progressed toward completion. The encoded value of the TSB Available (TSA) field in End Status Word 1 indicates whether these two fields have been stored by the device. For more details, see "Number of Bytes Transferred" on page 5-21.

Device Dependent Data Size

The Device Dependent Data Size field is a 16-bit field that is used to indicate the number of bytes of device-dependent data stored by a device or subsystem as a result of the associated command control block. If the value of this field exceeds 18 bytes, a Read Completion Status control-block command must be issued to retrieve the data beyond byte 18. Refer to "Read Completion Status" on page 1-60 for details on the Read Completion Status command.

The validity of this field is determined by examining the encoded value of the TSB-available bit in End Status Word 1.

Device Dependent Data Area

The Device Dependent Data Area is an 18-byte field used to hold a maximum of 18 bytes of device-dependent data. The amount of device-dependent data contained within the field is indicated by the value in the Device Dependent Size field.

Indirect-List Structure

An indirect list is a variable-length list consisting of the address-count pairs used to support data chaining. Both the location of the indirect list and its length are specified in the control block. Within the indirect list, the Address field and the Count field are each doubleword fields. The format and content of an indirect list is shown in the following figure.

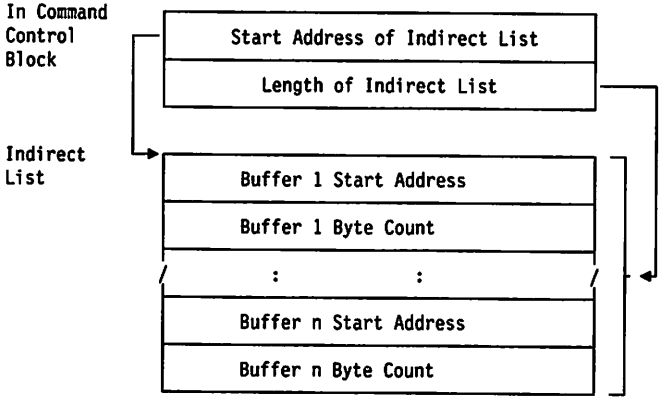


Figure 1-33. Indirect-List Format

Commands

In the Locate mode, there are two types of commands: control-block commands and immediate commands.

Control-Block Commands

The Locate mode architecture defines a number of control-block commands that have common uses. This section describes these commands and their associated command control blocks. These commands are defined in alphabetic order. The architected control-block commands are:

- Initialize Device
- No Operation
- Read
- Read Completion Status
- Read Configuration
- Run Diagnostic Test
- Write.

Control-block commands are primarily concerned with the transfer of data. Therefore, the semantics of commands that transfer data follow those of the Read and Write commands.

All control-block commands can be in the form of the base or extended command control blocks and can use indirect lists. The semantics of commands transferring indirect data in read and write operations are defined in "Reading with a Base Control Block and Indirect Lists" on page 4-12 and "Writing with a Base Control Block and Indirect Lists" on page 4-14. The semantics of interrupts used in control-block commands follow those described in sections "Signalling Protocol" and "Exception Conditions" on page 5-9.

Initialize Device

The Initialize Device command is used to write initialization data to a device. Prior to receiving initialization data, a device might not be able to execute its entire command set. However, prior to initialization, the device must be able to execute the Read Configuration command and the Initialize Device command.

The Memory Address 2 field points to initialization data, and the Byte Count 1 field defines the amount of initialization data to be read by the device. Data is transferred to the address in a device's memory, using the same semantics as the Write command.

The Initialize Device command should be directed only to a device that requires initialization and that has been put into a state in which it can be initialized. Attempting to initialize a device that does not require initialization can cause a specification exception. Software in the system master can determine whether a device needs to be initialized by issuing the Read Configuration command.

The Initialize Device command has the option to request an interrupt when the command is completed without an error or with a suppressed error.

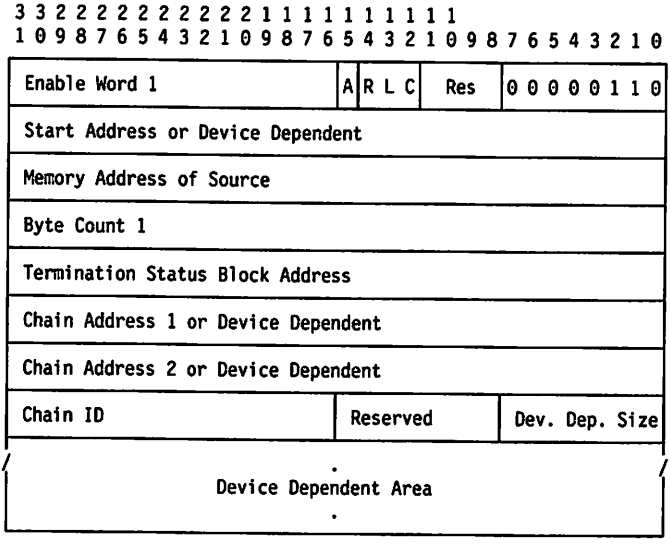


Figure 1-34. Initialize Device Control Block

Options

The following options are used with the Initialize Device command; if the device or subsystem does not support these options, a specification error can result.

Bit 14 The restart (R) bit specifies whether a device restarts initialization or continues from a previous point. If this bit is set to 1, the initialization sequence starts at the beginning. If this bit is set to 0, the sequence continues from a prior point.

Bit 13 The location (L) bit specifies whether the Start Address field, used as a data load point, is a destination address that is internal to the device. If this bit is set to 1, the Start Address field is an internal load point. If this bit is set to 0, the Start Address field is available for device-dependent applications.

Bit 12 The complete (C) bit specifies whether the device should complete initialization with this command. If this bit is set to 1, the device should complete initialization. If this bit is set to 0, the device should not complete initialization.

If this bit is set to 1 and the device can complete its initialization, the device-not-initialized bit in End Status Word 1 is set to 0. If initialization cannot be completed when this bit is selected, the device-not-initialized bit is set to 1.

No Operation

The No Operation (Noop) command initiates a branching structure within command chains or requests an interrupt that can signal a progress point in a command chain.

No data transfer to buffer areas in shared memory is performed. Memory Address 1 and Memory Address 2 fields do not specify buffer areas.

This command has the option to request an interrupt when the command is completed.

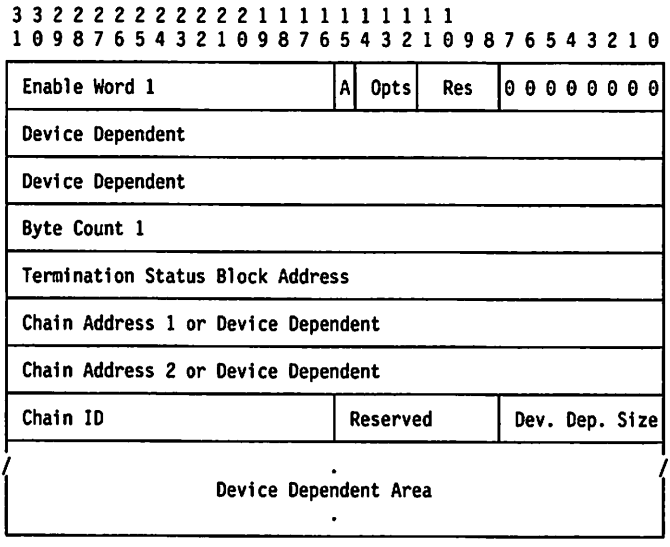


Figure 1-35. No-Operation Control Block

Read

The Read command transfers data from the device specified in the Attention port (source) to a location in shared memory that is pointed to by the Memory Address 2 field (destination).

This command has the option to request an interrupt when the command is completed without error or with a suppressed error.

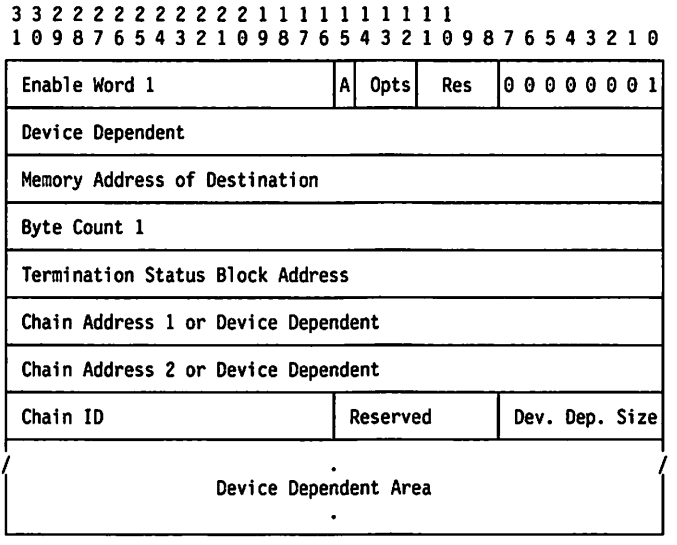


Figure 1-36. Read Control Block

If an indirect list is specified in this control block (the indirect-list-1 bit of Enable Word 1 is set to 1), semantics for data movement are defined in "Reading with a Base Control Block and Indirect Lists" on page 4-12.

If an indirect list is not specified (the indirect-list bit of Enable Word 1 is set to 0), the Memory Address 2 field contains the starting address of the destination area. The Byte Count 1 field contains the length, in bytes, of the data to be transferred. If the Byte Count 1 field is 0, no data is transferred.

A running count of the number of bytes transferred is maintained within the device (residual byte count). This count is developed from the initial value in the Byte Count 1 field. As each byte is transferred from the device, the Residual Byte Count field is decreased by 1. The operation is completed when the Residual Byte Count field is 0 or the device has no more data to transfer. The address in the Memory Address 2 field is copied to a Residual Buffer Address field in the termination status block, and the residual byte count is transferred to the Residual Byte Count field in the termination status block.

When a command control block is completed, a program can obtain the residual byte count and residual buffer address from the

associated termination status block to determine the progress of the operation. Storing these fields in the termination status block is optional, except when short-length or long-length exception conditions are detected.

Note: Refer to "Exception Conditions" on page 5-9 for a description of short-length and long-length exceptions.

If the residual byte count and residual buffer address are not stored for a read operation that is completed with an error, the entire operation should be performed again.

Data transfers during a read operation can also be terminated because of irrecoverable errors at the device (other than short-length and long-length exceptions). The major-error/exception and interrupt-requested bits in the End Status Word 1 are set to 1 to indicate that a major error occurred and that an interrupt was requested.

Read Completion Status

The Read Completion Status command allows a program to retrieve the completion status of the previous immediate or control-block command from a device. The data is moved from the device to an area that is pointed to by the Memory Address 2 field.

Completion status data is transferred to the system master, using the same semantics as the Read command.

This command must be implemented if a device has the potential to report more status than can be presented in the termination status block, or if the device has the potential to report status in response to immediate commands.

The Read Completion Status command has the option to request an interrupt when the command is completed without error or with a suppressed error.

The maximum size of the completion status information depends on the device and can be determined with the Read Configuration command.

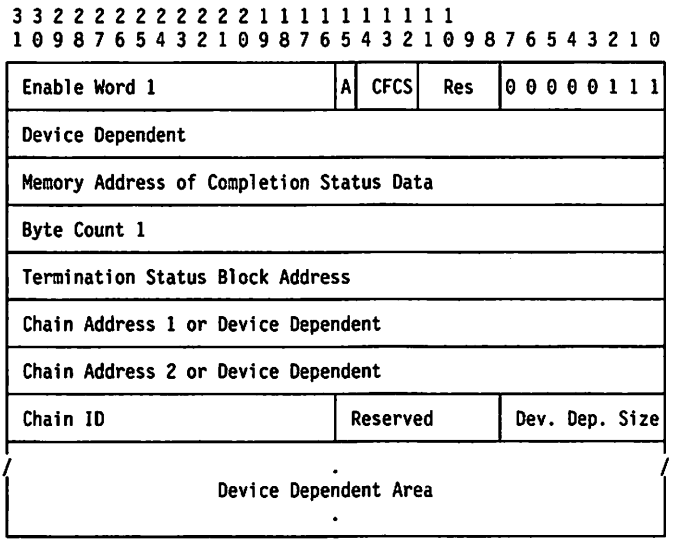


Figure 1-37. Read Completion-Status Control Block

Byte Count 1 specifies the number of bytes of completion status data that the software will read from the device.

The completion status data returned to the system is dependent on the encoding of the Command for Completion Status field, as described in the following table. An invalid or unsupported encoding of this field might raise a specification error.

Value	Description
000	The completion-status data to be returned is for the last command (either immediate or control block) that was executed by the device prior to execution of this Read Completion Status command.
001	The completion-status data to be returned is for the last control-block command that was executed by the device prior to execution of this Read Completion Status command.
010	The completion-status data to be returned is for the last immediate command that was executed by the device prior to execution of this Read Completion Status command.
All other values are reserved.	

Figure 1-38. Command for Completion Status Field Encoding

If the device does not have completion status to report that matches the specification given in the Command for Completion Status field, a short-length exception condition can be returned to the system master.

The format of the data returned in response to the Read Completion Status command is shown in the following figures. The data block returned depends on whether the status corresponds to a control-block command or to an immediate command. The following figure shows the format of completion-status data when the previously executed command was an immediate command. Figure 1-40 on page 1-63 shows the format of completion-status data when the previously executed command was a control-block command.

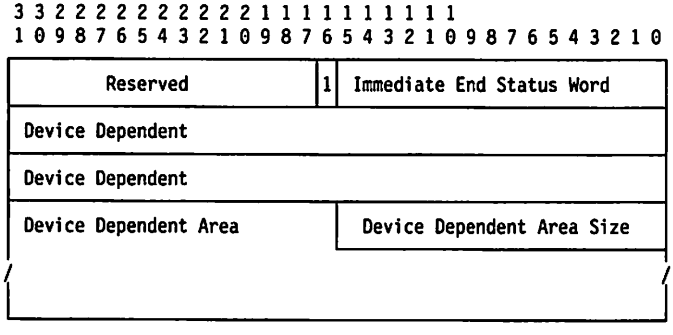


Figure 1-39. Completion-Status Data Block for Immediate Commands

The format, content, and description of the Immediate End Status Word are the same as those of End Status Word 1 of the termination status block defined in "End Status Word 1" on page 1-46.

Note: It is suggested that the device return (as device-dependent data) the immediate command or the address of the control block with the completion status.

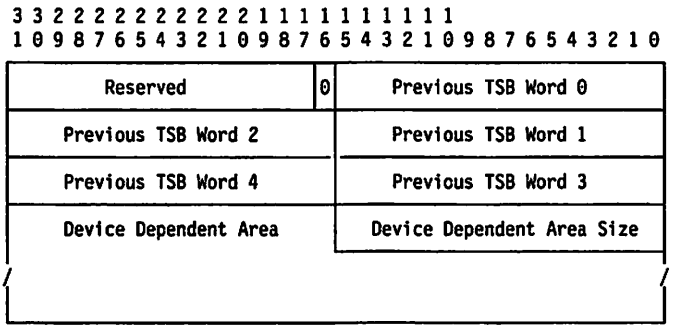


Figure 1-40. Completion-Status Data Block for Base Control Blocks

Read Configuration

The Read Configuration control-block command is used to obtain parametric data about the subsystem or the devices attached to the subsystem. The device must always be capable of executing this command, even if the device requires initialization using the Initialize Device command.

Configuration data is transferred to the system master, using the same semantics as the Read command. Data is delivered to the system master in the location pointed to by the Memory Address 2 field.

The Read Configuration command has the option to request an interrupt when the command is completed without error or with a suppressed error.

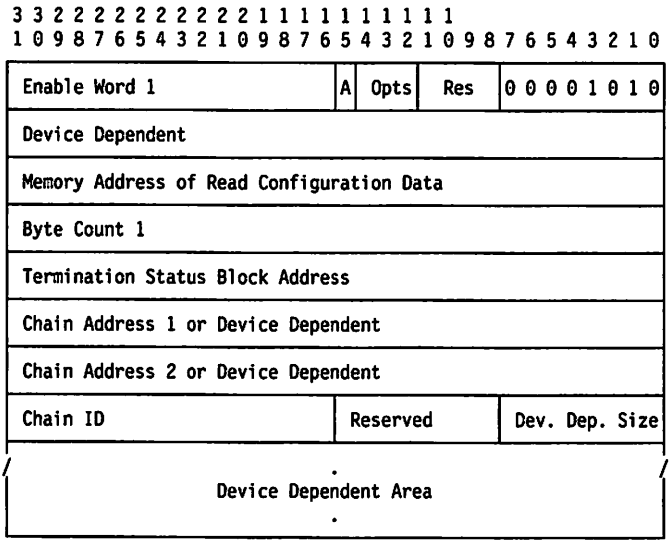


Figure 1-41. Read Configuration Control Block

Byte Count 1 specifies the number of bytes of configuration data that software will read from the device. Byte Count 1 should contain a value that is equal to at least 44 bytes. Data read beyond the first 44 bytes is device dependent.

If a long-length-exception condition is detected, the system master can determine the maximum amount of configuration data by checking the device-dependent-area size and reissuing the command with the appropriate byte count.

If a short-length-exception condition is detected, the number of valid bytes returned depends upon the value specified for Byte Count 1.

The following figure shows the format of the read-configuration data block returned to the system. It is followed by a description of each of the configuration data fields.

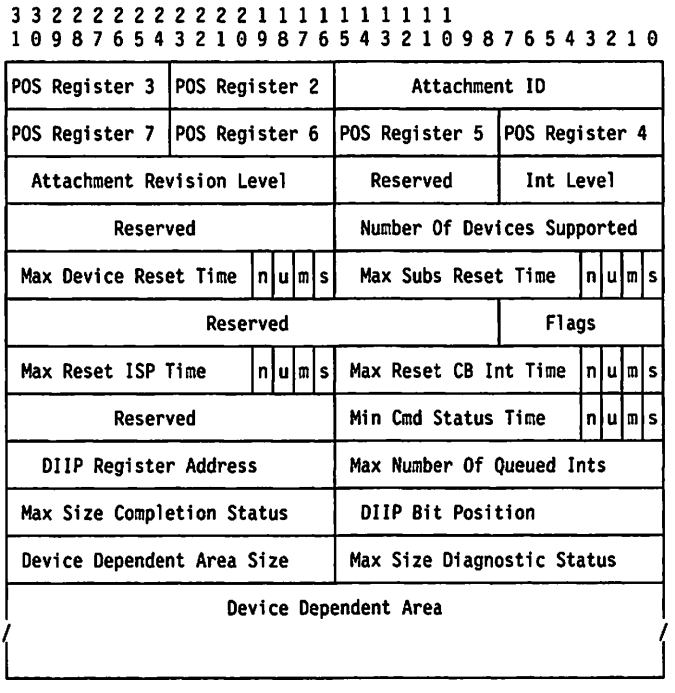


Figure 1-42. Read Configuration Data Block

Attachment ID

The Attachment Identification field is a 16-bit field that contains the ID of the subsystem.

POS Register 2

The POS Register 2 field is an 8-bit field that reflects the contents of POS Register 2.

POS Register 3

The POS Register 3 field is an 8-bit field that reflects the contents of POS Register 3.

POS Register 4

The POS Register 4 field is an 8-bit field that reflects the contents of POS Register 4.

POS Register 5

The POS Register 5 field is an 8-bit field that reflects the contents of POS Register 5.

POS Register 6

The POS Register 6 field is an 8-bit field that reflects the contents of POS Register 6.

POS Register 7

The POS Register 7 field is an 8-bit field that reflects the contents of POS Register 7.

Interrupt Level (Int Level)

The Interrupt Level field is an 8-bit field that contains the interrupt level assigned to the specified device at the end of system initialization.

Reserved

These areas are reserved.

Attachment Revision Level

The Attachment Revision Level field is a 16-bit field that indicates the revision level of the subsystem.

Number of Devices Supported

The Number of Devices Supported on the Subsystem field is a 16-bit field that indicates the maximum number of devices that can be attached to the subsystem.

Maximum Subsystem Reset Time

The Maximum Subsystem Reset Time field is a 16-bit field that contains the maximum time for the subsystem to reset. It can be specified in seconds, milliseconds, microseconds, or nanoseconds, depending on the setting of the 4 lower-order bits. The bits are defined in the following figure.

Bit Name	Description
s	The seconds bit, when set to 1, indicates that the field is specified in seconds.
m	The milliseconds bit, when set to 1, indicates that the field is specified in milliseconds.
u	The microseconds bit, when set to 1, indicates that the field is specified in microseconds.
n	The nanoseconds bit, when set to 1, indicates that the field is specified in nanoseconds.
Note: One and only one bit must be set to 1. All other combinations are reserved.	

Figure 1-43. Time-Field Specifier

Maximum Device Reset Time

The Maximum Device Reset Time field is a 16-bit field that contains the maximum time for the device to reset. It can be specified in seconds, milliseconds, microseconds, or nanoseconds, depending on the setting of the 4 lower-order bits as indicated in Figure 1-43.

Flags

The Flags field is an 8-bit field that consists of the following indicators.

Bits 7–4 These bits are reserved.

Bit 3 The device-dependent-data-available bit (bit 3) indicates whether the specified device supports the device-dependent data in the termination status block. See Figure 1-31 on page 1-48 for a description of the TSB-available bits. The maximum amount of device-dependent data is specified in the Maximum Size for Completion Status field. When the bit is set to 1, the device supports device-dependent data available in the termination status block. When the bit is set to 0, the device does not support device-dependent data available in the termination status block.

Bit 2 The interrupt-support bit (bit 2) indicates whether the device uses the Device Interrupt Identifier port. When the bit is set to 1, the device uses the Device Interrupt Identifier port. When the bit is set to 0, the device does not support the Device Interrupt Identifier port.

Bit 1 The reset bit (bit 1) indicates whether the Reset Subsystem command resets all devices. When the Reset bit is set to 1, the Reset Subsystem command resets all devices. When the Reset bit is set to 0, the Reset Subsystem command does not reset all devices attached to the subsystem. In this case, individual Reset Device commands are required.

Bit 0 The load bit (bit 0) indicates whether the specified device requires a program load using the Initialize Device command. If this bit is set to 1, a program load is required. If this bit is set to 0, the device never requires a program load.

Reserved

This field is reserved.

Maximum Reset Control Block Interrupt Time

The Maximum Reset Control Block Interrupt Time field is a 32-bit field that indicates the maximum time the addressed device takes to perform a Reset Control Block Interrupt command. It can be specified in seconds, milliseconds, microseconds, or nanoseconds, depending on the setting of the 4 lower-order bits as indicated in Figure 1-43 on page 1-67.

Maximum Reset Interrupt Status Port Time

The Maximum Reset Interrupt Status Port Time field is a 32-bit field that indicates the maximum time the specified device takes to perform a Reset Interrupt Status Port command. It can be specified in seconds, milliseconds, microseconds, or nanoseconds, depending on the setting of the 4 lower-order bits as indicated in Figure 1-43 on page 1-67.

Maximum Command/Busy Status Time

The Maximum Command/Busy Status Time field is a 32-bit field that indicates the maximum time required to present status for an incoming command to the Command Busy/Status port. It can be specified in seconds, milliseconds, microseconds, or nanoseconds, depending on the setting of the 4 lower-order bits as indicated in Figure 1-43 on page 1-67.

Maximum Number of Queued Interrupts

The Maximum Number of Queued Interrupts field is a 32-bit field that indicates the maximum number of interrupts that can be internally queued by the specified device.

DIIP Address

The DIIP Address field is a 16-bit field that indicates the I/O address of the Device Interrupt Identifier port for the specified device. This field is valid only when the interrupt-support bit in the Flags field is set to 1.

DIIP Bit Position

The DIIP Bit Position field is a 16-bit field that indicates the bit position assigned to the specified device in the Device Interrupt Identifier port. This field is valid only when the interrupt-support bit in the Flags field is set to 1.

Maximum Size for Completion Status

The Maximum Size for Completion Status field is a 16-bit field that indicates the maximum number of bytes needed to hold the completion-status data block for the specified device. See "Read Completion Status" on page 1-60 for a complete description of this command.

Maximum Size for Diagnostic Status

The Maximum Size for Diagnostic Status field is a 16-bit field that indicates the maximum number of bytes needed to hold diagnostic-status data as a result of the Run Diagnostic Test command. See "Run Diagnostic Test" on page 1-70 for a complete description of this command.

Device Dependent Area Size

The Device Dependent Area Size field is a 16-bit field that indicates the number of bytes in the Device Dependent Configuration Data Area field.

Device Dependent Configuration Data Area

The Device Dependent Configuration Data Area field is a variable-length field containing device-dependent configuration data. The use of this field is device dependent and can be determined by reading the device specifications.

Run Diagnostic Test

The Run Diagnostic Test command causes the specified device to run diagnostic tests and to return the results to the location pointed to in the Memory Address 2 field. The data transfer uses the same semantics as the Read command.

The maximum size of the data area needed by the device to successfully store its diagnostic status is obtained by the Read Configuration command. The format, meaning, content, and uses of the data returned by the command are device dependent.

The Run Diagnostic Test command has the option to request an interrupt when the command is completed without error or with a suppressed error.

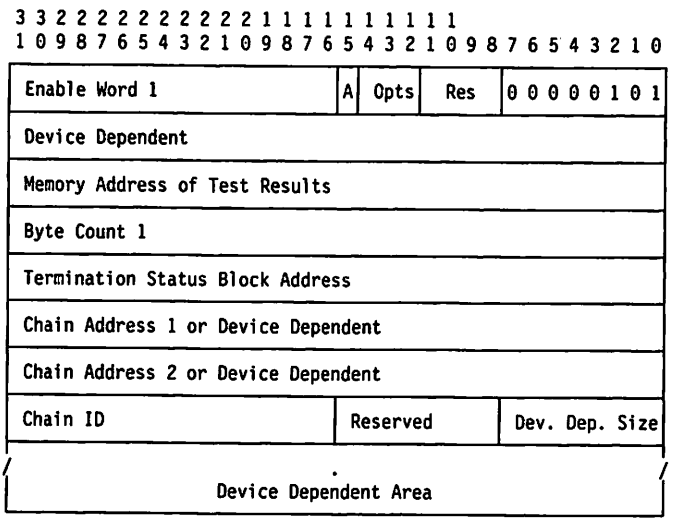


Figure 1-44. Run-Diagnostic-Test Control Block

The Byte Count 1 field specifies the number of bytes of diagnostic data to be returned.

Write

The Write command transfers data from the location in shared memory pointed to by the Memory Address 2 field to the device specified in the Attention port.

The Write command has the option to request an interrupt when the command is completed without error or with a suppressed error.

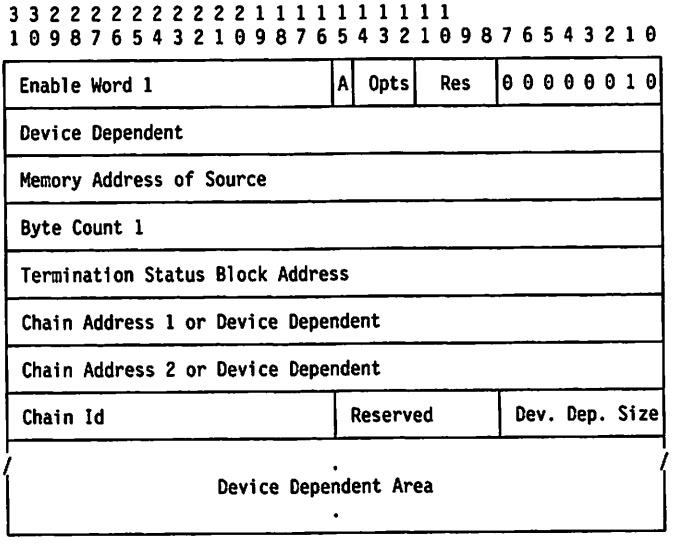


Figure 1-45. Write Control Block

If an indirect list is specified, the semantics for data movement are defined in "Writing with a Base Control Block and Indirect Lists" on page 4-14.

If an indirect list is not specified, the Memory Address 2 field is the starting address of the source data in shared memory, and the Byte Count 1 field contains the number of bytes to be transferred.

The Byte Count 1 field is copied to a residual byte count, which is used as a counter during the transfer. As each byte is transferred from shared memory to the device, the residual byte count is

decreased by 1. The operation is completed normally when the residual byte count is 0 and all bytes are received by the device. The address in the Memory Address 2 field is copied to a residual buffer address, which is used as the current pointer during the transfer.

These two residual values can be stored in the termination status block. After the values are stored, a program can determine the progress of the operation by obtaining the residual byte count and residual buffer address from the associated termination status block. When the residual byte count and residual buffer address are not stored in the termination status block and an error occurs, the entire operation should be performed again.

Transfer of data might be terminated because of an irrecoverable error at the device. When this happens, the major-error/exception and interrupt-requested bits in the End Status Word 1 are set to 1.

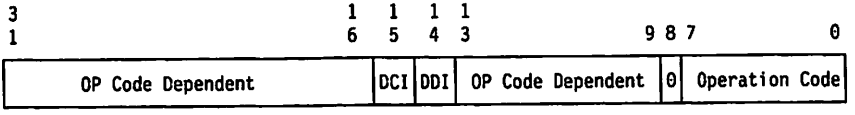
Immediate Commands

This section contains general descriptions of the immediate commands defined in the Locate mode. These commands are control oriented and use the same delivery interface as control-block commands (that is, delivery using the Command Interface and Attention ports). The immediate commands are:

- No Operation (Noop)
- Reset Device
- Reset Interrupt Status Port
- Reset Control Block Interrupt
- Reset Subsystem (software-controlled)
- Resume
- Run Immediate Diagnostic Test
- Suspend
- Reset Subsystem (hardware-controlled).

An immediate command uses one of two formats, depending on the value of the format identifier (bit 8) in the immediate command. Format 0 allows a command to use bits 31-16 for operation-code-dependent purposes. This gives the implementer additional flexibility. Format 1 reserves bits 31-16 of the immediate command. This allows architecture-defined commands the flexibility to grow. All commands described in this document are Format 1.

Immediate Command – Format 0



Immediate Command – Format 1

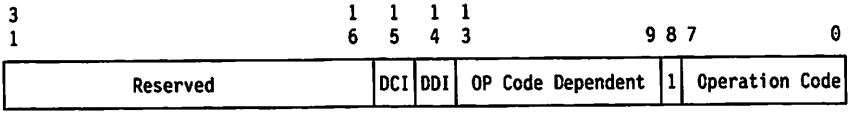


Figure 1-46. Immediate-Command Formats

The following is a description of the fields of an immediate command and their meanings:

Bits 31 – 16 Refer to bit 8.

Bit 15 The disable-command-interrupt (DCI) bit specifies whether the device requests an interrupt after successfully completing this command. If this bit is set to 1, the device does not request a system interrupt. If this bit is set to 0, the device requests an interrupt after completing the command and reports the status to the Interrupt Status port. For some commands, this bit is ignored.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is enabled to generate interrupts to the system. If this bit is set to 1, interrupts are disabled for the device specified. If this bit is set to 0, interrupts are enabled for the device specified. When interrupts are disabled, the device must be able to internally store interrupts until the system enables the interrupts.

Bits 13 – 9 The Op Code Dependent field is defined for each command.

Bit 8 The format-identifier bit determines the format used by the immediate command. If this bit is set to 0, Format 0 is used, allowing the immediate command to use bits 31-16 for operation-code-dependent purposes. If this bit is set to 1, Format 1 is used, causing bits 31-16 of the immediate command to be reserved.

Bits 7–0 The Operation Code field identifies the specific operation to be performed by the immediate command. Format 0 operation codes are implementation dependent. Format 1 operation codes are defined in the following figure.

Operation-Code Bits	Description
76543210	
00000000	Reset device
00000001	Reserved
00000010	Noop
00000011	Reserved
00000100	Reset Interrupt Status port
00000101	Reserved
00000110	Reserved
00000111	Reserved
00001000	Reset Control Block interrupt
00001001	Reserved
00001010	Reserved
00001011	Reserved
00001100	Device dependent
00001101	Device dependent
00001110	Device dependent
00001111	Device dependent
00010000	Reserved
00010001	Reserved
00010010	Run Immediate Diagnostics
00010011	Device dependent
00010100	Reserved
00010101	Reserved
00010110	Device dependent
00010111	Device dependent
00011000	Reserved
00011001	Reserved
00011010	Reserved
00011011	Reserved
00011100	Reserved
00011101	Reserved
00011110	Reserved
00011111	Suspend
00100000	Resume
All combinations of bits 7-0 not shown are reserved.	

Figure 1-47. Operation-Code Bits — Format 1

Some immediate commands use interrupts to reply to the system master. The semantics of interrupts are defined in "Signalling Protocol" on page 3-13. Details on exception handling are found in "Exception and Error Handling" on page 5-4.

No Operation (Noop)

The No Operation command can change the interrupt-enablement status of the device specified in the Attention port without causing any other effect.

This command can request an interrupt, upon successful completion of the command, by setting the DCI bit to 0.

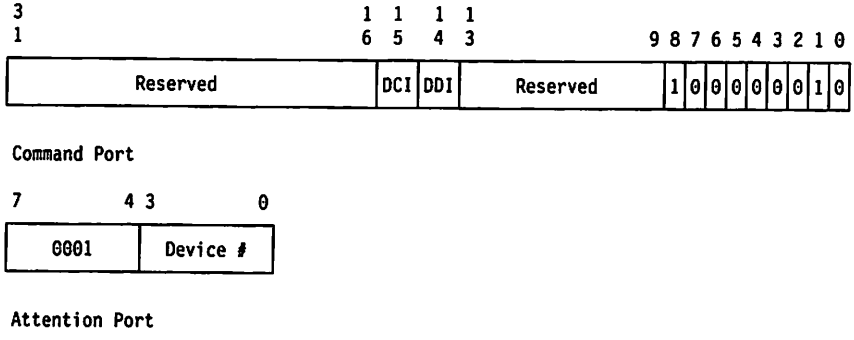


Figure 1-48. No-Operation Command Structure

Bit 15 The disable-command-interrupt (DCI) bit specifies whether the device requests an interrupt after successfully completing this command. If this bit is set to 1, the device does not request a system interrupt. If this bit is set to 0, the device requests an interrupt after completing the command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the device specified. If this bit is set to 0, interrupts are enabled from the device specified.

Bits 13–9 These bits are reserved.

Programming Note: A Noop command will be rejected by a device that is currently executing a control block.

Reset Device

The Reset Device command is used to put the specified device into a known state. A device number of 0 directs the command to the subsystem. The known device state is defined as follows:

- All pending interrupts for the device are cleared. If the device supports Device Interrupt Identifier ports, the internal count of logical interrupts is set to 0, and the DIIP bit is set to 0.
- Any control block or control-block chain that is currently being executed is purged.
- Any control-block chain that is suspended at the device is purged.
- The device is restored to its default setting (device interrupts are enabled or disabled depending on the value of the DDI bit in the Immediate Reset Device command).
- The device executes any device-dependent reset activities.

A Reset Device command always requests an interrupt upon successful completion.

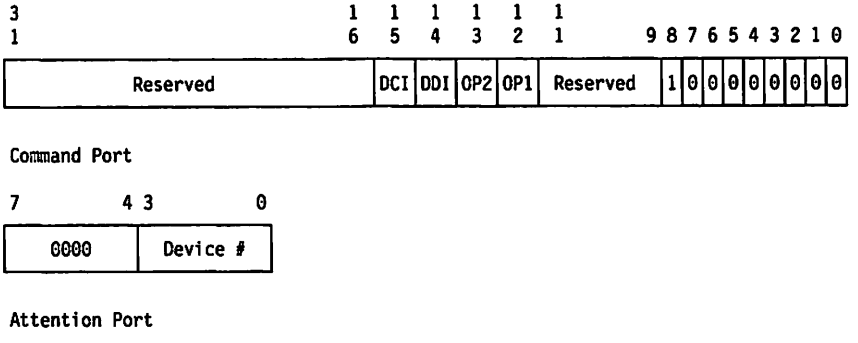


Figure 1-49. Reset Device Command Structure

Bit 15 The disable-command-interrupt (DCI) bit is ignored by the device. An interrupt is always requested on completion of this command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the device specified. If this bit is set to 0, interrupts are enabled from the device specified.

Bit 13 The option-2 (OP2) bit specifies whether the subsystem and all its attached devices are reset, or whether just the subsystem is reset. If the device number is not 0, the option-2 bit is ignored. If the device number is 0 and this bit is set to 0, the subsystem and all its attached devices are reset. If the device number is 0 and this bit is set to 1, the subsystem is reset, but all attached devices are not reset.

Bit 12 The option-1 (OP1) bit specifies whether a partial or complete reset of the device is performed. When this bit is set to 0, the device is reset completely so that the device must be reinitialized. The device's internal memory is reset. If this bit is set to 1 and the device has already been initialized, the device is partially reset and does not need to be reinitialized. The device's internal memory is not reset.

Programming Notes:

1. The Reset Device command should be used to force a device into a known state. It should, however, be used with care in error-recovery situations because all pending interrupts are lost when the device is reset.
2. When a Reset Device command is executed to device 0, with the option-2 bit set to 1, only the internal states of commands directed to device 0 are affected. Other devices attached to the subsystem are not reset.
3. An interrupt from a prior command for a device might be present in the Interrupt Status port when a Reset Device command is issued. A program must receive and ignore this interrupt by reading and resetting the Interrupt Status port without processing the interrupt. The program should test for an interrupt identifier of 0 because a Reset Device command always enqueues an

interrupt with an identifier of 0 when the command is completed without error.

For details on error handling see "Subsystem Level" on page 5-6.

Reset Interrupt Status Port

This command resets an interrupt in the Interrupt Status port. It clears the Interrupt Status port only if the device specified in the Attention port is the same as the device associated with the interrupt in the Interrupt Status port. If the Interrupt Status port is cleared, the interrupt-valid bit in the Command Busy/Status port is set to 0. If the device specified does not match the device in the Interrupt Status port, the Interrupt Status port is not cleared, and the disable-device-interrupt bit is the only bit checked by the device.

When the Interrupt Status port is cleared by this command, the physical interrupt to the system master is reset unless the Device Interrupt Identifier port is used. When the Device Interrupt Identifier port is used, the interrupt is reset only when all bits in the Device Interrupt Identifier port are 0.

This command does not request an interrupt upon completion.

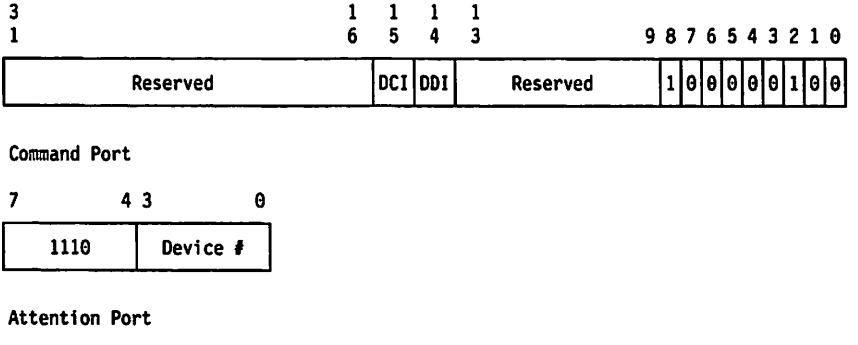


Figure 1-50. Reset Interrupt Status Port Command Structure

Bit 15 The disable-command-interrupt (DCI) bit is ignored by the device. An interrupt is never requested on completion of this command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the device specified. If this bit is set to 0, interrupts are enabled from the device specified.

Programming Notes:

1. A device that is currently executing a command must accept the Reset interrupt Status Port command. This command allows a program to reset interrupts that occur within a chain of control blocks without having to issue a Suspend command to the device.
2. Refer to "Signalling Protocol" on page 3-13 for details on interrupt handling.

Reset Control Block Interrupt

The Reset Control Block Interrupt command resets logical interrupts received for control-block commands when the Device Interrupt Identifier port is used. It causes the internal logical-interrupt count of the device specified to be decreased by the number specified in the Count field.

When a count of 0 is specified, the logical-interrupt count is not modified; the disable-device-interrupt bit is the only bit that effects a change.

This command never requests an interrupt of the system master.

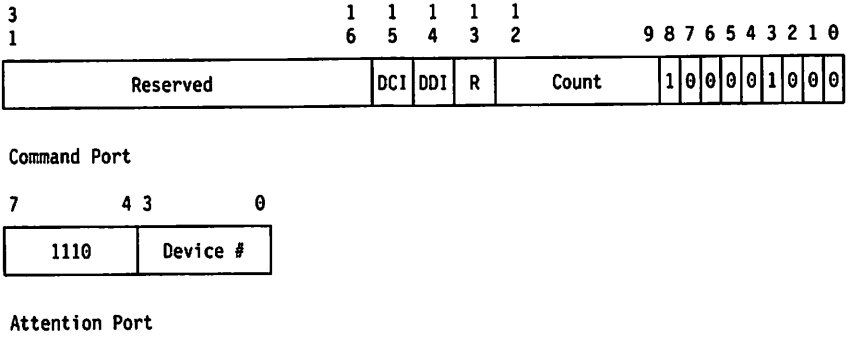


Figure 1-51. Reset Control Block Interrupt Command Structure

Bit 15 The disable-command-interrupt (DCI) bit is ignored by the device. An interrupt is never requested on completion of this command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the device specified. If this bit is set to 0, interrupts are enabled from the device specified.

Bits 12–9 The Count field is a 4-bit field that specifies the number of logical interrupts to be reset for the device specified. A value of 0 does not affect the logical-interrupt count.

When the count of logical interrupts becomes less than or equal to 0, the bit in the Device Interrupt Identifier port that is assigned to the device is set to 0. When all bits in the Device Interrupt Identifier port are set to 0 and the Interrupt Status port does not indicate an interrupt, the physical interrupt to the system master is reset.

When the Device Interrupt Identifier port is being used, a program can reset logical interrupts for control blocks singly or in multiples by using this command. The program does not have to wait for a physical interrupt in order to reset logical interrupts.

Programming Notes:

1. A device that is currently executing a control-block command must accept the Reset Control Block Interrupt command. This allows a program to reset interrupts that occur within a chain of control blocks without having to issue a Suspend command to the device.

2. When specifying the count value in this command, do not exceed the number of termination status blocks that have the interrupt-requested bits in the End Status Word 1 set to 1; otherwise, loss of physical interrupts can result.

   If the logical-interrupt count for a device becomes less than or equal to 0, the bit in the Device Interrupt Identifier port associated with the device is set to 0, and the subsystem does not generate a physical interrupt for logical interrupts from the device. While in this state, the logical interrupts continue to be posted in the associated termination status blocks, and the logical-interrupt count is increased. A logical-interrupt count greater than 0 causes the device to set the associated bit in the Device Interrupt Identifier port to 1 and to raise a physical interrupt if the interrupt is enabled.

3. When the Device Interrupt Identifier port is being used, the Reset Control Block Interrupt command is used to clear interrupts for control blocks. However, when a control block cannot be read or the termination status block cannot be stored, the device puts an interrupt into the Interrupt Status port. This interrupt must be cleared using the Reset Interrupt Status Port command.

4. Refer to "DIIP Interrupts (Multiple Form)" on page 3-26 for details on interrupt handling.

Reset Subsystem (Software Controlled)

The Reset Subsystem command has the same format as the Reset Device command, except that it must be issued to device 0.

The Reset Subsystem command can be used to perform a software-controlled reset (soft reset) of the subsystem and all attached devices. The option-2 bit specifies whether the subsystem and all attached devices are reset or whether the subsystem alone is reset.

When the command is initiated, all device activity for that subsystem is stopped, and the subsystem is reset to a known state.

After the command is completed, the Interrupt Status port is set to 0, and the interrupt-valid bit in the Command Busy/Status port is set to 1, indicating that a reset of the subsystem has been completed and an interrupt has been requested. The results of the Reset Subsystem command are:

- The enable-interrupts, DMA, and reset-reject bits in the Subsystem Control port are set to 0.
- Any previous physical interrupt to the system is cleared.
- All activity on the subsystem not related to resetting the subsystem ends.
- If the subsystem supports the Device Interrupt Identifier port, all bits in this port are set to 0.
- The Command and Attention ports are set to 0.
- When the option-2 bit is set 0, all devices attached to the subsystem perform the following actions:
  - All pending interrupts are cleared.
  - If the device uses the Device Interrupt Identifier port, the logical-interrupt count is set to 0.
  - Any suspended chain operation is purged.
  - The device interrupts are enabled.
  - The device is put into a state enabling execution of control-block commands.
  - The device performs any device-dependent reset activities.
- If the option-1 bit is set to 0 (indicating a complete reset), the subsystem and attached devices perform a power-on self-test and might require reinitialization.

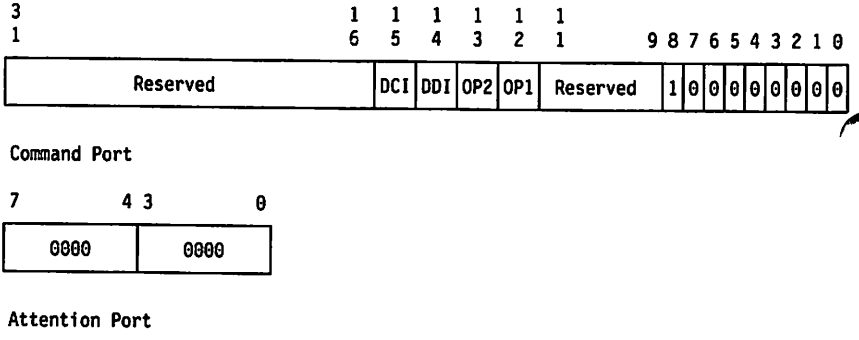


Figure 1-52. Reset Subsystem Command Structure

Bit 15 The disable-command-interrupt (DCI) bit is ignored by the device. An interrupt is always requested on completion of this command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the subsystem is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the subsystem. If this bit is set to 0, interrupts are enabled from the subsystem.

Bit 13 The option-2 (OP2) bit specifies whether the subsystem and all its attached devices are reset or whether the subsystem alone is reset. If this bit is set to 0, the subsystem and all its attached devices are reset. If this bit is set to 1, the subsystem is reset, but all attached devices are not reset.

Bit 12 The option-1 (OP1) bit specifies whether a partial or complete reset of the subsystem is performed. When this bit is set to 0, the subsystem is reset completely so that the subsystem must be reinitialized; the subsystem's internal memory is reset. If this bit is set to 1, the subsystem is partially reset and does not need to be reinitialized; the subsystem's internal memory is not reset.

Programming Notes:

This command differs from the hardware-controlled reset (hard reset) in that it does not cause a reset of the internal logic that controls the subsystem. Additionally, the soft reset depends on the availability of the Command and Attention ports for delivery, whereas the hardware-controlled Reset Subsystem command does not.

1. Programmable-option-select data is not reset by this command.
2. After a subsystem reset begins, the Command and Attention ports are not available until the reset has been completed.
3. The system software should use the software-controlled reset (soft reset) when it is unable to recover a device through the Reset Device command. The Reset Subsystem command must be used carefully because it ends all commands in progress on the subsystem. It also causes any interrupts queued by attached devices to be lost.
4. If the subsystem fails to complete a software-controlled reset within a specified time, the software in the system master should use a hardware-controlled reset.
5. The software-controlled reset normally is completed by setting the enable-DMA (DMA) and enable-interrupt bits in the Subsystem Control port to 0. These bits remain in that state unless they are changed after the reset starts. If the enable-interrupt bit is not changed, the subsystem will not generate an interrupt.
6. A subsystem reset causes the busy bit in the Command Busy/Status port to be set to 1 and causes the reject bit in the Command Busy/Status port to be set to 0. Commands submitted normally through the Attention port have the same effect on the busy and reject bits. Programs cannot use the busy and reject bits to distinguish between commands submitted through the Attention port and commands that cause a subsystem reset (which take longer). Therefore, to avoid resource conflicts, programs should maintain an external indication of any subsystem-reset activity.

Resume

Resume is an immediate command that puts the device that is specified in the Attention port into a state that enables it to execute all control-block commands. The Resume command is used to restart an active control-block chain that has been halted by a Suspend command. The device might not be able to restart if it is suspended because of internal conflicts or conditions. If the interrupt queue is full, the device remains in a suspended state until its queue is no longer full.

If the Resume command requests an interrupt on completion, the interrupt is requested before execution of a suspended control-block chain is resumed.

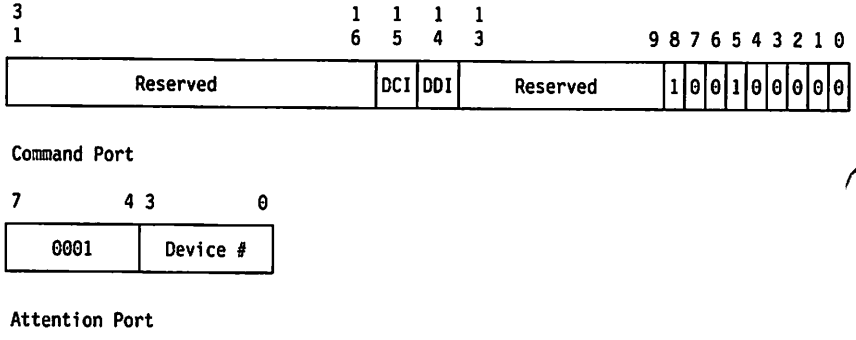


Figure 1-53. Resume Command Structure

Bit 15 The disable-command-interrupt (DCI) bit specifies whether the device requests an interrupt after successfully completing this command. If this bit is set to 1, the device does not request a system interrupt. If this bit is set to 0, the device requests an interrupt after completing the command and reports the status to the Interrupt Status port.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is enabled to generate interrupts to the system. If this bit is set to 1, interrupts are disabled for the device specified. If this bit is set to 0, interrupts are enabled for the device specified.

Run Immediate Diagnostic Test

The Run Immediate Diagnostic Test command is used to initiate a diagnostic test to the specified device. The results of the test are returned as an interrupt in the Interrupt Status port upon command completion. The interrupt code returned indicates whether the command was completed with or without errors. The command always requests an interrupt upon completion.

A more comprehensive diagnostic report can be obtained with the Run Diagnostic Test control-block command.

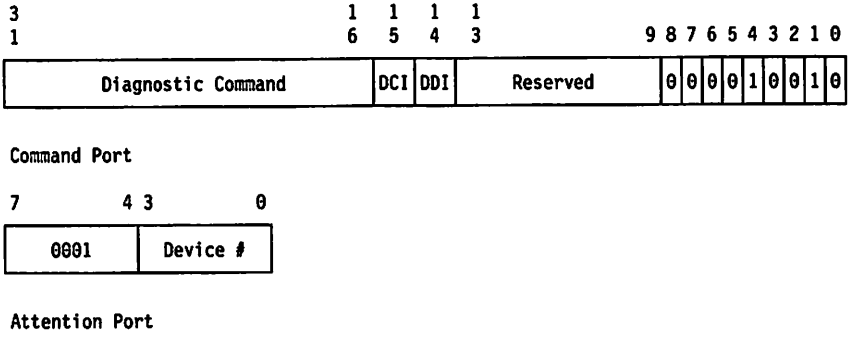


Figure 1-54. Run Immediate Diagnostic Test Command Structure

Bits 31 – 16 The Diagnostic Command field is a 16-bit field that indicates the diagnostic test to be performed. The code used is device dependent.

Bit 15 The disable-command-interrupt (DCI) bit is ignored by the device. An interrupt is always requested on completion of this command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the device specified. If this bit is set to 0, interrupts are enabled from the device specified.

Bits 13 – 9 These bits are reserved.

Suspend

The Suspend command is used to put the specified device in the Attention port into a state that no longer executes control-block commands. A Resume, Reset Device, or Reset Subsystem command is needed to restore the device to a state in which it can execute control-block commands again.

When the command is accepted by a device that is not executing a command, the internal state is set to show that the control-block execution is disabled.

When the command is accepted by a device that is executing a control block or a control-block chain, the execution of the Suspend command is deferred until the current control block or chain element has completed its execution. The completed command always stores the termination status block. The suspend bit in End Status Word 1 of the termination status block is set to 1 to indicate that the associated control block was the last one completed before the suspend request was completed. If this command is issued while a control block in a chain element, other than the ending chain element, is being run, the next element in the chain will be executed only after a subsequent Resume command is received.

If the Suspend command is issued while a control block that is not a chain element, or is the ending chain element, is being executed, the current control block will be completed, and all subsequent control blocks will be rejected until a Resume command is received.

When the Suspend command is completed, the device will reject any control-block commands directed to it until either a Resume, Reset Device, or Reset Subsystem command has been successfully executed. The device will accept any non-control-block command or direct hardware command directed to it.

A Suspend command always requests an interrupt upon completion. If a control block is being executed at the device, the interrupt will be a control-block interrupt. If a control block is not being executed, the interrupt returned will be an immediate command completed, without error.

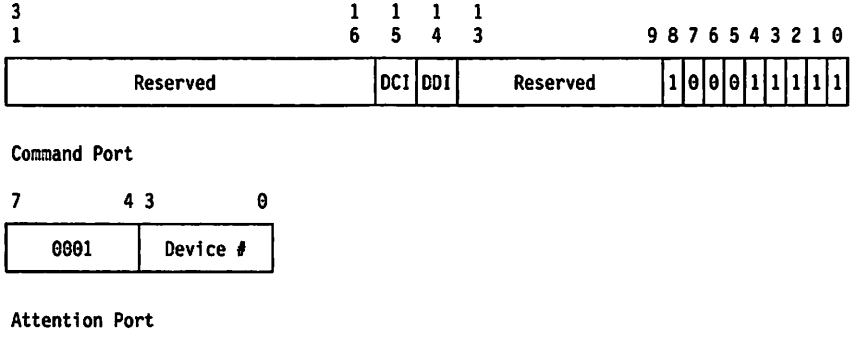


Figure 1-55. Suspend Command Structure

Bits 31 – 16 These bits are reserved.

Bit 15 The disable-command-interrupt (DCI) bit is ignored by the device. An interrupt is always requested on completion of this command.

Bit 14 The disable-device-interrupt (DDI) bit specifies whether the device is allowed to generate interrupts to the system. If this bit is set to 1, interrupts are disabled from the device specified. If this bit is set to 0, interrupts are enabled from the device specified.

Bits 13 – 9 These bits are reserved.

Programming Notes:

1. If a chain operation is suspended, the termination status block can be examined to determine the next control block to be executed when the chain is resumed. For more information, refer to "Signalling Protocol" on page 3-13.
2. A Suspend command is not necessary before a Reset Control Block Interrupt or Reset Interrupt Status Port command is issued to a device executing a control-block chain because these commands are always accepted by the device.
3. If a control block cannot be read, or the termination status block cannot be stored, and a Suspend command is issued, the program can determine that the Resume command will not be able to restart the chain.

4. The Suspend command always requests an interrupt upon completion. By examining the interrupt value returned as a result of a Suspend command, a program can determine the device state at the time the command was executed. If the device is not executing a control block when the Suspend command is accepted, the interrupt returned indicates that the immediate command was completed without error. If the device is executing a control block when the Suspend command is accepted, the command requests a control-block interrupt for the device. If the interrupt returned indicates that a hardware failure occurred while executing a control block, the program determines that the device was executing a control block and that the chain cannot be resumed. If the interrupt returned is not for a hardware failure, the program might have to examine the completion status of the termination status blocks associated with the device to determine whether the chain can be restarted.

Reset Subsystem (Hardware Controlled)

A program uses the hardware-controlled Reset Subsystem command to stop the activity of all devices attached to the subsystem and to restore the subsystem and attached devices to a known state. The state of each device after the reset is defined in the description of the Reset Device command.

This command is initiated by the reset bit in the Subsystem Control port, which controls a reset signal internal to the subsystem. If the subsystem and its attached devices require an initial program load, they are put into a state in which they can accept an Initialize Device command to receive their program loads. Upon completion, the command stores an interrupt in the Interrupt Status port. POS data is not reset by this command. If the Device Interrupt Identifier ports are supported, these ports are set to 0.

A Reset Subsystem command is originated by the program. This is done by writing a 1 to the reset (RST) bit of the Subsystem Control port, followed by writing a 0 value to the reset bit in an implementation-defined amount of time. In other words, the reset bit is switched from 0 to 1 to 0.

Setting the reset-control bit of the Subsystem Control port to 1 clears the physical interrupt line from the subsystem and causes all device

activities on the subsystem to abend. The following values are set in the Command Busy/Status port:

- Busy (B) = 1
- Interrupt valid (IV) = 0
- Reject (REJ) = 0.

The enable-subsystem interrupt (EI), enable-DMA (DMA), and reset-reject (RR) control bits in the Subsystem Control port are set to 0 by the subsystem. EI and DMA remain 0 unless specifically set by the program while the device-reset operation is in progress. All activities not related to subsystem reset are ended.

When the subsystem senses the transition of the reset-control bit from 1 to 0, it completes reset actions for the subsystem and all attached devices.

At the completion of this command, the Interrupt Status port and all bits in the Command and Attention ports are set to 0, and all device interrupts are cleared.

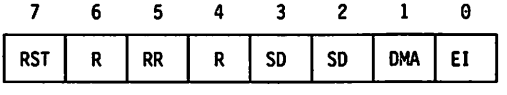


Figure 1-56. Subsystem Control Port (Hardware-Controlled Subsystem Reset)

Programming Notes:

1. Using the hardware-controlled Subsystem Reset command ends all other commands in progress on the subsystem and causes all interrupts from attached devices to be lost.
2. This command should be used in error-recovery procedures by programs in the system master when all other recommended retry actions have failed.
3. If the subsystem fails to complete a Reset Subsystem command within the maximum time defined for it by each implementation, the software unit may have reset the reset bit in the Subsystem Control port to 0 too rapidly. In this case, setting the reset bit to 1 again will restart the Reset Subsystem command sequence.

4. Notice that the normal completion of this command sets the Subsystem Control port DMA and EI control bits to 0. These control bits should be set to 1 by the program in the system master after it has switched the reset-control bit to 0 or after the busy bit in the Command Busy Status port is set to 0. If these control bits are not reset, DMA and interrupts for the subsystem remain disabled after the Reset Subsystem command has been completed.

5. A Subsystem Reset command causes the busy bit in the Command Busy/Status port to be set to 1 and the reject bit in the Command Busy/Status port to be set to 0. Commands submitted normally through the Attention port have the same effect on the busy and reject bits. By examining these bits, programs cannot distinguish between commands submitted through the Attention port and commands that cause a subsystem reset (which take longer). Therefore, to avoid resource conflicts, programs should maintain an external indication of any subsystem-reset activity.

Chapter 2. Physical Level

The physical level provides support for gaining access to data and control areas in shared memory and I/O address space and for interrupting the system master.

At the physical level, the architecture defines the control areas in I/O address space that are used to deliver immediate commands and command control blocks to the various devices in a subsystem. It also defines how devices gain access to data and command control blocks in shared memory, as well as how a device or subsystem interrupts a system master.

Structure

The following figure shows the structure of the physical level in terms of the components used to provide the command and command-control-block delivery.

© Copyright IBM Corp. 1991

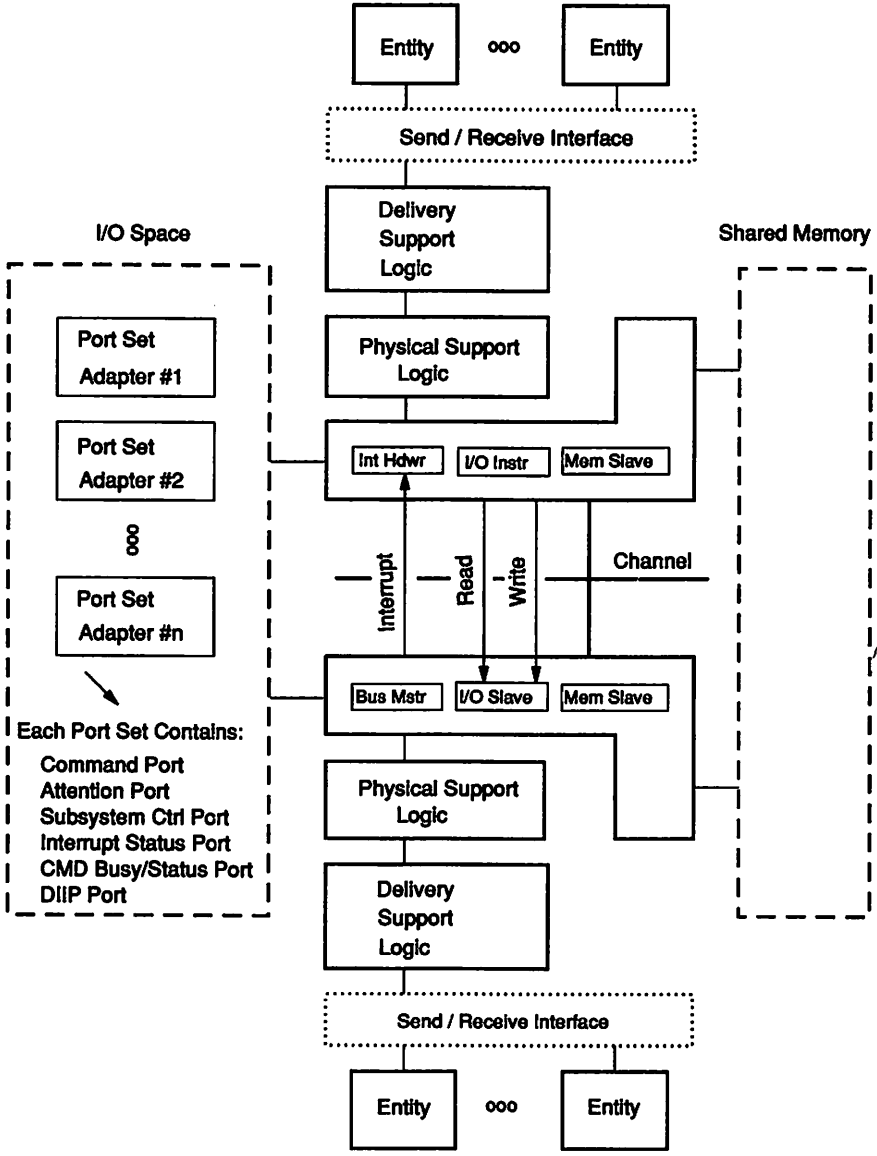


Figure 2-1. Locate Mode Physical Level

Figure 2-1 shows the physical-level structure for a configuration with a single system master and a single subsystem. The physical-level

services are provided by a combination of hardware support, physical-support logic, and control areas in I/O address space.

System-Master Hardware Support

The hardware support in a system master includes an interrupt controller, the instructions used to gain access to information in I/O address space, and shared-memory-slave support.

The interrupt controller supports the use of interrupts as a means of signalling the system master from one or more subsystems.

The I/O instructions provide the system master with access to information in control areas in I/O address space that are shared with the subsystem.

The memory-slave support provides access to data and control information either by bus masters or by memory instructions.

Subsystem Hardware Support

The hardware support in a subsystem includes I/O-slave support and bus-master support.

The I/O-slave support provides a means for the logic in a system master to initialize the subsystem, enable access to the subsystem, signal the subsystem, and identify the subsystem as the source of the system interrupts.

The bus-master support provides access to data and control information in shared memory. The subsystem interface to the system is through the following ports:

- Command port
- Attention port
- Subsystem Control port
- Interrupt Status port
- Command Busy/Status port
- Device Interrupt Identifier port (optional).

Support Logic

The physical-support logic implements the services, protocols, and functions that support the sharing of control areas by different command and command-control-block delivery flows. It also provides the support necessary for access to the shared memory used by a subsystem for command control blocks and data in data buffers.

Memory Address Space

One of the most important aspects of the physical level is that it provides access to data and control information in shared memory. The delivery-level protocols need access to command control blocks, which are physically located in system-master memory that is shared with a subsystem.

The locations of the data buffers, used during data transfer, are also assigned locations in shared memory. The locations of the data buffers are determined by the delivery-level support and can change from command control block to command control block.

The command control blocks are assigned locations within shared memory. The starting location of the command control block is provided by the command-delivery support for each delivery request.

The shared-memory addresses that are supported are defined during system configuration.

I/O Address Space

The physical-level services provide access to a set of control areas (ports) in I/O address space that are shared by the command and command-control-block delivery for all the entities in a system or subsystem.

Note: This minimizes the amount of I/O address space required, but it requires the management of the sharing of this address space.

The range of I/O addresses assigned to each subsystem is determined during system configuration. A description of the control

areas is defined by the architecture for the Locate mode and can be found in "I/O Address Space" on page 1-15.

Physical-Level Services

The services provided by the physical level are used to gain access to data and control information and to support interrupting a system master from any subsystem on the Channel. The services provided are:

- Push
- Pull
- Interrupt.

These services are used by the delivery-level logic in both the system master and subsystems to gain access to control areas in I/O address space, to gain access to command control blocks and data buffers in shared memory, and to cause an interrupt of the system master.

Note: A push from a system master to a control area in the I/O address space of the Attention port causes an interrupt of the associated subsystem upon completion.

Push and Pull

The delivery-support logic uses the I/O-address-space form to move control parameters between the system master and the subsystem. It uses the shared-memory form to gain access to command control blocks and data buffers.

A system master uses I/O instructions (IN or OUT) to gain access to control areas in I/O address space.

A subsystem uses bus-master, DMA-type operations to gain access to the shared memory for the specific subsystem or entity that performs the service.

The push service moves data or control information from a location in the caller's shared memory to a location in shared memory or I/O address space.

The following conceptual service primitive illustrates the parameters required when the push service is invoked:

```
PUSH(unit id, addr space, from addr, to addr, length, return code)
```

The ***unit ID*** parameter identifies the system master or subsystem that applies.

The ***address space*** parameter indicates whether the ***to address*** parameter refers to a location in shared memory or in I/O address space.

The ***from address*** parameter identifies the location of the information to be pushed into the location that is indicated by the ***to address*** parameter.

The ***to address*** parameter identifies the destination in I/O address space or shared memory of the information that is indicated by the ***from address*** parameter.

The ***length*** parameter indicates the number of bytes to be pushed into the location in I/O or shared memory, identified by the ***to address*** parameter.

The ***return code*** parameter indicates the success or failure of the push operation.

The pull service moves information from a location in I/O address space or shared memory to a location in the caller's shared memory.

The following conceptual service primitive illustrates the parameters required when the pull service is invoked:

```
PULL(unit, addr space, from addr, to addr, length, return code)
```

The ***unit ID*** parameter identifies the system master or subsystem that applies.

The ***address space*** parameter indicates whether the ***to address*** parameter refers to a location in shared memory or in I/O address space.

The ***from address*** parameter identifies the location of the information to be pulled into the location that is indicated by the ***to address*** parameter.

The ***to address*** parameter identifies the destination in the caller's shared memory of the information that is indicated by the ***from address*** parameter.

The ***length*** parameter indicates the number of bytes to be pulled into the location that is identified by the ***to address*** parameter.

The ***return code*** parameter indicates the success or failure of the pull operation.

Interrupt

In addition to the push and pull services, the physical level also provides a service to interrupt the system master.

Note: As stated previously, there are command-delivery protocols that can be used to cause an interrupt to a subsystem. This is accomplished by using the push service primitive to put the appropriate control information into the control area in I/O address space that is assigned to the subsystem Attention port, not by using the interrupt service.

The following conceptual service primitive illustrates the parameters required when the service is invoked:

```
INTERRUPT (return code)
```

There are no parameters required by the interrupt service primitive; however, the interrupt service does have a return code.

The ***return code*** parameter indicates the success or failure of the interrupt operation.

Data Delivery

In subsystems, the bus-master support can also be used to transfer data to and from entities in the subsystem. The command control blocks contain the location of the data in shared memory to which the subsystem entities have access. The push and pull services can be used by these entities as well as by the delivery-support logic to gain access to the data in shared memory.

Note: Different forms of address transformation might be required to establish addressability between a system master or subsystem's local shared memory and the shared memory in the command control blocks. The specific address transformation is determined by the particular system master or subsystem implementation.

Physical-Level Protocols

The protocols defined at the physical level for access to control areas in I/O address space are the same as those required for Channel operations to an I/O slave.

The protocols defined for access to command control blocks and data buffers in shared memory are the same as those required for Channel operations to a shared-memory slave.

The protocol defined for interrupting a system master from a subsystem is the same as the Channel protocol for interrupting a system master.

The protocols at the delivery level for command and command-control-block delivery define the uses of the various control areas. See Chapter 3, "Delivery Level" on page 3-1 for additional information.

Notes:

Chapter 3. Delivery Level

The delivery level provides the set of services and protocols that support the delivery of requests from programs in a system master to devices in a subsystem. The primary form for these requests is one or more control blocks that have been grouped together by addresses. This grouping of control blocks includes the command control block, a status block, and an optional indirect list. When the request consists of more than one control block, the next one is pointed to by a chain address maintained for that purpose in each command control block.

The delivery level also supports the delivery of immediate and hardware-controlled commands to a subsystem. The immediate commands are limited to control-type functions that occupy one doubleword. The hardware-controlled commands are used to directly control the basic operation of the subsystem.

The Locate mode architecture defines the delivery-level protocols that are used with bus-master type subsystems designed to operate as shared-logic control units. It also defines the functions and identifies the support logic required in both the system master and the subsystem to implement the services and protocols of the delivery level.

Structure

The following describes the delivery-level structure and the services that support command and control-block delivery between entities.

© Copyright IBM Corp. 1991

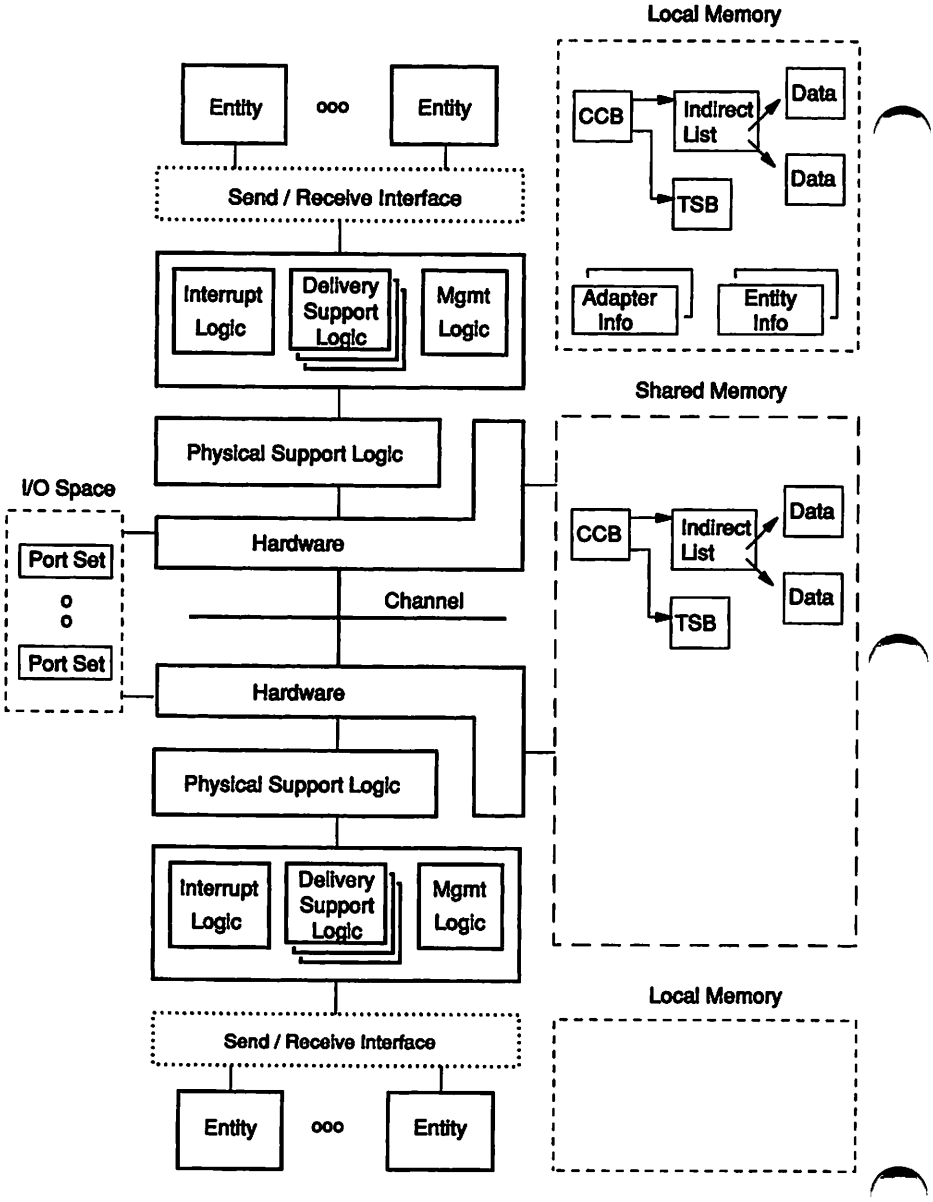


Figure 3-1. Overall Delivery-Level Structure

Figure 3-1 shows that the delivery support in the system master has multiple instances of the request-delivery logic and the local control

areas needed to maintain the states of both multiple Locate mode subsystems as well as multiple devices associated with each of them.

The figure shows that in a system master, there is typically a single instance of the interrupt handler that supports the completion of the request-delivery protocol. It also shows that in a subsystem, there is a single instance of both the request-delivery support and the system-interrupt support. These must be managed to allow them to support the multiple request-delivery protocols.

Figure 3-1 on page 3-2 also shows that delivery support is provided to programs by the send and receive interfaces and that the delivery logic uses the service of the underlying physical level.

The send and receive interfaces provide the local operating-environment policy support for interfaces to the delivery-level support and for scheduling this support on the processing-level threads.

Processing-level threads of execution are used in a multitasking programming environment.

Figure 3-1 on page 3-2 shows two views of a sample delivery request (set of coupled command control blocks). The request control-block structure needs to be understood in the local address space seen by the system-master entity and the request delivery-level support, as well as in the shared memory understood by the subsystem entities and delivery-level support.

This happens when a program in a system master is running in virtual or protected mode. The program in the system master would use logical addresses when building control blocks or referring to fields within the command control blocks. On the other hand, the device in the subsystem would be using physical addresses when fetching, storing, or referring to fields within the command control blocks.

From the delivery-support view, all addresses must be physical addresses in shared memory. The programs in the system master can provide a local view by prefixing each control block with the necessary logical addresses.

Because the request delivery-level protocols use the control areas in the I/O space to deliver requests and to process any related interrupts, these are also shown as part of the overall structure.

Locate mode commands can be delivered by one of two methods:

- The hardware-controlled method involves writing to the Subsystem Control port, which causes the subsystem to react immediately to the functions encoded in the port bits. For the definition of these functions, see "Subsystem Control Port" on page 1-20.
- The software-controlled method involves writing to the Command and Attention ports and waiting for the command handler in the subsystem to accept or reject the command. For the definition of these functions see "Command Port" on page 1-18 and "Attention Port" on page 1-18.

Resetting the subsystem always forces the subsystem to a known state, regardless of the state of the subsystem and its attached devices. The specific actions for a reset of the subsystem are described in "Reset Subsystem (Hardware Controlled)" on page 1-90.

Immediate-Command Delivery

Immediate commands are mainly device directed and are control oriented. Similar to control-block commands, immediate commands use the Command and Attention ports for delivery.

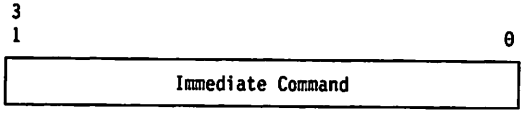


Figure 3-2. Command Port — Immediate Commands

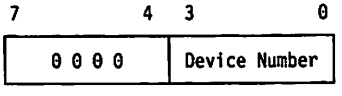
The 32-bit immediate command is put into the Command port.

In addition to the Command port, the immediate-command delivery requires the use of the Attention port.

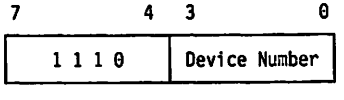
An immediate command that is written to the Command and Attention ports cannot be executed until the device has verified that the whole

command has been accepted by the subsystem. After acceptance has been verified, the busy bit in the Command Busy/Status port is set to 0. The command type is determined by examining the attention code in the Attention port. When the command is determined to be an immediate command, the operation code in the Command port is decoded to determine the device actions to be performed. The following figure shows the format of the Attention port when it is used for immediate-command delivery.

Reset Device/Reset Subsystem



Reset Control Block Interrupt/Reset Interrupt Status Port



All Other Immediate Commands

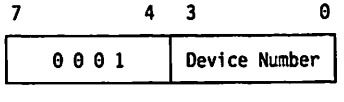


Figure 3-3. 8-Bit Attention Port—Immediate Commands

Bits 3—0 For a description of the Device Number subfield (bits 3-0), refer to the definition of the Attention port in "Attention Port" on page 1-18.

Control-Block Command Delivery

Control-block commands are mainly data-transfer oriented and use the Command and Attention ports for delivery.

Control-block commands are identified by examining the attention code in the Attention port. For a control-block command, the control-block starting address is contained in the Command port.

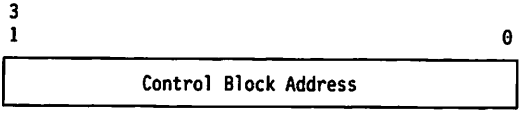


Figure 3-4. Command Port—Control-Block Commands

The address of a control block placed in the Command port must be the 32-bit, doubleword-aligned physical address of the location in shared memory where the first or only command control block is located.

In addition to the Command port, the command-control-block delivery requires the use of the Attention port. The following figure shows the format of the Attention port when it is used for command-control-block delivery.

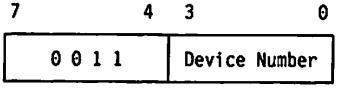


Figure 3-5. 8-Bit Attention Port—Control-Block Commands

Bits 3—0 For a description of the Device Number subfield (bits 3-0), refer to the definition of the Attention port in "Attention Port" on page 1-18.

Delivery-Level Services

The services provided by the delivery level are used to deliver requests from an entity (software) in a system master to an entity (device) in a subsystem. Three types of service are provided:

- Immediate-command delivery
- Hardware-controlled command delivery
- Command-control-block delivery.

A description of each of these services is provided in the following sections.

Immediate-Command Delivery Service

The immediate-command delivery service is used to pass an immediate command to a device or subsystem, using the services provided by the underlying physical level for access to the appropriate control areas in I/O address space.
The following conceptual service primitive illustrates the parameters required when invoking the immediate-command delivery service:

```
DELIVER_IMMEDIATE (destination, attention code, command, return code)
```

The ***destination*** parameter identifies the adapter (subsystem) and entity (device) to which the immediate command is to be delivered.

The ***attention code*** parameter contains the value to be placed in the Attention port on command delivery.

The ***command*** parameter contains the immediate command to be delivered.

The ***return code*** parameter is returned to the caller and indicates the successful or unsuccessful acceptance of the command.

Hardware-Controlled Command Delivery Service

The hardware-controlled command delivery service is used to set the appropriate hardware-control bit in the Subsystem Control port associated with a feature adapter and to use the services provided by the underlying physical level to set up and access the appropriate control areas in I/O space. The following conceptual service primitive illustrates the parameters required when invoking the command delivery service:

```
DELIVER_CONTROL (destination, value, return code)
```

The ***destination*** parameter identifies which adapter (subsystem) the hardware-controlled command is directed to.

The ***value*** parameter contains the hardware-control value to be delivered.

The first byte of the ***value*** parameter is used to identify the specific hardware-controlled command. It must be one of the following:

- Reset Subsystem
- Reset Reject Condition
- Set Subsystem Interrupt Enablement
- Set DMA Enablement
- Set Other Bits in the Subsystem Control Port.

When Set Other Bits is indicated, byte 2 of the ***value*** parameter contains the value to be set in the Subsystem Control port.

The ***return code*** parameter is returned to the caller and indicates the successful or unsuccessful acceptance of the command.

Control-Block Delivery Service

The control-block delivery service is used to pass the address of a chained or nonchained control block to an entity (device) in an adapter (subsystem), using the services provided by the underlying physical level to set up and access the appropriate control areas (ports) in I/O address space. The following conceptual service primitive illustrates the parameters required when invoking the control-block delivery service:

```
DELIVER_BLOCK (destination, location, return code)
```

The ***destination*** parameter identifies the adapter and entity to which the address of the chained or nonchained control block is to be delivered.

The ***location*** parameter contains the address in shared memory of the chained or nonchained control block to be delivered.

The ***return code*** parameter is returned to the caller and indicates the successful or unsuccessful acceptance of the command.

Delivery-Level Protocols

There are two types of delivery-level protocols: command-delivery protocols and signalling protocols.

Command-Delivery Protocol

The following figure shows the flow of commands through the Command and Attention ports. It is followed by a description of the protocols used to deliver these commands.

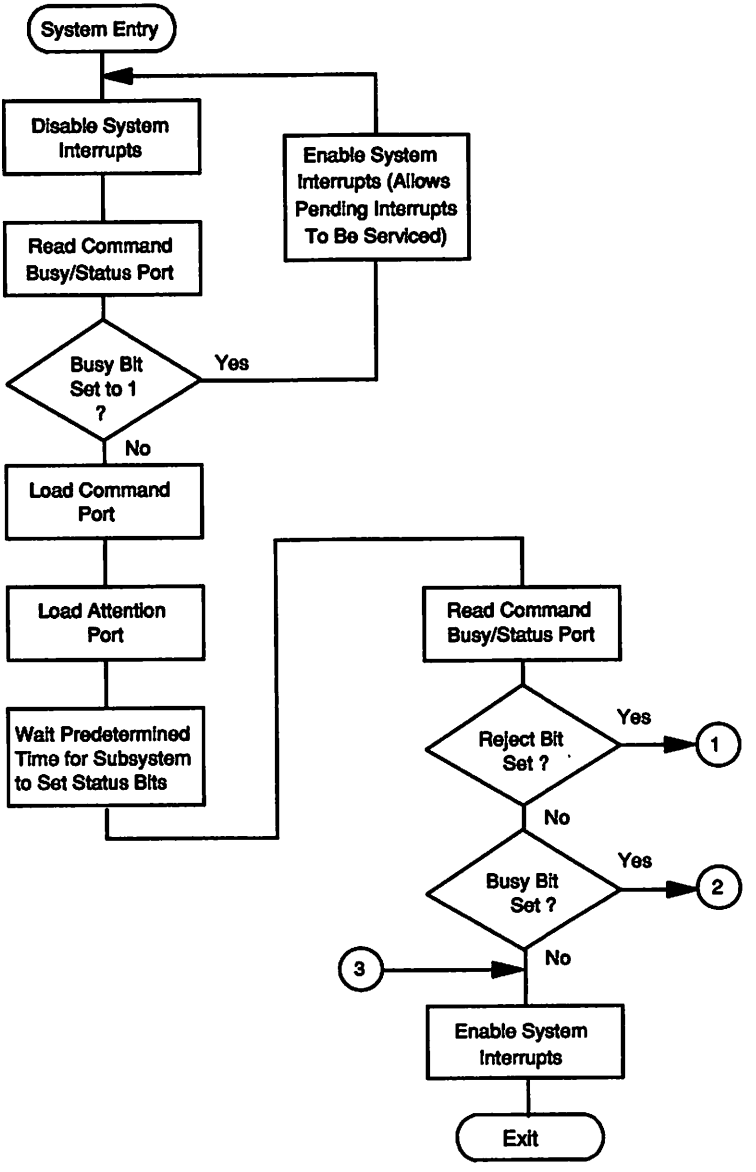


Figure 3-6 (Part 1 of 2). Command-Delivery Flow

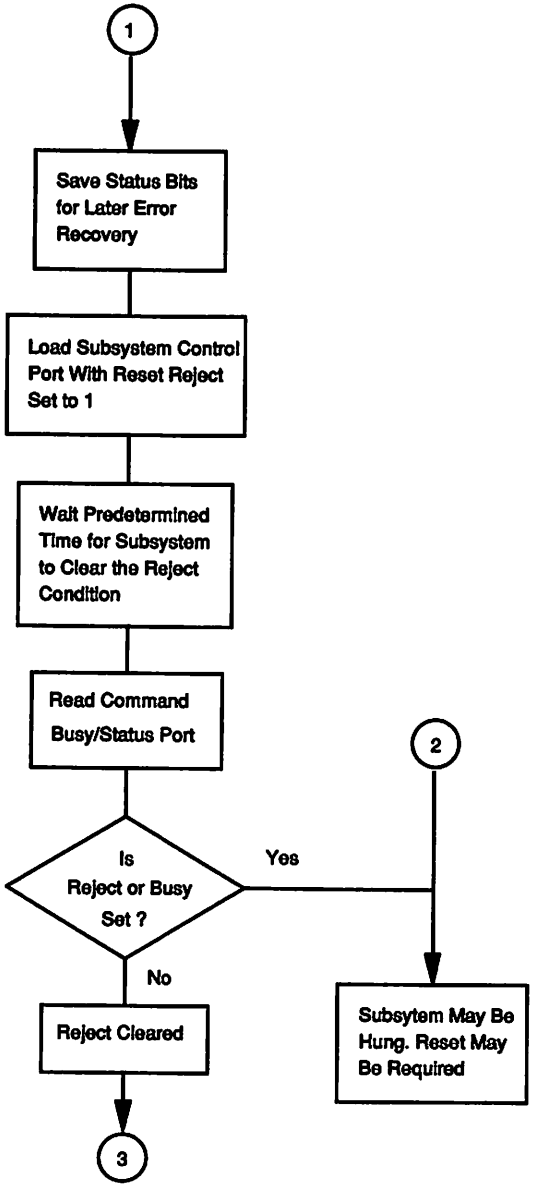


Figure 3-6 (Part 2 of 2). Command-Delivery Flow

System interrupts should be disabled before attempting to write a command to the subsystem. The Command Busy/Status port should be read to ensure that the busy bit is off, indicating that the Command

and Attention ports are available. The desired command (or address, in the case of a control block command) must be written to the Command port before the Attention port is written to with the desired attention code and device number. After the two ports are written, the Command Busy/Status port must be read to ensure that the command was not rejected by the subsystem.

Note: The amount of time that a subsystem requires to accept or reject a command is implementation dependent and can be determined in advance by examining the minimum command-status time in the configuration data, which is returned in response to a Read Configuration command.

If the busy and reject bits are both set to 0, the program can assume that the command has been accepted by the subsystem. System interrupts can then be enabled, and the system is free to perform other tasks.

If the reject bit is set to 1 in the Command Busy/Status port, it is an indication that the subsystem has rejected the command. The program should examine the status bits in the Command Busy/Status port to determine the reason for rejection and take appropriate action to resolve the condition.

If the reject bit is 1, it must be explicitly reset by the system by setting the reset-reject bit to 1 in the Subsystem Control port. To ensure that the reject condition has been cleared, the program can read the Command Busy/Status port to ensure that the reject and busy bits have been set to 0 by the subsystem. If the condition has not been cleared, the system might need to use more severe error recovery actions; that is, a Reset Device or Reset Subsystem command could be required.

Signalling Protocol

This section describes the protocols governing interrupts in the Locate mode, how interrupts are presented to the system master, and how they can be cleared.

Interrupts are used as the signalling mechanism between the subsystem and the system master in the Locate mode. They indicate that the subsystem, or a device attached to the subsystem, has

completed a command and requires the system master to act upon the completion by servicing the interrupt request.

For simplicity in the discussion of interrupts, the term "device interrupt" is used to refer to an interrupt that either originates from a device attached to the subsystem, or the subsystem itself when a command is directed to it as device 0, or when the reset-subsystem (RST) bit in the Subsystem Control port is toggled.

Interrupt Identification

In the Locate mode, a subsystem has a control area (the Interrupt Status port) in I/O address space that is set by the subsystem with the identification of the device that caused the interrupt, as well as the general reason for the interrupt. Programs in the system master must read this information to determine the source and cause of the interrupt.

Each control-block command also has a unique area (the termination status block) in shared memory where interrupt information and completion status can be stored by the subsystem. Programs in the system master can read this area to determine how a particular command control block has been completed.

Physical and Logical Interrupts

In the Locate mode, the architecture defines and uses two types of interrupts:

- Physical
- Logical.

A physical interrupt occurs when a subsystem activates the interrupt line to the system master, alerting it that an I/O event has occurred during the execution of a command issued to a device on the subsystem. This I/O event is associated with the successful or unsuccessful completion of a command control block, an immediate command, or a hardware-controlled command. When the system master recognizes the activated line, it typically preempts the program currently being executed and directs an interrupt-handling program to the I/O event. Physical interrupts are directly associated with immediate commands and direct hardware commands, and they can be signalled as a result of command control blocks.

Logical interrupts are associated only with command control blocks. A logical interrupt occurs when a completed command control block causes End Status Word 1 of the termination status block to be written into the shared memory with the control-block-interrupt-requested bit set to 1 and the device's bit set to 1 in the Device Interrupt Identifier port. This logically informs the system master of the interrupt, potentially before the physical interrupt can be delivered to the system master. An option of the Locate mode architecture allows a program in the system master to process logical interrupts for the command control block before the physical interrupt is presented. Logical interrupts are reset by the Reset Control Block Interrupt command. This is described in detail in "Resetting Interrupts" on page 3-38.

Physical-Interrupt Enablement

Physical-interrupt enablement determines whether an interrupt that has been detected by a device will be presented as a physical interrupt to the system master by the subsystem. In the Locate mode, interrupts can be disabled at the subsystem or device level.

When interrupts are disabled at the subsystem level, the subsystem and its attached devices cannot present a physical interrupt to the system master until the subsystem is enabled for interrupts.

The subsystem can be disabled for interrupts by setting the enable-interrupt (EI) bit in the Subsystem Control port to 0.

An individual device can be disabled for physical interrupts. A device is disabled for interrupts by setting the disable-device-interrupt (DDI) bit to 1 in any immediate command directed to a device. (It is enabled by setting the disable-device-interrupt bit in an immediate command to 0.)

When disabled, a device cannot present any further interrupt requests as physical interrupts to the system master. An interrupt request that is generated by a device after the device is disabled must be saved by the device for later presentation to the system master.

Once an interrupt value is written into the Interrupt Status port, it will cause an interrupt to the system master if the subsystem is enabled, even if the device that was the source of the interrupt has been subsequently disabled.

Interrupt Queuing within the Subsystem

A subsystem has a single Interrupt Status port in I/O address space for communicating interrupt data to the system master. Once an interrupt value is written into the Interrupt Status port, the port cannot be used to denote any other interrupt until it is explicitly cleared by a Reset Interrupt Status Port command issued from the system master. Because there can be multiple devices contending for this resource (and the Interrupt Status port can be used by only one device at a time), the subsystem is responsible for ensuring that each attached device is given an opportunity to send a physical interrupt request to the system master.

A device can detect an interrupt condition and be unable to present it to the system master at the time of detection. This occurs when the Interrupt Status port is currently in use, or when the device has been disabled for interrupts. To deal with this situation, each device must retain sufficient information to allow later presentation of the interrupt. An interrupt should never be lost in this case.

Interrupt Types

The Locate mode architecture defines two forms of interrupt handling:

- Non-DIIP or simple form, which serializes all interrupts through a single Interrupt Status port
- DIIP or multiple form, which uses the Interrupt Status port for non-control-block and hardware-failure interrupts but uses the Device Interrupt Identifier port (DIIP) for control-block interrupts.

Non-DIIP Interrupts (Simple Form)

In this form of interrupt architecture, all physical interrupts are presented to the system master by the subsystem recording device-level information in a single Interrupt Status port in I/O address space. Once the Interrupt Status port is written by the subsystem with a valid interrupt value, the Interrupt Status port is unavailable for a new interrupt cause until software in the system master explicitly clears it. Clearing the Interrupt Status port is usually accomplished by the Reset Interrupt Status Port command.

An interrupt value in the Interrupt Status port contains two fields:

- The device ID that caused the interrupt
- The reason the interrupt was sent (the Interrupt Identifier Code field).

  The Interrupt Identifier Code field is set by the subsystem to indicate whether a command control block, immediate command, or hardware-controlled command was successfully completed.

In the Locate mode, all interrupts are presented by a device in the same order in which the system master submitted the corresponding command. This restriction does not apply to the Move mode. If commands must be executed in another order, the Move mode must be used.

Note: This can be done in the implementation if the subsystem queues all interrupt requests in the order in which the device raised them.

The Locate mode requires that commands to a device be executed in the order in which they were submitted from the system master. This restriction does not apply to the Move mode. If commands must be executed in another order, the Move mode must be used.

Device – Control-Block Interrupts: This section describes the actions taken by a device when a command control block is completed and needs to raise an interrupt to the system master.

The codes stored in the Interrupt Status port for command control-block interrupts are:

- Command control block completed, no error
- Command control block completed, error
- Command control block completed, hardware failure.

The interrupt-valid (IV) bit in the Command Busy/Status port is set to 1 when the Interrupt Status port is written with the interrupt code.

Normally, when a command control block is completed and requires an interrupt, the termination status block associated with the command control block has at least End Status Word 1 stored with the interrupt-requested (INT) bit set to 1 before the physical interrupt is requested.

A program can elect to inhibit the storing of all termination-status-block status in the non-DIIP-interrupt case when it is completed without error. When this option is selected, a physical interrupt request is enqueued for the device without any corresponding logical interrupt. This option is offered to improve system performance. It should be used with care, especially with control-block chains, because it is more difficult for a program in the system master to determine the completion status of a particular control block. (If a program were written so that only the last control block in a chain requested an interrupt on normal completion, no problems would occur.)

The inhibiting of the storing of termination-status-block status is ignored and a logical interrupt is raised if the command control block is completed with any of the following conditions:

- The command control block was completed with an error that was not suppressed. An example of an error that cannot be suppressed is an irrecoverable hardware failure during a read.
- The command control block is suspended as the result of a Suspend command.

These conditions cause a logical interrupt to be raised and the storing of at least End Status Word 1 with the interrupt-request (INT) bit set to 1 in the termination status block.

Note: If termination-status-block status is stored, a program running in the system master can observe that a device has completed a command control block with the interrupt-request (INT) bit set before the actual interrupt is requested. This is possible if the program tests the termination status block associated with the completed command control block before the interrupt is received by the system master.

This technique is not recommended, however, because an interrupt cannot be cleared in the non-DIIP-interrupt case unless the subsystem has placed the interrupt value in the Interrupt Status port.

The correct method of detecting a pending interrupt is to check the interrupt-valid (IV) bit in the Command Busy/Status port. When this bit is equal to 1, the Interrupt Status port can be read to determine the interrupt type and device origin. Any interrupt present in the Interrupt Status port is cleared by the Reset Interrupt Status Port command.

A command control block can also fail as a result of a hardware failure when the subsystem or device is attempting either to read a control block or to store status information in a termination status block. In either of these cases, termination-status-block status is not stored, and an interrupt code of hex 8 (Hardware failure – control block command) is presented in the Interrupt Status port. The execution of any control-block chain for the device is terminated when this error is encountered.

Device – Non-Control-Block Interrupts: The codes for non-control-block interrupts are:

- Immediate command completed, no error
- Immediate command completed, error
- Immediate command completed, hardware failure
- Reset subsystem or device completed, no error.

The interrupt-valid (IV) bit in the Command Busy/Status port is set to 1 when the Interrupt Status port is written.

Subsystem – Physical Interrupts: When a subsystem determines that a physical interrupt must be raised to the system master, the subsystem must ensure that the following actions are taken:

1. The Interrupt Status port must be written with a valid interrupt identifier and device number. The device must be enabled to interrupt the system master.
2. The interrupt-valid bit in the Command Busy/Status port must be set to 1 to indicate that the Interrupt Status port contains a valid interrupt. The subsystem must be enabled to interrupt the system master before the interrupt-valid bit can be set to 1.
3. The subsystem must activate the physical-interrupt-request line for the level it was assigned at initialization time.

After the system master has serviced the interrupt, it must reset the interrupt by performing the following actions:

1. The Interrupt Status port must be reset by issuing a Reset Interrupt Status Port command. This causes the subsystem to clear the Interrupt Status port and reset the interrupt-valid bit in the Command Busy/Status port. This, in turn, causes the physical-interrupt-request line to become inactive.

2. The system master must reset the interrupt controller (if it has one) by issuing an End of Interrupt command.

   The subsystem can now raise a new physical interrupt to the system if any interrupts are pending within the subsystem.

System Master – Physical Interrupts: The following discussion deals with programming considerations in the system master with the non-DIIP-interrupt-handling model of Locate mode architecture.

An interrupt is sent from the subsystem to the system master if the subsystem is enabled to send interrupts to the system master and the Interrupt Status port has an interrupt value in it.

When the system master receives a physical interrupt on an interrupt level assigned to the subsystem, it should invoke a program to read the interrupt status of the subsystem to determine whether there is an interrupt to service. This program is called a *first-level interrupt-handling routine* for the subsystem. It is invoked in a state in which the system master is disabled for interrupts.

The first-level interrupt-handling routine determines the interrupt state of a subsystem by reading its Command Busy/Status port in I/O address space to determine whether the interrupt-valid (IV) bit is equal to 1. If this is the case, the subsystem or one of its attached devices has requested a physical interrupt. If the IV bit is 0, the first-level interrupt-handling routine can return with the report that it did not have an interrupt to service.

If the IV bit is set to 1, the first-level interrupt-handling routine must read the Interrupt Status port for the subsystem to obtain the identity of the device presenting the interrupt and the reason. The Device ID field identifies the device, and the Interrupt ID field indicates the type and reason for the interrupt.

The interrupt should be cleared by the first-level interrupt-handling routine as soon as possible to avoid blocking the interrupt level and to free the subsystem to present interrupts from other devices that can be attached. The interrupt is cleared by the Reset Interrupt Status Port command. The Attention port value specified in the Reset Interrupt Status Port command must have a device ID equal to the device ID read from the Interrupt Status port at the time of the interrupt. In some cases, the clearing action must be done before

analysis of the cause of the interrupt is completed. This can be true when dealing with chained command control blocks in a multitasking environment. In this case, the clearing operation is done in the first-level interrupt-handling routine, and a *second-level interrupt-handling routine* is scheduled. The second-level interrupt-handling routine is like the program that issued the command chain. To be most effective, the second-level interrupt-handling routine should be waiting for a signal to continue processing. The first-level interrupt-handling routine provides the needed signal, and it unblocks execution of the second-level interrupt-handling routine. Having freed the subsystem and the interrupt level, the first-level interrupt-handling routine returns control to its caller.

When the second-level interrupt-handling routine is unblocked and signalled, it needs input arguments. Most likely, the argument supplied to a second-level interrupt-handling routine is the value read by the first-level interrupt-handling routine from the Interrupt Status port.

When a second-level interrupt-handling routine approach is used, the first-level interrupt-handling routine can prevent reentry into the second-level interrupt-handling routine for interrupts by disabling the device that caused the interrupt. This is done by setting the disable-device-interrupt (DDI) bit to 1 before issuing the Reset Interrupt Status Port command. The second-level interrupt-handling routine issues an immediate command to the device to reenable interrupts when it has finished processing a chain or single control block and wants to receive more signals from the first-level interrupt-handling routine.

If an interrupt is not cleared from the Interrupt Status port, it will cause a later physical interrupt when the subsystem and the interrupt level are reenabled.

Non-Control Block Completed without Error, Reset Subsystem

The clearing and processing of an interrupt of this type is straightforward. It consists of the following steps:

1. Clear the Interrupt Status port with a Reset Interrupt Status Port command with the Attention port device ID set to the device ID read from the Interrupt Status port.

2. Find the program that issued the command to the device and cause it to gain control (unblock it if in a multitasking environment).
3. The program that issued the command should easily determine the specific command completed, because only one non-control-block command can be active for a device at a time.
4. Issue an End of Interrupt command to reset the system-master interrupt controller to free the interrupt level.

The End of Interrupt command referred to is the command described in the documentation for the system-master interrupt controller. It is not a Locate mode architected command, but it is included here for implementation information.

When an interrupt controller exists between the subsystem and the system master, the resetting of the interrupt controller must be explicitly performed in addition to the reset, which is required by the subsystem to clear an interrupt. Interrupt Reset commands in this case should be issued as follows:

1. Reset Interrupt Status Port or Reset Control Block Interrupt
2. End of Interrupt to the interrupt controller.

This ensures that the subsystem interrupt is cleared so that the subsystem can present a new interrupt when the interrupt controller is cleared. If the interrupt controller is cleared without the subsystem interrupt being cleared first, a physical interrupt from the interrupt controller will be received as a result of the prior subsystem interrupt.

Non-Control Block Completed with Error or Hardware Failure

Handling an interrupt of this type is more complex than handling a non-control-block command that was completed without error. The processing is identical for both cases to the point of recovery for the non-control-block command. The additional complexity occurs in handling the recovery for interrupts of this type. Recovery is determined, for the most part, by each implementation. For more detail, see "Exception and Error Handling" on page 5-4.

System Master – Control Block Interrupts: Interrupt handling for control blocks can involve the use of first- and second-level interrupt-handling routines, discussed in "System Master – Physical Interrupts" on page 3-31. This is the case because the second-level interrupt-handling routine is typically given the task of determining which control block in the command chain was the source of the interrupt, and whether the chain had gone to completion. The task of locating a particular control block in a chain can involve a search through termination status blocks and interrogation of control-block Enable Word 1. In a multitasking environment, it is best to have this search done in a task that is separate from a first-level interrupt-handling routine. This leaves the first-level interrupt-handling routine free to service other devices on the subsystem.

A second-level interrupt-handling routine would probably be signalled while the device it is servicing has been disabled for interrupts. This action would be taken by the first-level interrupt-handling routine as part of clearing the Interrupt Status port with a Reset Interrupt Status Port command. Disabling the device performs two functions. First, it assures the second-level interrupt-handling routine that it will not be reentered from the first-level interrupt-handling routine until it specifically reenables the device. Second, it assures the first-level interrupt-handling routine that it will not be called upon to service additional interrupts from a device until the second-level interrupt-handling routine has processed its work request from the first-level interrupt-handling routine. This allows devices on the adapter to continue to obtain interrupt service from the first-level interrupt-handling routine while others are being serviced by the second-level interrupt-handling routine. A second-level interrupt-handling routine, when signalled, is supplied with the data read from the Interrupt Status port when the interrupt was received by the first-level interrupt-handling routine.

Searching Control-Block Chains: The second-level interrupt-handling routine can also be designed to determine which control block in a command chain has interrupted and whether the chain is complete. This task can be greatly simplified if certain programming conventions concerning control-block chains are followed. Some examples of these conventions are:

- Always request a successful completion interrupt in the last (or only) control block in a command chain.

- Ensure that End Status Word 1 in each termination status block in the chain is set to 0 before the chain is issued to the device.
- Once a chain is initiated, do not modify control blocks that have not been reported as complete.

By searching the termination status blocks of a chain of commands, a second-level interrupt-handling routine can find the specific command control block that interrupted and can determine the state of a chain of control blocks. After analyzing the results of the search, the second-level interrupt-handling routine can take one of the following paths:

- If the chain has ended and has not been suspended, the second-level interrupt-handling routine should reenable interrupts for the device. Status should be returned to the program that created the chain. The interrupt code passed to the second-level interrupt-handling routine by the first-level interrupt-handling routine determines the kind of ending status given.
- If the chain has not ended and has not been suspended, the second-level interrupt-handling routine should log any error found, enable the device for interrupts, and block itself waiting for the next signal from the first-level interrupt-handling routine for the command-control-block chain.
- If the chain has been suspended, the second-level interrupt-handling routine needs to log any errors found. If the chain has ended, the second-level interrupt-handling routine does not need to save state to restart the chain at a later point. The second-level interrupt-handling routine can return a final result of the program that created the chain or control block. Status should be returned to the program that created the chain. The interrupt code passed to the second-level interrupt-handling routine by the first-level interrupt-handling routine determines the kind of ending status given. If the chain has not ended, the address of the next control block in the chain can be used to restart the chain at a later point if necessary. Any error found is noted.

  Another control-block command stream can be started at the device after the device is enabled for interrupts. This can be done by issuing a write to the Command and Attention ports. The

second-level interrupt-handling routine will block itself waiting for new interrupts if a new command stream is started.

In the cases cited previously where the device is still active on a control-block chain, the second-level interrupt-handling routine reenables interrupts by using the Command Reset Interrupt Status port with its disable-device-interrupt bit set to 0. This command will be accepted by a device that is busy executing a control block.

Control Block Completed without Error

When the interrupt ID that is returned to the interrupt-handling program in the system unit indicates that the control-block command was completed without error, there are two cases to consider:

- If the interrupt-handling program knows that the control block is not part of a chain, it can quickly determine which control-block command caused the interrupt, without examining the termination status blocks. This follows from the fact that only one control block was active.
- If the interrupt-handling program does not know the chain status of the control block, it should use a chain-searching technique to locate the interrupting control block.

Although the architecture permits a program to suppress storing termination status blocks for control-block commands that are completed without error, the use of this option must be weighed against the extra analysis that could be needed in interrupt-handling programs to handle chains of control blocks that use interrupts on normal completion to track progress.

A Reset Interrupt Status Port command will clear this interrupt condition from the subsystem and make the Interrupt Status port available.

Control Block Completed with Error

When the interrupt ID indicates that an error occurred that was not caused by the hardware, the interrupt-handling program needs to examine the termination status block associated with the control block to determine the cause of the error.

If a control block is not part of a chain, the interrupting control block can easily be identified because a device can have only one control block active at a time.

When the command control block is part of a chain operation, the interrupt-handling program might be required to search the control-block chain to determine which control block requested an interrupt. This is done by examining, in chain order, each control block that is thought to be incomplete, and testing for completion with termination-status-block status indicating that it requested an interrupt. The first termination status block found indicating a request for an interrupt locates the failing command control block. For more details on exception handling see "Exception and Error Handling" on page 5-4.

Control-Block-Command Hardware Failure

When the interrupt ID indicates that an error caused by the hardware occurred while a control block was being processed, the interrupt-handling program cannot accurately identify the control block when it is part of a chain. This is because a termination status block is not stored for hardware-related errors, even though the control-block chain is terminated.

If a control block is not part of a chain, the interrupting control block can easily be identified because a device can have only one control block active at a time.

DIIP Interrupts (Multiple Form)

In this form of interrupt handling, command-control-block interrupts are not restricted to presentation in the serially-reusable Interrupt Status port. Instead, they are presented through the Device Interrupt Identifier port (DIIP), which provides a means of signalling interrupts for multiple devices in a single port. Not only can multiple devices present logical interrupts in a single DIIP, but each device that signals the presence of a logical interrupt in the DIIP by setting its designated device bit can indicate that several command control blocks have been completed with their interrupt-request bits set to 1 in End Status Word 1 of the termination status blocks.

In this interrupt mode, every command control-block interrupt is a logical interrupt and results in at least End Status Word 1 being

written to memory before a physical interrupt request is made. This differs from the non-DIIP-interrupt case, in which termination-status-block status can be disabled for completion of error-free command control blocks when an interrupt is requested.

As defined in the description of the DIIPs, each device is assigned a unique bit in the DIIP that, when set to 1, indicates that at least one command-control-block logical interrupt has been raised for the assigned device. Each time a device detects a command-control-block logical interrupt, an internal count of logical interrupts presented by the device is incremented by 1. If this internal count of logical interrupts presented by the device is greater than 0, the assigned DIIP bit for the device is set to 1, signalling the presence of a logical interrupt in the DIIP. If the device that requested the command-control-block interrupt is enabled for interrupts and the subsystem is also enabled for interrupts, a physical interrupt will be requested.

When the DIIP is supported, at least one DIIP must exist. The implementation defines as many DIIPs as are needed to address the number of devices attached to it. When multiple DIIPs exist, they must occupy consecutive addresses in shared I/O space. The first DIIP defines its least-significant bit 0 to indicate that command-control-block logical interrupts have been raised for the subsystem. Bits 1-15 are then assigned to the first 15 devices attached to the subsystem. Any remaining devices are assigned to other DIIPs, with each port supporting 16 devices. Device assignment continues in each port, starting at the least-significant bit and extending to the most-significant bit. Unassigned bits of a DIIP must always be set to 0.

Device Control-Block Interrupts: When a command control block is completed with an interrupt request, at least End Status Word 1 is stored in the termination status block associated with the command control block, indicating that a logical interrupt request is pending for this command. The device issuing the request increments an internal logical interrupt count by 1 for every logical interrupt request. If the logical interrupt count is greater than 0, the device detects the presence of a logical interrupt by setting its associated bit to 1 in the Device Interrupt Identifier port (DIIP). A physical interrupt for a device is presented to the system master if the following conditions are met:

- The DIIP bit for the device is set to 1.
- The device is enabled for interrupts.
- The subsystem is enabled to send interrupts.

The DIIP interrupt architecture allows a program to reset logical interrupts as soon as they are detected. Typically, a program detects the need to process logical interrupts for a device by reading a subsystem DIIP. The number and specifics of the command control blocks that have been completed with logical interrupts are then determined by examining the End Status Word 1 of control-block commands associated with the device whose DIIP bit is 1.

Logical interrupts that have been processed by programs in the system master are cleared for a specific device by issuing a Reset Control Block Interrupt command. This command contains a count value that allows a program in the system master to reset as many as 15 logical interrupts for a device at one time. The count value specified in the command is subtracted from the internal logical interrupt count for the device. If the resulting new logical interrupt count for the device is less than or equal to 0 as a result of the Reset Control Block Interrupt command, the DIIP bit associated with the device is set to 0 in the appropriate DIIP. This signals the fact that the device currently has no outstanding logical interrupts left to process, and it prevents any further interruption of the system master on behalf of the device for logical interrupts.

When the DIIP architecture is implemented, most control-block interrupts are presented to the system master through the DIIPs. The only exception to this rule occurs when a command control block cannot be read by a device or the subsystem or when completion

status could not be stored in the control block's termination status block by the subsystem or device.

If the command control block could not be read or the termination-status-block status could not be written, the interrupt request is not presented in the DIIP, but is signalled as interrupt code hex 8 (Hardware failure—control block command) in the Interrupt Status port. This type of interrupt is handled in an identical fashion to the non-DIIP interrupt, and the resulting interrupt must be signalled to the system master in the Interrupt Status port of the subsystem.

Note: If the failing command control block is an element in a command chain, it might not be possible to locate the failing control block, because no termination-status-block status has been stored.

When the Interrupt Status port is written to and the DIIP is supported, the same rules apply for setting the interrupt-valid bit in the Command Busy/Status port to 1. Interrupt-handling software in the system master should test the interrupt-valid bit so that it can determine whether the Interrupt Status port needs to be read to handle the interrupt request recorded there. A practice of testing and reading the Interrupt Status port before reading any DIIPs ensures that control-block hardware failures are handled on a higher priority than are logical interrupts caused by recoverable command-control-block completions.

As in the non-DIIP-interrupt case, a control-block hardware-failure interrupt is cleared from the Interrupt Status port by the Reset Interrupt Status Port command. Interrupt data remains in the Interrupt Status port until it is explicitly reset by the system master.

Software in the system master can detect that a control-block logical interrupt has been raised for a device before it receives a physical interrupt.

The recommended method for detecting control-block interrupts is to read the DIIPs for any nonzero bits. Nonzero bits represent devices that have at least one control-block logical interrupt. A nonzero value for a DIIP bit indicates that at least one control-block logical interrupt exists for a program to handle.

Note: This method of detecting control-block interrupts before receiving the physical interrupt is simpler and uses fewer system-master processor cycles in the DIIP-interrupt case than in the non-DIIP-interrupt case. In the non-DIIP-interrupt case, it is possible to detect a logical interrupt before the physical interrupt is requested. The program in the system master can do this by polling the termination status blocks for one with the interrupt-request bit set to 1. This is not recommended because it can waste processor cycles that could be used by other tasks.

Once a logical interrupt is found, the DIIP-interrupt case is superior to the non-DIIP-interrupt case because it allows a logical interrupt to be reset without having to wait for the interrupt value to be presented in the subsystem's Interrupt Status port.

Device Non-Control-Block Interrupt: If a non-control-block logical interrupt is found by a device, it is signalled to the system master in exactly the same manner as in the non-DIIP-interrupt case. The interrupt code is signalled in the Interrupt Status port. The specific codes are:

- Immediate/hardware-controlled command completed, no error
- Immediate/hardware-controlled command completed, with error
- Immediate/hardware-controlled command completed, hardware failure
- Reset subsystem or device completed, no error.

Subsystem – Physical Interrupts (DIIP Form): A subsystem can raise a physical interrupt to the system master, following the same steps as in the non-DIIP case (see pages 3-16 through /SFEND/). In addition, a physical interrupt can be raised if a device is enabled for interrupts and its associated DIIP bit is set to 1. The interrupt-valid bit is set to 1 only if the Interrupt Status port contains a valid interrupt value. The DIIP bits are reset by the Reset Control Block Interrupt command (see "Reset Control Block Interrupt" on page 1-80 for details).

System Master – Physical Interrupts: The following discussion deals with programming considerations in the system master with the Device Interrupt Identifier port (DIIP) interrupt-handling model of the Locate mode architecture.

A subsystem will attempt to interrupt the system master if the following conditions are met:

- The subsystem is enabled to send interrupts to the system master.
- Either of the following conditions is true:
  - The Interrupt Status port has an interrupt value in it.
  - Any device is enabled for interrupts, and its associated DIIP bit is set to 1.

If these conditions are met, the system master will receive an interrupt from the subsystem.

Unlike the non-DIIP interrupt, a physical interrupt in the DIIP-interrupt case can indicate that at least one device has presented a control-block logical interrupt and a device has interrupt data present in the Interrupt Status port.

In order to determine the source of a physical interrupt, a program in the system master must read both the Command Busy/Status port and the DIIPs. As in the non-DIIP-interrupt case described in "System Master – Physical Interrupts" on page 3-20, interrupt handling can be viewed as consisting of two levels: a first-level interrupt-handling routine and a second-level interrupt-handling routine.

When a system master receives a physical interrupt, it invokes the first-level interrupt-handling routine associated with the subsystem. This is based upon the interrupt level assigned to the subsystem at configuration and initialization time.

The first-level interrupt-handling routine determines the interrupt state of a subsystem by reading its Command Busy/Status port and DIIPs. If the interrupt-valid bit in the Command Busy/Status port is 0 and all the bits in the DIIPs are also 0, the interrupt is not for this subsystem (interrupt sharing). The interrupt handler simply returns control to its caller with an indication that it did not have an interrupt to service.

If an interrupt exists for the subsystem or any of its devices, a procedure is followed that allows the first-level interrupt-handling routine to schedule the second-level interrupt-handling routine to process device requests for interrupt servicing. A first-level interrupt-handling routine typically tries to schedule all the requests it received on a single invocation to the appropriate second-level interrupt-handling routines. Because second-level interrupt-handling routines are scheduled, the first-level interrupt-handling routine must not accept further interrupts from the device. This is done by disabling the devices. While the devices are disabled, the first-level interrupt-handling routine ignores the DIIP bits for these devices when it is reentered on an interrupt request. (The second-level interrupt-handling routines may not have finished resetting all interrupts at that time.)

Devices are disabled by the first-level interrupt-handling routine in two ways:

- When an interrupt is found in a DIIP.

  A Reset Control Block Interrupt command is used with the disable-device-interrupt (DDI) bit set to 1. The count value in the command is set to 0. This allows the second-level interrupt-handling routine to reset as many logical interrupts as it encounters in its processing, using a true interrupt count.

- When an interrupt is found in the Interrupt Status port.

  A Reset Interrupt Status Port command is used with the disable-device-interrupt (DDI) bit set to 1.

Second-level interrupt-handling routines need to use the services provided by the first-level interrupt-handling routine to reenable a device for interrupts. This allows the second-level interrupt-handling routine to be capable of being called again by the first-level interrupt-handling routine when the first-level interrupt-handling routine detects that the DIIP bit for its device is set to 1.

When all DIIPs have been processed, the first-level interrupt-handling routine returns control to its caller. It indicates that it has cleared an interrupt when it schedules any second-level interrupt-handling routine. If no second-level interrupt-handling routine was scheduled on entry to the first-level interrupt-handling routine, no interrupts were found or cleared by it on this invocation.

As each second-level interrupt-handling routine is scheduled by the first-level interrupt-handling routine, it passes on information explaining why it is being invoked. In the case of an interrupt found in the Interrupt Status port, it would most likely be the contents of the Interrupt Status port Interrupt ID field. For routines that are scheduled to handle logical interrupts, a code is passed that indicates that a search of the command control blocks is needed.

The procedure for dealing with an interrupt found in the Interrupt Status port in the DIIP-interrupt case is similar to that in the non-DIIP-interrupt case. If the interrupt-valid (IV) bit in the Command Busy/Status port is set to 1, the first-level interrupt-handling routine must read the Interrupt Status port for the subsystem to determine the device and the cause of the interrupt. The Interrupt ID field indicates the type and cause of the interrupt, and the Device ID field indicates the identity of the device.

The interrupt should be cleared as soon as possible to avoid blocking the interrupt level and to free the subsystem to present interrupts from other attached devices. The interrupt is cleared by the Reset Interrupt Status Port command. The Attention port must be set to the Device ID read from the Interrupt Status port. In this case, the clearing action is done before analysis of the cause of the interrupt is completed. The clearing operation is done in the first-level interrupt-handling routine, and a second-level interrupt-handling routine is scheduled.

The second-level interrupt-handling routine can be the program that originally issued the command. To be most effective, the second-level interrupt-handling routine should be waiting for a signal to continue processing. The first-level interrupt-handling routine provides the needed signal, and it unblocks the execution of the second-level interrupt-handling routine. In the DIIP case, the first-level interrupt-handling routine continues processing to determine whether it needs to schedule second-level interrupt-handling routines for DIIP interrupts.

Non-Control Block Completed without Error, Reset Subsystem or Device

When the interrupt ID from the Interrupt Status port indicates that a non-control-block command was completed without error, processing of the interrupt consists of a Reset Interrupt Status Port command, followed by an End of Interrupt command to reset the interrupt controller on the system master to free the interrupt level.

Note: The End of Interrupt command referred to is the command described in the documentation for the system-master interrupt controller. It is not a Locate mode architected command, but it is included here for implementation information.

Non-Control Block Completed with Error, Non-Control-Block Hardware Failure

When the interrupt ID indicates that a non-control-block command was completed with an error, recovery depends upon the interrupt code returned and the command that was submitted. The program must keep track of all submitted non-control-block commands that have not been completed. There can be only one non-control-block command active for each device. The interrupt is cleared in the same way as a non-control-block command that was completed without error. For more details on error handling, see "Exception and Error Handling" on page 5-4.

Control-Block-Command Hardware Failure

When the interrupt ID indicates that a hardware failure occurred while a control-block command was being processed, the program might not be able to locate the exact control block that failed, if the control block is an element of a control-block chain. This is because, in these cases, termination-status-block status is not stored.

In the case of a single nonchained control block, a device can have only one control-block address active at a time, so the interrupting control block is immediately known. In the case in which a control-block chain might be present, a second-level interrupt-handling routine and a control-block-searching procedure can be used. In this case, the first control block that has an End

Status Word 1 of 0 is located. The execution of the control-block chain is terminated by the device when such a condition is signalled.

For more details on error handling, see "Exception and Error Handling" on page 5-4.

DIIP Interrupt Present

To process a logical interrupt in the DIIP interrupt architecture, a program in the system master must read the termination status blocks associated with the control-block commands. This happens because only one DIIP bit is returned to the system master to indicate that the device has posted logical interrupts in shared storage.

Because this is the case, a programming convention needs to be employed to simplify the task of locating the termination status block of a control block that is complete. The same convention and searching approach that is described in the non-DIIP-interrupt case can be used; that is:

- Always request a successful completion interrupt in the last (or only) control block in a command chain.
- Ensure that the End Status Word 1 in each termination status block in the chain is set to 0 before the chain is issued to the device.
- Do not modify control blocks that have not been reported as complete once a chain is initiated.
- Enable termination-status-block status storage on the completion of every control block.

Using this convention, a second-level interrupt-handling routine can be written that uses a search program like the one described to find control-block interrupts in the non-DIIP-interrupt case. The difference in the DIIP case is that a second-level interrupt-handling routine uses a loop to call the search routine. The objective of the loop is to call the search routine to find logical interrupts and to count them. As each interrupting control-block command is found, any error or exception condition that is found is processed. For a discussion of error handling, see "Exception and Error Handling" on page 5-4. Calls to the search routine continue if all the following conditions are met:

- The search routine indicates that an interrupt has been found.
- The end of the chain has not been reached.
- The chain has not been suspended.

If any of these conditions are not met, searching stops.

After calls to the search program are stopped in the second-level interrupt-handling routine, several paths are possible. On any path, the logical interrupts found are reset. Resetting is done by the Reset Control Block Interrupt command. As many as 15 interrupts can be reset with one use of the command. If more than 15 interrupts are found, multiple Reset Control Block Interrupt commands are used.

The actions for stopping the search are as follows:

- If the chain has ended and has not been suspended, the second-level interrupt-handling routine should reset the interrupts found and reenable interrupts for the device. The second-level interrupt-handling routine should return status to the program that created the chain. The ending status provided by the second-level interrupt-handling routine is determined from the End Status Word 1 of the last interrupting control block.
- If the chain has not ended and has not been suspended, the second-level interrupt-handling routine should log any errors found. The second-level interrupt-handling routine resets the logical interrupts found and enables the device for interrupts. The second-level interrupt-handling routine blocks itself waiting for the next signal from the first-level interrupt-handling routine for the control-block chain.
- If the chain has been suspended, the second-level interrupt-handling routine needs to log any errors found. The second-level interrupt-handling routine resets the number of logical interrupts found. If the chain has ended, status does not need to be saved by the second-level interrupt-handling routine to restart the chain at a later point. Final status can be returned to the program that created the chain or control block. The ending status provided by the second-level interrupt-handling routine is determined from the End Status Word 1 of the last interrupting control block found by the search program. If the chain has not ended, the address of the next control block in the

chain, returned by the search routine, can be used to restart the chain at a later point. Any error found is noted.

Another command-control-block stream can be started at the device after the number of logical interrupts found is reset and the device is enabled for interrupts. This is done by issuing a write to the Command and Attention ports.

The second-level interrupt-handling routine will block itself waiting for new interrupts if a new command stream is started.

In the cases cited previously where the device is still active on a control-block chain, the second-level interrupt-handling routine reenables interrupts for a device by calling the first-level interrupt-handling routine, which issues a Reset Control Block command with a count of 0 and with its disable-device-interrupt bit set to 0. This command will be accepted by a device that is busy executing a control block.

The order in which devices are processed for control-block interrupts and the number of control-block interrupts that are processed for each device are determined solely by the interrupt-handling software.

Note: System software is free to read DIIPs at any time, but in a typical case, these ports would be read after the system master received the physical interrupt.

DIIP versus Non-DIIP: The description of DIIP (multiple form) interrupt handling can be concluded by summarizing its advantages over the non-DIIP (simple form) interrupt case. The advantages are:

- Multiple interrupt sources are presented in the DIIP case.

  In the DIIP case, interrupt-handling software in the system master can determine the interrupt status of all its devices on a single physical interrupt of the system master and decide the order of service. In the non-DIIP case, only one device source is presented to the system master.

- Multiple logical interrupts can be reset.

  Multiple control-block logical interrupts from one device can be reset with a single execution of the Reset Control Block Interrupt command in the DIIP case. The non-DIIP case allows only one to be reset.

- Logical interrupts can be reset quickly.

  In the DIIP case, logical interrupts can be reset as soon as they are detected in the system master. Software in the system master does not have to wait for a value to be presented in the Interrupt Status port, as in the non-DIIP case.

- Device logical-interrupt status can be found quickly.

  The use of single bits for a device in a DIIP allows programs in the system master to determine whether any logical interrupts are pending for a device without having to poll each termination status block, as in the non-DIIP case.

- Logical interrupts are presented quickly.

  In the DIIP case, logical interrupts are queued quickly at the device by incrementing a simple count. They are presented to the system master immediately by setting the assigned DIIP bit each time a device requests a logical interrupt and its internal count is nonzero. This means that control-block logical interrupts can be responded to in the order the program in the system master selects, rather than waiting for the subsystem to serialize presentation through a single Interrupt Status port.

Resetting Interrupts

This section describes the actions and effects of Reset type commands on the Interrupt Status port and the Device Interrupt Identifier ports (DIIPs).

Interrupt Status Port: The value in the Interrupt Status port remains constant once it is set by the subsystem or a device, until software in the system master clears the port with a Reset Interrupt Status Port command or a Reset Subsystem command.

The normal method of clearing the Interrupt Status port is the Reset Interrupt Status Port command. This command clears the Interrupt Status port when the port contains a valid interrupt value and the device that caused the interrupt matches the value specified in the Attention port. After the Interrupt Status port is cleared, the interrupt-valid (IV) bit in the Command Busy/Status port is set to 0.

Device Interrupt Identifier Ports: The Device Interrupt Identifier ports (DIIPs) are reset by the Reset Control Block Interrupt, Reset Device, or Reset Subsystem command.

The normal method of clearing logical interrupts and DIIPs is the Reset Control Block Interrupt command.

When the Reset Control Block Interrupt command is used, the reset count specified in the command should match the number of logical interrupts to be cleared. If the reset count used is greater than or equal to the number of logical interrupts counted internally by the device, the subsystem sets the DIIP bit for the device to 0 and will not set the DIIP bit for the device until enough control-block interrupts occur for the device to set its internal logical-interrupt count to a positive value. When the DIIP bit for a device is 0, it cannot raise a physical interrupt.

A Reset Device command sets the DIIP bit for the device to 0 and sets the device's internal logical-interrupt count to 0. An interrupt is requested when execution of the Reset Device command is complete, because a Reset Device command always requests an interrupt upon successful completion. (For details, see "Reset Device" on page 1-77.)

Notes:

Chapter 4. Processing Level

In general, the services and protocols at the processing level are not defined by the architecture. However, the architecture does define configuration and initialization, exception handling, and the control-block and data-chaining protocols. The control-block and data-chaining protocols are described in this section. A discussion of configuration, initialization, and exception handling can be found in "Exception and Error Handling" on page 5-4.

There are two types of control-block chaining:

- Command chaining.

  Command chaining allows a series of control blocks to be presented to a device for execution with a single request. The request points to the first control block in a chain of control blocks. This frees the system master to perform other work.

- Data chaining.

  Data chaining uses an indirect list pointed to from the control block to perform data transfers into a set of separate shared-memory areas (pages). This capability is useful when dealing with data buffers in a virtual shared-memory system that are logically contiguous but are mapped by the operating system into separate pages.

© Copyright IBM Corp. 1991

Command Chaining

A command-control-block address is delivered from the system master to the subsystem through I/O instructions to write to the Command and Attention ports. Once this command address is delivered and accepted, the device performs the operation specified by the command control block. If command control blocks are issued to the device one at a time, a program in the system master will wait for the command control block to be completed before it issues the next control block to the device. Such a practice involves at least three I/O operations from the system master to initiate the control block and verify its acceptance. At least three I/O operations are needed to verify and reset an interrupt from the device on completion.

If a program needs to issue several commands to a device, instructions are saved in the system master by using command chaining. This is illustrated in Figure 4-1 on page 4-3 (Control Blocks without Command Chaining) and Figure 4-2 on page 4-4 (Control Blocks with Command Chaining).

Figure 4-1 on page 4-3 shows an example of three command control blocks initiated as individual commands from the system master. As indicated, each I/O command requires at least two I/O operations to be delivered to the subsystem through the Attention and Command ports. An I/O operation to the Command Busy/Status port is required to verify command acceptance.

Before control block 2 is sent to the device, control block 1 must be completed and must interrupt the system master. This requires at least three I/O operations from the system master:

- One I/O operation to read the Interrupt Status port
- Two I/O operations to write to the Command and Attention ports to clear the Interrupt Status port.

A program in the system master can then issue the next command control block (control block 2). The procedure is the same as the procedure for control block 1. When control block 2 is completed, control block 3 is issued.

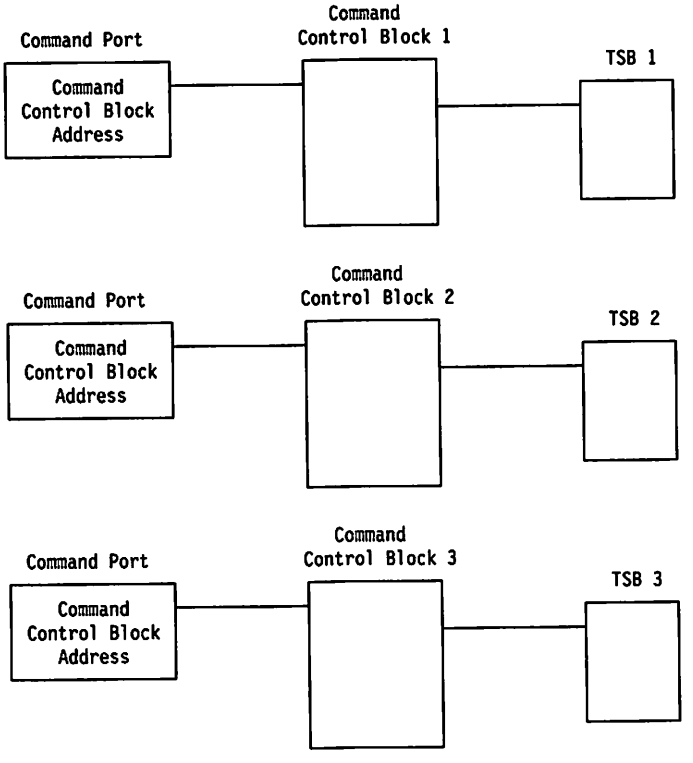


Figure 4-1. Control Blocks without Command Chaining

When command-control-block chaining is used, the system master issues one set of I/O commands to the subsystem to initiate the chain, and the device performs the three I/O operations independently of the system.

In the following figure, the operation defined in control block 1 is executed, and the completion status is written to the termination status block. After this is done, the device reads the next control block (control block 2), that is, independent of the system master. Control block 2 is executed by the device, and the completion status is written to its termination status block. The device then reads and executes control block 3. After control block 3 has been executed, its completion status is written to its termination status block, and a system-master interrupt is requested. This notifies the program that all three commands have been successfully executed.

The system master is free to perform other work while the device is executing control blocks 1, 2, and 3 and does not have to execute extra instructions to submit each control-block command. This is useful in a multitasking environment. See "Signalling Protocol" on page 3-13 for details of programming conventions, the architecture for interrupts, and command-control-block chains.

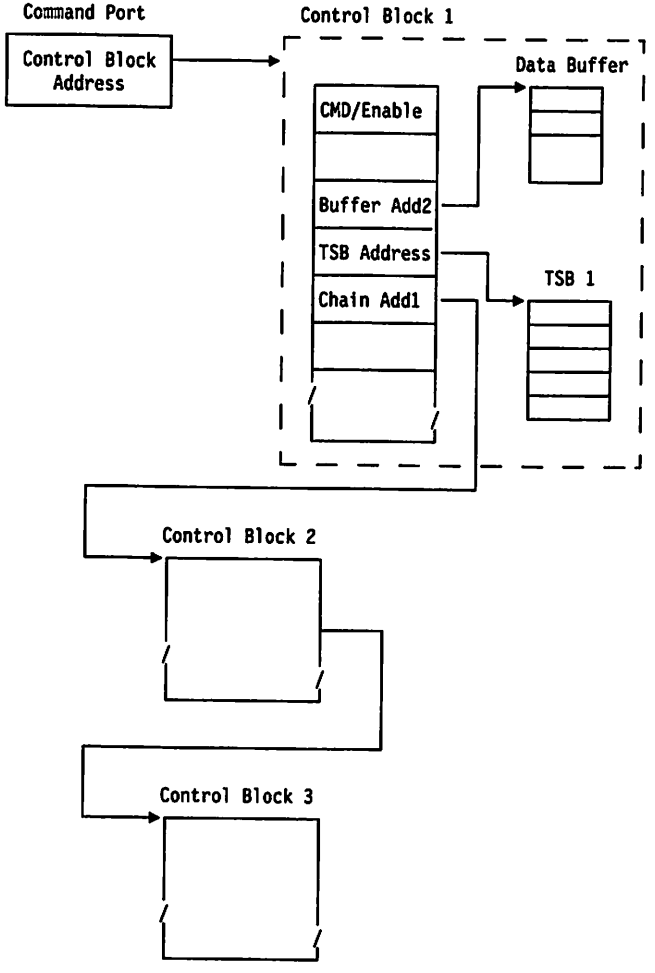


Figure 4-2. Control Blocks with Command Chaining

Conditional Command Chaining

The Locate mode architecture supports the ability to conditionally continue processing command control blocks in a command chain if the current control block is completed with a specified condition. This is called conditional chaining, and it uses a separate control-block-chain address field (Chain Address 2) to specify the next control block to be executed. The form of command chaining illustrated in Figure 4-2 on page 4-4 is called Chain No Error. It uses the other chain-address field in the control block (Chain Address 1) to specify the next control block to be executed when the current control block is completed without error. An individual control block can specify one or both forms of command chaining. When only one form is enabled, the chain continues down the specified path when the control block is completed in a manner that satisfies the chain specification. When both forms of chaining are specified in a control block, two branches in the chain are enabled. The control-block execution path is Chain Address 2 if that condition is met. If the condition is not met and the control block is completed without error, the Chain Address 1 execution path is followed. If neither condition is met, the chain is halted.

An example of the use of both Conditional Chaining and Chain No Error is shown in the following figure. In the example, control block 1 uses the chain condition specification, Chain on Error, and also enables Chain No Error.

If control block 1 is completed with an error and the conditional chain condition is met, the chain continues using Chain Address 2. In the following figure, the path of chain execution is Chain Address 2.

If the condition is not met and control block 1 is completed without error, the chain continues using Chain Address 1. In this case, the path of execution follows Chain Address 1.

Using chaining in this manner allows a program to provide a predetermined error-recovery path. The conditional-chain path is used for commands that are performed dependent on the specified condition.

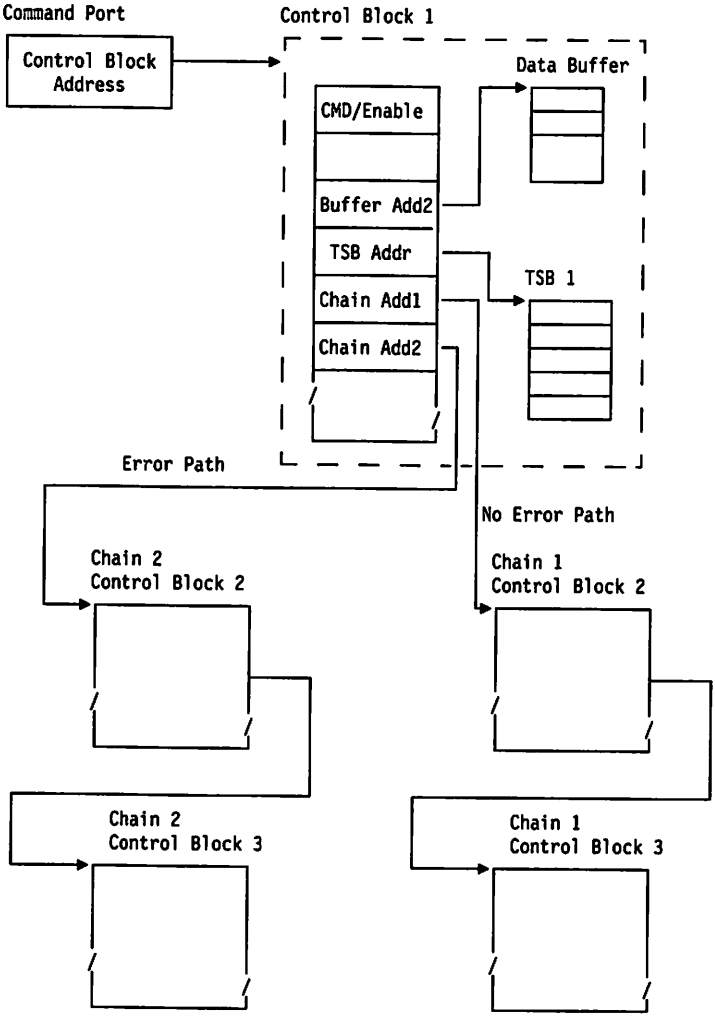


Figure 4-3. Command Chaining with Conditional and Chain No Error Enabled

Command-Chaining Options

Command chaining is specified by setting the chain-no-error bit or the conditional-chaining bit in Enable Word 1 of a command control block. This results in four types of chaining:

- No chaining
- Chain no error (Enable Chain Address 1)
- Conditional chain (Enable Chain Address 2)
- Two-way chain (Enable Chain Address 1 and 2).

The types of chaining are described in detail below.

- No chaining.

  The operation in the control block stands alone. After the device has completed processing of the control block, the device is ready to execute another command from the program. This is specified when the chain-no-error and conditional-chaining bits in Enable Word 1 of the current command control block are both 0.

- Chain no error (unconditional chain).

  This type of command chaining is enabled if the chain-no-error bit in Enable Word 1 of the command control block is set to 1 and the conditional-chaining bit is set to 0. If the current control block is completed without error or is completed with an error that has been suppressed, chaining continues, using control-block Chain Address 1 to read the next control block. An example of this type of chain is a write to a disk, followed by a read to verify the data.

  If a control block terminates with an unsuppressed error, the chain is halted.

- Conditional chaining.

  This type of command chaining specifies that chaining is to use the Chain Address 2 field of the current control block when the control block is completed in a manner that matches the chain-condition-specification bits in Enable Word 1 of the current control block.

  If the current control block is not completed in a manner that matches the chain-condition-specification bits in Enable Word 1 of the current control block, chaining is halted.

  The conditions that can be specified by the settings of the chain-condition-specification bits are:

000 ***Chain on Error***

An unsuppressed error/exception occurred. The errors excluded from this condition are: specification error, hardware failure reading a control block, and hardware failure storing a termination status block.

001 ***Chain on More Status***

The device or subsystem completed the current command control block in a state that indicates that more status is available at the device than could be stored in a termination status block.

010–111 These codes are reserved.

Note: A program in the system master can determine whether a command control block was completed by selecting the conditional chain address (Chain Address 2). This is indicated by the setting of the chain-direction bit in the termination-status-block End Status Word 1. To ensure that the termination status block is stored when conditional chaining is specified, the program should set the termination-status-block bit in End Enable Word 1 of the control block to 0 when the chain condition is Chain on More Status. The chain condition Chain on Error causes termination-status-block End Status Word 1 to be stored, ignoring the setting of the termination-status-block bit in Enable Word 1.

- Two-way chaining.

  This type of chaining describes the actions to be taken when both Conditional Chaining and Chain No Error are enabled in the same control block (both the chain-no-error and conditional-chaining bits are set to 1).

  Note: When both forms of chaining are specified in a single control block, testing is performed first to determine whether the Conditional Chaining specification is met. If it is not met, testing is performed to determine whether the Chain No Error condition is met.

  Two-way command chaining specifies that chaining is to continue using the Chain Address 2 field of the current control block, if the current control block is completed in a manner that matches the

conditions specified in the chain-condition-specification bits in Enable Word 1.

If the current control block is not completed in a manner that matches the chain-condition-specification bits in Enable Word 1 of the current control block, chaining continues, using the Chain Address 1 field of the current control block, provided the control block was completed without error or with an error that was suppressed.

Chaining is halted if the chain condition does not match the ending condition of the current control block or if the control block is completed with an unsuppressed error.

Control-Block Execution

Control blocks in a chain must be executed in the sequence in which they are presented. This allows programs to depend on a predetermined order of execution. The device cannot perform the next control block until it has completed all operations on the current control block, including the storing of the termination status block.

In the Locate mode, the ordering restriction on command execution also applies to all commands outside of a chain. The restriction is applied to simplify and speed up interrupt-handling software in the system master.

Data Chaining

Data chaining, often referred to as indirect lists, allows a single control block to gather data from several areas of shared memory and transfer it to or from a destination or source.

Data chaining is enabled with the indirect-list-1 (IL1) and indirect-list-2 (IL2) bits in the control-block Enable Word 1 and control-block Enable Word 2, respectively. In the base control block, only IL1 is used, because only one byte count is present to specify the length of the indirect list in system shared memory. This data area, whether used as a source or destination, is accessed by incrementing addresses. With the extended command control block, both Byte Count 1 and Byte Count 2 can be used to specify two data areas addressed by Shared Memory Address 1 and Shared Memory Address 2. Each address can be specified as incrementing or decrementing during the data transfer.

Note: IL1 is associated with the shared-memory address chosen by the setting of the memory-address-1-selected (MA1) and memory-address-2-selected (MA2) bits in Enable Word 1. From this point on, the term "Memory Address x" will be used to refer to either shared-memory address, as selected by the MA1 and MA2 bits.

If IL1 is enabled, Memory Address 1 or Memory Address 2 points to the doubleword-aligned beginning of an indirect list of buffer descriptors. The indirect list is in error if Memory Address x points to an invalid address, the indirect list does not begin on a doubleword boundary, or the length of the indirect list is not a multiple of eight bytes. A specification exception might be noted in these cases.

For reference, an indirect list is illustrated in Figure 4-4 on page 4-11.

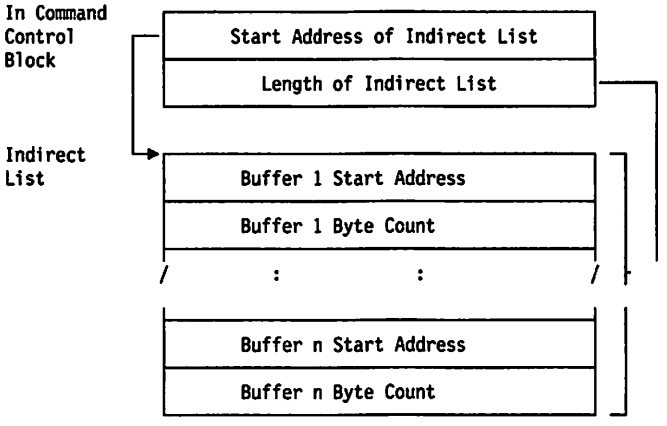


Figure 4-4. Indirect List

Each element (buffer descriptor) in the indirect list consists of eight bytes. The first four bytes define the buffer start address—the location in shared memory to be used as the source or destination of a data transfer. The second four bytes define the buffer length—the size, in bytes, of the data transfer.

The total data-transfer length defined for the control block is the sum of the buffer lengths of all the elements in the indirect list.

The Byte Count 1 field in the control block, associated with Memory Address x, contains the length, in bytes, of the indirect list. The indirect list is in error if the byte count is not a multiple of eight bytes. Because each element consists of eight bytes, the number of elements in the list is Byte Count 1 divided by 8. A specification error exists if any element points to an invalid address. Testing of element validity must be performed before data transfer begins. If no specification testing is done on each address, the results of the operation are not defined by the architecture. An example of a command control block with an indirect list is shown in Figure 4-5 on page 4-12.

Note: The indirect list is useful for software systems that are running user programs in a virtual shared-memory mode. In this case, a user request to read or write from a contiguous area of virtual space might require I/O to a set of noncontiguous real shared-memory pages. The indirect buffer list would then describe the physical pages that map the contiguous virtual area. The start of such an area might fall on an arbitrary byte boundary.

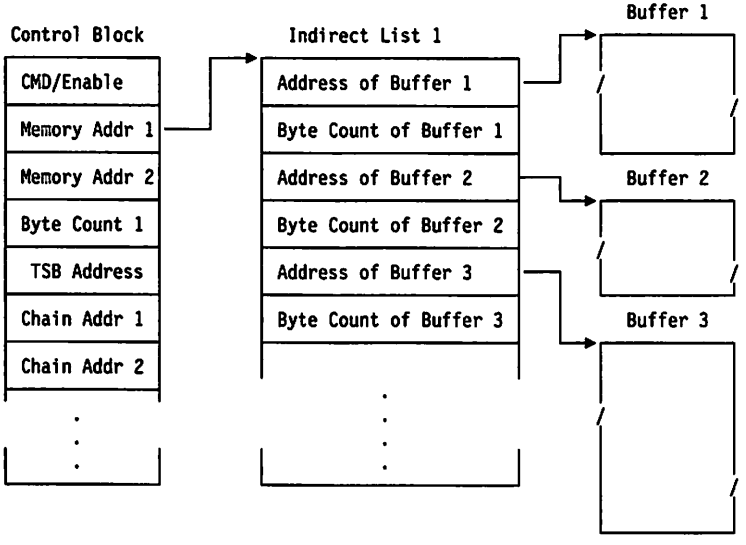


Figure 4-5. Control-Block Command with Indirect List

Reading with a Base Control Block and Indirect Lists

In a read operation, the control block is read from the system master by the device. If IL1 is equal to 1 and EXT is equal to 0, Memory Address x is read to determine the starting address of an indirect list in shared memory. An indirect list must start on a doubleword boundary. The length of the indirect list, in bytes, is found in Byte Count 1. The length of an indirect list must be a multiple of eight bytes.

An indirect list consists of individual buffer descriptors that are eight bytes long and contain a 4-byte buffer address, followed by a 4-byte buffer length. The buffer descriptors give the starting address and length of each element in the indirect list. The total number of data

bytes that can be transferred using an indirect list in a read command is equal to the sum of the buffer lengths of all the buffer descriptors in the indirect list. An implementation might choose to compute this total value and reject a command with a specification error if the total data-transfer size exceeds its capability. In this case, no data transfer occurs.

As data is transferred during the read operation, data is not allowed to exceed the length specified for an individual buffer descriptor. A buffer length of 0 transfers no bytes for that buffer descriptor. Additional source data from the device is transferred into the next buffer descriptor in the list if the end of the indirect list has not been reached and more source data exists.

A residual byte count can be developed by the implementation, initializing a value to the size of the buffer and subtracting 1 for each byte that is moved to the buffer from the device. Data movement to the buffer stops when the residual-byte-count value is 0. The buffer is said to be filled.

Data transfers continue in the read operation if any unfilled buffer descriptors remain in the indirect list and more data remains at the source. The indirect list is accessed by incrementing a value set to the contents of Memory Address x by 8 as each buffer descriptor is filled. Processing of the indirect list is completed when the number of list elements that have been filled equals the value in Byte Count 1 divided by 8. A residual buffer address can be developed by the implementation by initializing a value to the contents of Memory Address x and incrementing by 8 as each buffer descriptor is processed. See "Locate Mode Residual Data Values" on page 5-19 for details on developing the residual byte count and the residual buffer address.

If the operation terminates before successful completion, a program in the system master can read the residual byte count and residual buffer address in the corresponding termination status block to determine the progress of the operation and the recovery procedures. Storing of these values in the termination status block is optional (except for short- or long-length exception conditions) and is indicated by the conditional termination-status-block bit in Enable Word 1 of the command control block. If storing of these values is not supported, the program should assume that a complete retry of the

operation is needed. For details on this topic, see "Locate Mode Residual Data Values" on page 5-19.

When the device has more data to send to the program than there is space for in a buffer, a long-length exception condition exists. This error condition is always indicated in End Status Word 1. The error might be suppressed if the option is supported by the implementation. For a definition of this condition, see "Long-Length Exception" on page 5-11.

When the program defines buffer descriptors that require more data than the device has to send, a short-length exception condition exists. This error condition is always indicated in End Status Word 1. The error might be suppressed if the implementation supports this option. For a definition of this condition, see "Short-Length Exception" on page 5-9.

Apart from long-length and short-length exception conditions, transfer on a read might be terminated because of an irrecoverable error at the device. In this case, a major error is indicated in End Status Word 1, and an I/O interrupt is requested. Device-dependent error data can also be stored in the termination status block. For more detail on this topic, see "Exception Conditions" on page 5-9.

Reading with an Extended Control Block and Indirect Lists

This case is the same as the base control-block case, except that either Memory Address 1, Memory Address 2, or both can be specified as a combination of indirect lists or direct addressing, and incrementing or decrementing of shared-memory addresses. For further details, see the definition of the indirect-list-1 bit on 1-33 and the indirect-list-2 bit on 1-42.

Writing with a Base Control Block and Indirect Lists

In a write operation, the control block is sent from the system master to the device. The control block is read, and if the indirect-list-1 (IL1) bit is equal to 1 and Ext is equal to 0, Memory Address 2 is read to determine the starting address of an indirect list in shared memory. An indirect list must start on a doubleword boundary. The length of an indirect list must be a multiple of eight bytes. The length of the indirect list, in bytes, is found in Byte Count 1.

An indirect list consists of individual buffer descriptors that are eight bytes long and contain a 4-byte buffer address, followed by a 4-byte buffer length. The buffer descriptors give the starting address and length of each element in the indirect list. The total number of data bytes that can be transferred using an indirect list in a write command is equal to the sum of the buffer lengths of all the buffer descriptors in the indirect list. An implementation might choose to compute this total value and reject a command with a specification error if the total data-transfer size exceeds its capability. In this case, no data transfer occurs.

As data is transferred during the write operation, data is not accessed beyond the extent of the individual buffer descriptor. A buffer length of 0 transfers no bytes for that buffer descriptor.

As bytes are moved from an individual buffer element, a residual byte count can be developed by the implementation, initializing a value to the size of the buffer and subtracting 1 for each byte that is moved to the device from the buffer. Data movement from the buffer stops when the residual-byte-count value is 0. The buffer is said to be empty.

Data transfers continue in the write operation if any full buffer descriptors remain in the indirect list. The indirect list is accessed by incrementing a value set to the contents of Memory Address x by 8 as each buffer descriptor is emptied. Processing of the indirect list is completed when the number of list elements that have been emptied equals the value in Byte Count 1 divided by 8. A residual buffer address can be developed by the implementation by initializing a value to Memory Address x, and incrementing by 8 as each buffer descriptor is processed.

If the operation terminates before successful completion, a program in the system master can read the residual byte count and buffer address in the corresponding termination status block to determine the progress of the operation and to determine recovery procedures. Storing of these values in the termination status block is optional and is indicated by the termination-status-block bits in End Status Word 1. If storing of these values is not supported, the program should assume that a complete retransfer is needed. For more information on this topic, see "Locate Mode Residual Data Values" on page 5-19.

Transfer on a write might be terminated because of an irrecoverable error at the device. In this case, a major error is indicated in End Status Word 1, and an I/O interrupt is queued. Device-dependent error data can also be stored in the termination status block. For more information on this topic, see "Exception Conditions" on page 5-9.

Writing with an Extended Control Block and Indirect Lists

This case is the same as the base control-block case, except that either Memory Address 1, Memory Address 2, or both can be specified as a combination of indirect lists or direct addressing, and incrementing or decrementing of shared-memory addresses. For further details, see the definition of the indirect-list-1 bit on 1-33 and the indirect-list-2 bit on 1-42.

Writing Physical Data Blocks and Indirect Lists

The architecture does not require that a device write the elements of an indirect list as a single physical data block. However, if the device is capable of being written and then later read, the architecture does require that the same indirect list used on the write be capable of retrieving the same physical data blocks on the read.

Chapter 5. Design Considerations

This section contains a discussion of topics covering broad areas in Locate mode architecture. It addresses the following subjects:

- Locate mode setup and initialization
- Locate mode configuration
- Locate mode exception handling.

Locate Mode Setup and Initialization

This section contains some general architectural rules on the topic of setup and initialization. Many of these areas are implementation defined and are not part of the SCB architecture.

The following are used by Locate mode delivery and must be taken into consideration when the system and subsystem are configured:

- I/O address space

  During setup, the I/O base address space and its length are defined.

- System interrupt level
- System arbitration level
- Maximum number of devices that can be attached to the subsystem.

The specific control area definitions in I/O address space for Locate mode delivery are defined in "I/O Address Space" on page 1-15.

© Copyright IBM Corp. 1991

Locate Mode Configuration

This section contains information on the configuration of subsystems that support the Locate mode. Much of this area is implementation defined. The following are some of the architectural rules.

Locate Mode Ports

The following ports are required implementations of the Locate mode architecture:

- One Attention port
- One Command Interface port
- One Subsystem Control port
- One Command Busy/Status port
- One Interrupt Status port.

The size of the Attention and Interrupt Status ports depends on whether a Type 1 or Type 2 register interface is used. It is determined by the maximum number of devices supported by the subsystem. A given subsystem can use only the Type 1 or Type 2 register interface and only a single size for its Attention and Interrupt Status ports. This value must be determined before any commands are issued by programs.

Device Interrupt Identifier Port

The Device Interrupt Identifier ports are optional. If these ports are supported, the number of Device Interrupt Identifier ports required is determined by the following equation:

(1 + number of devices attached to the subsystem)/16

The definition of these ports is found in "Device Interrupt Identifier Port" on page 1-28.

Obtaining Configuration Information

The configuration information for a particular subsystem varies with the installation. The subsystem also has values that are important to the operation of programs. The Read Configuration command is used to obtain both configuration and device-dependent information and can be directed either to an individual device or to the subsystem. See page 1-63 for a complete description of the Read Configuration command.

The following is a list of examples of information returned by the Read Configuration command:

- The attachment ID of the subsystem
- The POS register values (Micro Channel systems)
- The revision level of the subsystem
- The interrupt and arbitration levels assigned to the subsystem
- The maximum number of devices that can be attached to the subsystem
- An indication of whether the subsystem supports the Device Interrupt Identifier port
- An indication of whether the device-dependent information is returned
- An indication of whether the subsystem needs to be initialized
- The I/O address of the Device Interrupt Identifier port
- The time it takes for a device to perform a reset
- The data block returned by the Run Diagnostics command
- The size and definition of the programs
- The time a subsystem takes to perform a reset
- The time a device takes to reset a control-block interrupt
- The time a device takes to reset the Interrupt Status port
- The minimum time that must elapse between the submission of a command and the reading of the Command Busy/Status port for acceptance data.

Exception and Error Handling

The Locate mode defines exception and error conditions and provides a standardized method of reporting these conditions. The following topics are discussed:

- Exception and error handling
- Exceptions and error conditions
- The actions to take when a condition is detected
- The data reported in a status block for a condition
- A summary of exception and error conditions that can occur.

Errors and exceptions exist at two levels in the Locate mode: the device level and the subsystem or command-delivery level. Many of the details of error handling and recovery are device dependent; however, some general guidelines are described in the following.

Device Level

At the device level, most exceptions or errors are indicated by an interrupt to the system master. The treatment and handling of Locate mode signalling is described in detail in "Signalling Protocol" on page 3-13.

To process an exception, an error handling program determines which command caused the exception and reads the associated termination status block for status. This topic is discussed in "System Master – Physical Interrupts" on page 3-31. A discussion of how to process residual status is found in "Locate Mode Residual Data Values" on page 5-19.

At the device level, errors and exceptions are generally handled in the following manner:

1. Determine which device caused the exception.

   This information is found either in the Interrupt Status port or in the Device Interrupt Identifier port (DIIP) for the subsystem.

2. Determine the type of exception.

   The type of the exception is found in the Interrupt Status port interrupt code or in a nonzero DIIP bit. If it was a control-block command, the termination status block might need to be examined to determine the exact cause of the exception.

3. Determine which command caused the exception.

4. Read the termination status block for ending status.

   If the interrupt code was a control-block command completed with error, read the termination status block to get the details of the exception.

When the above steps are complete, proceed as follows:

- Short-length exception (suppressed).

  Determine the number of data bytes successfully read, using residual status. Do not treat it as an error.

  Note: To determine the number of bytes successfully transferred in a control-block command, refer to "Number of Bytes Transferred" on page 5-21.

- Short-length exception (not suppressed).

  Determine the number of data bytes successfully read, using residual status. Report any error.

- Specification exception.

  Use any implementation-defined data, found in the termination status block of the command, to find the reason for the error. Report any error.

- Long-length exception (suppressed).

  Determine the number of data bytes successfully read, using residual status. Do not treat it as an error.

- Long-length exception (not suppressed).

  Determine the number of data bytes successfully read, using residual status. Report any error.

- Device overrun on write.

  Determine the number of data bytes successfully written if residual status is present. Report any error.

- Major error on data transfer.

  Determine the number of data bytes successfully transferred if residual status is present. Report any error.

- Hardware failure—control-block command.

  It might not be possible to find the control-block command that failed. Report any error.

- Hardware failure—non-control-block command.

  Retry the failed command, or report an error if the device is not functional.

- Non-control-block command completed with error.

  Determine which command failed. Report any error.

- Device does not respond after maximum time.

  Retry the failed command, or report an error if the device is not functional.

Subsystem Level

At the subsystem level, exceptions or errors are noted by signalling an interrupt to the system master or by setting the reject and status bits in the Command Busy/Status port.

The treatment and handling of Locate mode signalling is described in "Signalling Protocol" on page 3-13.

At the subsystem level, errors and exceptions are generally handled as follows:

- Control-block command errors.

  These are handled the same way as in the device or entity case. See "Device Level" on page 5-4.

- The subsystem does not respond after the maximum time.

  Determine which command is in progress:

  - Hardware-controlled Reset Subsystem command.

    When the command is a hardware-controlled Reset Subsystem command, report any error, retry the command, and take the subsystem out of service if the retry is unsuccessful.

  - Software-controlled Reset Subsystem command.

    When the command is a software-controlled Reset Subsystem command, retry the command and issue a hardware-controlled Reset Subsystem command if the retry is unsuccessful.

  - The subsystem is taking too much time to clear the busy bit on command submission.

    Read the Command and Attention ports. Issue a software-controlled Reset Subsystem command, and attempt to resubmit the command.

- Command rejected in Command Busy/Status port.

  The status bits in the Command Busy/Status port are read, and processing proceeds as follows:

  - Device unavailable.

    Determine which command failed. Clear the rejection condition from the subsystem. Retry the command, or report any error.

  - Invalid command.

    Determine which command failed. Clear the rejection condition from the subsystem. Report any error.

  - Device busy.

    Determine which command failed. Clear the rejection condition from the subsystem. Report any error, or wait a specified amount of time and retry the command.

- Device control-block execution suspended.

  Determine which command failed. Clear the rejection condition from the subsystem. Report any error. The caller can reenable control-block execution with a Resume command if the suspension is the result of a Suspend command.

- Invalid device number.

  Determine which command failed. Clear the rejection condition from the subsystem. Report any error.

- Device limits reached.

  Determine which command failed. Clear the rejection condition from the subsystem. Report any error.

  Note: The caller needs to process and clear interrupts for the device with the queue full by ensuring that the device is enabled for interrupts, using the Reset Interrupt Status Port command. This is necessary for the interrupt-handling software to read the Interrupt Status port and issue a Reset Interrupt Status Port command for the device indicated, if the interrupt-valid (IV) bit of the Command Busy/Status port is set to 1.

Exception Conditions

There are five types of exception conditions in the Locate mode:

- Short-length exception
- Long-length exception
- Specification exception
- Device-overrun exception
- Major error on data transfer.

Each is described in this section.

Short-Length Exception

- Definition of a short-length exception.

  A short-length exception can occur when data is being read during the execution of a control-block command. The condition exists when the shared data area specified in the control block (the target buffer area) requests more data than is available from the source device.

  The condition can arise in any of the control-block commands that use the semantics of the control-block read commands to transfer data from a device to an area in shared memory. The condition is not restricted to the control-block Read command; it can be detected in the following control-block commands:

  - Read
  - Read Configuration
  - Read Completion Status
  - Run Diagnostic Test.

- Standard actions on detection.

  When the condition is detected, the following actions are always taken:

  1. The residual byte count is stored.

     The residual byte count is stored in the termination status block. The stored value is the number of bytes that remain to be transferred for the current buffer. See "Residual Byte Count" on page 5-19.

2. The residual buffer address is stored.

   The residual buffer address is stored in the termination status block. The stored value is the address of the buffer element being used in the current transfer. See "Residual Buffer Address" on page 5-20.

3. The short-length-exception bit in End Status Word 1 is set to 1.

4. The TSB-available bits in End Status Word 1 are set to show that residual status is stored in the termination status block.

- Optional actions on detection.

  When the condition is detected, the following optional actions can be taken:

  1. Device-dependent status is stored.

     An implementation can choose to store device-dependent data in the termination status block when the condition is detected. If this is done, the TSB-available bits in End Status Word 1 are set to reflect this.

  2. Exception suppressed.

     If the exception has been suppressed in the current control block by setting the suppress-exception-short bit in Enable Word 1 to 1, the condition is not treated as a major exception. An interrupt is not requested for this control block unless the disable-interrupt bit in Enable Word 1 is set to 0. If the disable-interrupt bit is set to 0, the interrupt-requested bit in End Status Word 1 is set to 1, the done bit in the termination status block is set to 0, and an interrupt is requested for the condition.

     A command chain that is defined to continue on No Error (by setting the chain-no-error bit in Enable Word 1 to 1) is not terminated if the error is suppressed. If conditional chaining was specified, the conditional chain can continue if the ending condition matches the conditional specification.

3. Exception not suppressed.

   If the exception has not been suppressed in the current control block by setting the suppress-exception-short bit in Enable Word 1 to 0, the condition is treated as a major exception. The major-error and interrupt-requested bits in End Status Word 1 are set to 1, the done bit is set to 0, and an interrupt is requested for the condition.

   A command chain that was defined to continue on No Error (by setting the chain-no-error bit in Enable Word 1 to 1) cannot continue. If conditional chaining is enabled in the current control block, with the conditional-chaining bit set to 1, the conditional chain can continue if the condition matches the ending status.

Long-Length Exception

- Definition of long-length exception.

  A long-length exception can occur when data is being read during the execution of a control-block command. The condition exists when the shared-data area specified in the control block (the target buffer area) requests less data than is available from the source device.

  The condition can arise in any of the control-block commands that use the semantics of the Read command to transfer data from a device to an area in shared memory. It is not restricted to the control-block Read command, and it can be detected in the following control-block commands:

  - Read
  - Read Configuration
  - Read Completion Status
  - Run Diagnostic Test.

- Standard actions on detection.

  When the condition is detected, the following actions are always taken:

  1. The residual byte count is stored.

     The residual byte count is stored in the termination status block. The stored value is 0, because all bytes specified in the current buffer have been transferred (there is no residual byte count). See "Residual Byte Count" on page 5-19.

  2. The residual buffer address is stored.

     The residual buffer address is stored in the termination status block. The stored value is the address of the buffer element being used in the current transfer. In this case, it is the last buffer in an indirect list or the only buffer, when an indirect list is not used. See "Residual Buffer Address" on page 5-20.

  3. The long-length-exception bit in End Status Word 1 is set to 1.

  4. The TSB-available bits in End Status Word 1 are set to show that residual status is stored in the termination status block.

- Optional actions on detection.

  When the condition is detected, the following optional actions can be taken:

  1. Device-dependent status is stored.

     An implementation can choose to store device-dependent data in the termination status block when the condition is detected. If this is done, the TSB-available bits in End Status Word 1 are set to reflect this.

  2. The exception is suppressed.

     If the exception has been suppressed in the current control block and the suppress-exception-long bit in Enable Word 1 is set to 1, the condition is not treated as a major exception. An interrupt is not requested for this control block unless the disable-interrupt bit in Enable Word 1 is set to 0. If the disable-interrupt bit is set to 0, the interrupt-requested bit in End Status Word 1 is set to 1, the done bit is set to 0, and an interrupt is requested for the condition.

A command chain that is defined to continue on No Error by setting the chain-no-error bit in Enable Word 1 to 1 is not terminated when this exception is suppressed. If conditional chaining is specified, the command chain can continue if the ending condition matches the conditional specification.

3. The exception is not suppressed.

   If the exception has not been suppressed in the current control block and SEL in Enable Word 1 is set to 0, the condition is treated as a major exception. The major-error and interrupt-requested bits in End Status Word 1 are set to 1, the done bit is set to 0, and an interrupt is requested for the condition.

   A command chain that is defined to continue on No Error by setting the chain-no-error bit in Enable Word 1 to 1 cannot continue. If conditional chaining is enabled in the current control block by setting the conditional-chaining bit to 1, the conditional chain can continue if the condition matches the ending status.

Specification Exception

- Definition of specification exception.

  A specification exception can occur during the reading of any control-block command in the Locate mode. The condition occurs when an illegal or unsupported value or option is detected in a control-block field. Specification testing is optional in an implementation. The condition can be detected in any command control blocks.

- Standard actions on detection.

  When the condition is detected, the following actions are always taken:

  1. The command is terminated, and no data transfer occurs.
  2. If additional status is to be provided, it is stored in the termination status block. This is described in the list of optional actions for this condition.

3. This condition is treated as a major error. The specification-check bit in End Status Word 1 is set to 1, indicating that the condition has been detected. The major-error/exception bit in the termination status block in End Status Word 1 is also set to 1, the done bit is set to 0, and an interrupt is requested for the condition.

   Any control-block chaining that is in effect when the condition is detected is halted. Conditional chaining is not supported when a specification error is detected.

- Optional actions on detection.

  When the condition is detected, the following optional actions can be taken:

  – Device-dependent data is optionally stored.

    An implementation-defined value can be written to the termination status block in the device-dependent area to identify the field or option that has been found to be illegal.

  – The TSB-available bits are set to show device-dependent data.

    If device-dependent data is stored in the termination status block, the TSB-available bits in End Status Word 1 are set to reflect this.

Device-Overrun Exception

- Definition of device-overrun exception.

  A device-overrun exception can occur when data is being read or written during the execution of a control-block command. The condition exists when data is being transferred at a rate that the device cannot accept. When the condition is detected, data transfer to the device is halted.

  The condition can arise in any of the control-block commands that transfer data to or from an area in shared memory.

- Standard actions on detection.

  When the condition is detected, the following actions are always taken:

  1. Data transfer for the control-block command requesting the data transfer is terminated.
  2. If additional status is to be provided, it is stored in the termination status block. This is described in the list of optional actions for this condition.
  3. The condition is treated as a major error. The device-overrun bit in End Status Word 1 is set to 1, indicating that the condition has been detected. The major-error/exception bit in the termination status block in End Status Word 1 is also set to 1, the done bit is set to 0, and an interrupt is requested for the condition.

     A control-block command chain that is defined to continue on No Error by setting the chain-no-error bit in Enable Word 1 to 1 cannot continue. If conditional chaining is enabled in the current command control block by setting the conditional-chaining bit to 1, the conditional chain can continue if the condition matches the ending status.

- Optional actions on detection.

  When the condition is detected, the following optional actions can be taken:

  1. The residual byte count is optionally stored.

     The residual byte count is optionally stored in the termination status block. The stored value is the number of bytes remaining to be transferred for the current buffer. See "Residual Byte Count" on page 5-19.

  2. The residual buffer address is optionally stored.

     The residual buffer address is stored in the termination status block. The stored value is the address of the buffer element being used in the current transfer. See "Residual Buffer Address" on page 5-20.

3. The TSB-available bits are set to show that residual status was stored.

   The TSB-available bits in End Status Word 1 are set to show that residual status is stored in the termination status block, if needed.

4. Device-dependent status is optionally stored.

   An implementation can optionally store device-dependent data in the termination status block when the condition is detected.

5. The TSB-available bits are set to show device-dependent data.

   If device-dependent data is stored in the termination status block, the TSB-available bits in End Status Word 1 are set to reflect this.

Major Error on Data Transfer

- Definition of major error on data transfer.

  A major error on data transfer can occur when data is being read or written during the execution of a control-block command. The condition exists when the data requested cannot be provided because an error has arisen that cannot be corrected. Note that this condition differs from errors for device-overrun, long-length, and short-length exceptions. This condition also sets the major-error/exception bit in End Status Word 1. The condition arises in the following ways:

  - Data is being read.

    In this case, the data at the device cannot be delivered to the shared-memory buffer area defined in the control block because an error has arisen that cannot be corrected. This error cannot be a long-length or short-length exception.

  - Data is being written.

    In this case, the data from the shared-memory buffer area defined in the control block cannot be delivered to the device because an error has arisen that cannot be corrected. This error cannot be a device-overrun exception.

The condition can arise in any control-block command that transfers data to or from shared memory.

- Standard actions on detection.

  When the condition is detected, the following actions are always taken:

  1. Data transfer for the control-block command requesting the read or write is terminated.
  2. If additional status is to be provided, it is stored in the termination status block. This is described in the list of optional actions for this condition.
  3. An error condition that cannot be corrected is treated as a major error. The major-error/exception bit in the termination status block in End Status Word 1 is set to 1, indicating that the condition has been detected. The interrupt-requested bit in End Status Word 1 is set to 1, the done bit is set to 0, and an interrupt is requested for the condition.

     A control-block command chain that is defined to continue on No Error by setting the chain-no-error bit in Enable Word 1 to 1 cannot continue. If conditional chaining is enabled in the current command control block by setting the conditional-chaining bit to 1, the conditional chain can continue if the condition matches the ending status.

- Optional actions on detection.

  When the condition is detected, the following optional actions can be taken:

  1. The residual byte count is optionally stored.

     The residual byte count is optionally stored in the termination status block. The stored value is the number of bytes that remain to be transferred for the current buffer. See "Residual Byte Count" on page 5-19.

  2. The residual buffer address is optionally stored.

     The residual buffer address is stored in the termination status block. The stored value is the address of the buffer element being used in the current transfer. See "Residual Buffer Address" on page 5-20.

3. The TSB-available bits are set to show that residual status is stored.

   The TSB-available bits in End Status Word 1 are set to show that residual status is stored in the termination status block.

4. Device-dependent status is optionally stored.

   An implementation can optionally store device-dependent data in the termination status block when the condition is detected.

5. The TSB-available bits are set to show device-dependent data.

   If device-dependent data is stored in the termination status block, the TSB-available bits in End Status Word 1 are set to reflect this.

Locate Mode Residual Data Values

The termination status block for a control-block command contains fields that are designed to show the progress a control-block command has made in data transfer when it has been completed. These fields are the Residual Byte Count and Residual Buffer Address fields. Further definition and programmed use of these fields are discussed in the following section.

A summary of the residual data provided, organized by control-block-command ending status, is in "Summary—Residual Buffer Address and Residual Byte Count" on page 5-22.

Residual Byte Count

The termination status block contains the Residual Byte Count field that is used to indicate the number of bytes that remain to be transferred when the command is completed.

The use of the Residual Byte Count field is completely defined by the architecture when a single storage operand is specified in the control block. The use of this field is implementation defined when two shared-memory operands are specified in the same control block.

When the Residual Byte Count field is 0 and a long-length exception has not been indicated for the command, all data for the command has been successfully read or written. When a long-length exception exists, this field is 0, but additional data remains at the device. An implementation can provide the extent of this unread data in a device-dependent field in the termination status block.

When the Residual Byte Count field is not 0, it indicates that the operation has terminated; therefore, the data that was defined by the command could not be transferred. In this case, a residual byte count remains. To determine the number of bytes of data that have been successfully transferred by the command, a program in the system master needs to read the associated control block and determine the expected byte count of the transfer. To do this, a program might need to use the second data value provided in the termination status block, the residual buffer address.

Residual Buffer Address

The termination status block contains the Residual Buffer Address field, which is used to indicate the buffer that was in use when the command was completed.

The use of the Residual Buffer Address field is completely defined by the architecture when a single storage operand is specified in the control block. The use of this field is implementation defined when two shared-memory operands are specified in the same control block.

When indirect lists are not being used, the address returned for this value corresponds to the buffer field specified in the control block. The address that is presented corresponds to either the address of Memory Address 1 or the address of Memory Address 2. The value is determined by the setting of the memory-address-1-selected (MA1) bit or the memory-address-2-selected (MA2) bit in Enable Word 1.

For example, if the control block is located at Memory Address hex 40000 and MA2 is selected, the residual buffer address is hex 40008 (the base control-block address + hex 8).

If indirect lists are being used, the address returned is the physical address of the list element in use when the command was completed.

For example, assume that an indirect list is being used, it is located at Memory Address hex 60000, and it contains 4 buffer elements (is 32 bytes long). The list starts at shared-memory location hex 60000 and ends at location hex 6001F. In this case, the residual buffer address returned for a control block using this indirect list must be one of the following:

- Hex 60000 – the first buffer descriptor in the indirect list
- Hex 60008 – the second buffer descriptor in the indirect list
- Hex 60010 – the third buffer descriptor in the indirect list
- Hex 60018 – the last buffer descriptor in the indirect list.

Number of Bytes Transferred

To determine the number of bytes that have been successfully read or written by a control-block command, the residual byte count can be subtracted from the number of bytes requested in the command. This computation is of most interest when a short-length exception is detected. It defines the number of bytes that were successfully read. The computation involves the following cases:

- Control block in base format—no indirect list specified.

  The total data-transfer size is given by the contents of Byte Count 1. The number of bytes successfully moved is computed as:

  Bytes moved = Byte Count 1 - residual byte count

- Control block in extended format—no indirect list specified (only one memory address specified).

  The total data-transfer size is given by the contents of Byte Count 1 if Memory Address 1 is 1, or by the contents of Byte Count 2 if Memory Address 2 is 1. The Residual Byte Count field selected in the termination status block is Residual Byte Count 1 if Memory Address 1 is 1, or Residual Byte Count 2 if Memory Address 2 is 1. The number of bytes successfully moved is computed as:

  Bytes moved = total data-transfer size - residual byte count

- Control block in extended format—no indirect list specified (two memory addresses specified).

  The total data-transfer size is not defined by the architecture in this case. It is determined by the control-block operation code. An example of this is a case in which data is defined to be transferred from Memory Address 1 to Memory Address 2. If this convention were used, Memory Address 2 would be used to determine the number of data bytes read.

- Indirect list specified—only one memory address specified.

  The total data-transfer size is given by the byte counts of the indirect-list elements up to and including the element where the transfer stopped. From this total the residual byte count given in the termination status block needs to be subtracted.

- Indirect list specified – two memory addresses specified.

  The total data-transfer size is not defined by the architecture in this case. It is determined by the control-block operation code. An example of this is a case in which data is defined to be transferred from Memory Address 1 to Memory Address 2. If this convention were used, Memory Address 2 would be used to determine the number of data bytes read.

Summary – Residual Buffer Address and Residual Byte Count

This section provides a summary of the contents of the Residual Buffer Address and Residual Byte Count fields for the various ending conditions of control-block commands. The Residual Buffer Address and Residual Byte Count fields must be stored in the termination status block when a short-length or long-length exception condition is encountered. For all other errors and exceptions, storing of these fields in the termination status block is optional.

In the summary data in Figure 5-1, the terms "read" and "write" refer to any control-block operations using the semantics of the Read and Write control-block commands to transfer data to or from shared memory. For example, the term "read" refers not only to the Read command, but to other commands such as Read Configuration; the term "write" refers not only to the Write command, but to other commands such as Initialize Device.

Control Block Completion	Residual Buffer Address	Residual Byte Count
Normal end on read	Memory Address 2 in control block if no indirect list. If indirect list, address of last list element.	Zero.
Short-length exception on read	Memory Address 2 in control block if no indirect list. If indirect list, address of last list element.	Number of bytes remaining to transmit for current buffer element.
Long-length exception on read	Memory Address 2 in control block if no indirect list. If indirect list, locates last list element.	Zero.
Device error on read	Memory Address 2 in control block if no indirect list. If indirect list, address of current list element.	The number of bytes that remain to be read in the current buffer element.
Specification error in control block	Not set.	Not set.
Normal end on write	Memory Address 2 in control block if no indirect list. If indirect list, address of last list element.	Zero.
Device error on write, or device overrun	Memory Address 2 in control block if no indirect list. If indirect list, address of current list element.	The number of bytes that remain to be written in the current buffer element.
Normal end on no data transfer (Noop)	Not set.	Not set.

Figure 5-1. Residual Status Data after Control-Block Execution

Notes:

Index

A

address field 1-53
address space 1-15
architected bit 1-32
attachment ID field 1-65
attachment revision level field 1-66
attention code field 1-19
attention port 1-9, 1-18, 1-48, 2-3, 5-2
  attention code field 1-19
  device number field 1-20

B

base control block 4-12, 4-14
base control-block structure 1-10, 1-29
base termination status block 1-44
bits
  architected (A) 1-32
  busy (B) 1-9, 1-27, 1-91
  chain-condition-specification (CCS) 1-37
  chain-direction (CD) 1-47
  chain-no-error (CNE) 1-38
  chain-specification 4-7
  command-type (CT) 1-50
  complete (C) 1-57
  conditional termination-status-block (TSB) 1-33
  conditional-chaining (CC) 1-38
  decrement-memory-address-1 (DEC1) 1-42
  decrement-memory-address-2 (DEC2) 1-42
  device-dependent (DD) 1-26, 1-33, 1-35, 1-49
bits *(continued)*
  device-dependent-data-available 1-67
  device-not-initialized (INI) 1-48, 1-57
  device-overrun (DO) 1-48
  disable-command-interrupt (DCI) 1-73, 1-76, 1-78, 1-80, 1-81, 1-84, 1-86, 1-87, 1-89
  disable-device-interrupt (DDI) 1-73, 1-76, 1-78, 1-80, 1-81, 1-84, 1-86, 1-87, 1-89
  disable-interrupt (DI) 1-35
  done (D) 1-50
  enable-DMA (DMA) 1-21
  enable-subsystem-interrupts (EI) 1-21
  extended-status (ES) 1-47
  extended-structure (EXT) 1-35, 1-37
  format-identifier 1-72, 1-74
  indirect-list-1 (IL1) 1-33, 1-37, 1-59
  indirect-list-2 (IL2) 1-42
  interrupt-requested (INT) 1-49, 1-60, 1-72
  interrupt-support 1-68
  interrupt-valid (IV) 1-9, 1-27, 1-91
  load 1-68
  location (L) 1-57
  long-length-exception (LLE) 1-49
  major-error/exception (ME) 1-47, 1-60, 1-72
  memory-address-1-selected (MA1) 1-36, 1-37
  memory-address-2-selected (MA2) 1-35, 1-37
  microseconds (u) 1-67
  milliseconds (m) 1-67

© Copyright IBM Corp. 1991

bits *(continued)*
nanoseconds (n) 1-67
operation-code 1-75
option-1 (OP1) 1-78, 1-84
option-2 (OP2) 1-78, 1-84
reject (REJ) 1-9, 1-26, 1-91
reset (RST) 1-68, 1-90
reset-reject (RR) 1-20
restart (R) 1-57
seconds (s) 1-67
short-length-exception (SLE) 1-49
specification-check (SC) 1-49
status 1-9, 1-25
subsystem-reset (RST) 1-20
suppress-exception-long (SEL) 1-34
suppress-exception-short (SES) 1-34
suspended (SUS) 1-47
TSB-available (TSA) 1-48
busy bit 1-9, 1-27, 1-91
byte count 1 field 1-40, 1-56, 1-59, 1-61, 1-64, 1-71
byte count 2 field 1-42

C

chain address 1 field 1-40
chain address 2 field 1-41, 1-47
chain ID field 1-41
chain no error (unconditional chain) 4-7
chain-condition-specification bit 1-37, 4-7
chain-direction bit 1-47
chain-no-error bit 1-38
chaining 4-1
characteristics, command control blocks 1-30
command busy/status port 1-9, 1-24, 2-3, 5-2
command busy/status port indicators 1-91
command busy/status port indicators *(continued)*
busy 1-9, 1-91
interrupt-valid 1-9, 1-91
reject 1-9, 1-91
status 1-9
command chaining 4-1, 4-2
command control block 1-29
command delivery, hardware-controlled 3-4
command delivery, software-controlled 3-4
command for completion status field 1-61
command interface port 5-2
command port 1-9, 1-18, 2-3
command reject status 1-25
command rejected in command/busy status port 5-7
command word field 1-31
command-chaining options 4-7
chain no error 4-7
conditional chaining 4-7
no chaining 4-7
two-way chaining 4-8
command-control-block delivery 3-6
command-delivery protocol 3-10
command-type bit 1-50
commands
end of interrupt 3-20, 3-22, 3-34
initialize device 1-55
no operation 1-57, 1-76
read 1-58
read completion status 1-53, 1-60
read configuration 1-63, 5-3
reset control block interrupt 1-80
reset device 1-77
reset interrupt status port 1-79
reset reject 1-9
reset subsystem (hardware controlled) 1-85, 1-90

commands *(continued)*
reset subsystem (software controlled) 1-82
resume 1-86
run diagnostic test 1-70, 1-87
run immediate diagnostic test 1-87
suspend 1-47, 1-88
write 1-71
complete bit 1-57
conditional chaining 4-7
conditional command chaining 4-5
conditional termination-status-block bit 1-33
conditional-chaining bit 1-38
configuration 5-2
configuration information 5-3
control areas 1-9
See also ports
control block 1-29
control-block chaining 4-1
control-block chains, searching 3-23
control-block characteristics 1-30
control-block command delivery 3-6
control-block command error 5-6
control-block commands 1-55
initialize device 1-55
no operation 1-57
read 1-58
read completion status 1-60
read configuration 1-63, 5-3
run diagnostic test 1-70, 1-87
write 1-71
control-block delivery 1-1
control-block delivery service 3-9
control-block delivery service parameters 3-9
control-block execution 4-9
control-block execution suspended 5-8
control-block interrupt 3-17, 3-23, 3-25, 3-26, 3-28, 3-34
control-block structure 1-10
base control-block structure 1-10, 1-29
byte count 1 field 1-40, 1-59, 1-61, 1-64, 1-71
byte count 2 field 1-42
chain address 1 field 1-40
chain address 2 field 1-41, 1-47
chain ID field 1-41
command word field 1-31
device dependent size field 1-41
enable word 1 field 1-33, 1-59
enable word 2 field 1-41
extended control-block structure 1-10, 1-29
memory address 1 field 1-39
memory address 2 field 1-39, 1-59, 1-71
termination status block address field 1-40
count field 1-53, 1-81

D

data chaining 4-1, 4-10
data delivery 2-9
decrement-memory-address-1 bit 1-42
decrement-memory-address-2 bit 1-42
delivery flow 1-6
delivery level 3-1
delivery service 1-1
delivery structure 1-13
delivery-level protocols 3-10
command delivery 3-10
signalling 3-13
delivery-level services 3-7
control-block delivery 3-9
hardware-controlled command delivery 3-8

delivery-level services *(continued)*
immediate-command
delivery 3-7
signalling protocol 3-13
control-block chains, searching 3-23
control-block interrupt 3-17, 3-23, 3-25, 3-26, 3-28, 3-34
DIIP interrupt 3-26, 3-35, 3-37
first-level interrupt-handling routine 3-20
interrupt identification 3-14
interrupt queuing 3-16
logical interrupt 3-15
non-control-block interrupt 3-19, 3-21, 3-22, 3-30, 3-34
non-DIIP interrupt 3-16, 3-37
physical interrupt 3-14, 3-19, 3-20, 3-30, 3-31
physical-interrupt enablement 3-15
resetting interrupts 3-38
second-level interrupt-handling routine 3-21
delivery-level structure 3-1
control-block command delivery 3-6
immediate-command delivery 3-4
design considerations 5-1
configuration information 5-3
device-overrun exception 5-14
exception and error handling 5-4
command rejected in command/busy status port 5-7
control-block command error 5-6
control-block execution suspended 5-8
device busy 5-7
device does not respond 5-6
design considerations *(continued)*
exception and error handling *(continued)*
device limits reached 5-8
device overrun on write 5-5, 5-6
device unavailable 5-7
hardware failure – control-block command 5-6
hardware failure – non-control-block command 5-6
invalid command 5-7
invalid device number 5-8
long-length exception 5-5
major error on data transfer 5-6
short-length exception 5-5
specification exception 5-5
subsystem does not respond 5-7
exception conditions 5-9
long-length exception 5-11
major error on data transfer 5-16
short-length exception 5-9
specification exception 5-13
device busy 5-7
device dependent area size field 1-69
device dependent configuration data area field 1-70
device dependent data area field 1-53
device dependent data size field 1-53
device dependent size field 1-41
device does not respond 5-6
device interrupt identifier port 1-10, 1-28, 2-3, 3-39, 5-2
device limits reached 5-8

device number 1-20
device number field 1-20, 1-24
device number subfield 3-6
device overrun on write 5-5, 5-6
device unavailable 5-7
device-dependent area 1-10
device-dependent bit 1-26, 1-33, 1-35, 1-49
device-dependent-data-available bit 1-67
device-not-initialized bit 1-48, 1-57
device-overrun bit 1-48
device-overrun exception 5-14
diagnostic command field 1-87
DIIP
  See device interrupt identifier port
DIIP address field 1-69
DIIP bit position field 1-69
DIIP interrupt 3-26, 3-35, 3-37
disable-command-interrupt bit 1-73, 1-76, 1-78, 1-80, 1-81, 1-84, 1-86, 1-87, 1-89
disable-device-interrupt bit 1-73, 1-76, 1-78, 1-80, 1-81, 1-84, 1-86, 1-87, 1-89
disable-interrupt bit 1-35
done bit 1-50

E

enable word 1 field 1-33, 1-59
enable word 2 field 1-41
enable-DMA bit 1-21
enable-subsystem-interrupts bit 1-21
end of interrupt command 3-20, 3-22, 3-34
end status word 1 field 1-44, 1-46, 1-57, 1-60, 1-72
end status word 2 field 1-47, 1-50
error handling 5-4
  command rejected in command/busy status port 5-7
error handling *(continued)*
  control-block command error 5-6
  control-block execution suspended 5-8
  device busy 5-7
  device does not respond 5-6
  device limits reached 5-8
  device overrun on write 5-6
  device unavailable 5-7
  hardware failure – control-block command 5-6
  hardware failure – non-control-block command 5-6
  invalid command 5-7
  invalid device number 5-8
  subsystem does not respond 5-7
exception handling 5-4
  device overrun on write 5-5
  device-overrun exception 5-15
  long-length exception 5-5, 5-12
  major error on data transfer 5-6, 5-17
  short-length exception 5-5, 5-9
  specification exception 5-5, 5-13
exceptions 5-9
extended control block 4-14, 4-16
extended control-block structure 1-10, 1-29
extended termination status block 1-45
extended-status bit 1-47
extended-structure bit 1-35, 1-37

F

fields
  address 1-53
  attachment ID 1-65
  attachment revision level 1-66
  attention code 1-19

fields *(continued)*
byte count 1 1-40, 1-56, 1-59, 1-61, 1-64, 1-71
byte count 2 1-42
chain address 1 1-40
chain address 2 1-41, 1-47
chain ID 1-41
command for completion status 1-61
command word 1-31
count 1-53, 1-81
device dependent area size 1-69
device dependent configuration data area 1-70
device dependent data area 1-53
device dependent data size 1-53
device dependent size 1-41
device number 1-20, 1-24, 3-6
diagnostic command 1-87
DIIP address 1-69
DIIP bit position 1-69
enable word 1 1-33, 1-59
enable word 2 1-41
end status word 1 1-44, 1-46, 1-57, 1-60, 1-72
end status word 2 1-47, 1-50
flags 1-67
interrupt identifier 1-22
interrupt level 1-66
maximum command/busy status time 1-69
maximum device reset time 1-67
maximum number of queued interrupts 1-69
maximum reset control block interrupt time 1-68
maximum reset interrupt status port time 1-68
maximum size for completion status 1-69
fields *(continued)*
maximum size for diagnostic status 1-69
maximum subsystem reset time 1-66
memory address 1 1-39
memory address 2 1-39, 1-56, 1-59, 1-71
number of devices supported 1-66
op code 1-32, 1-74
op code dependent 1-73
operation code 1-32, 1-74
options 1-32
POS register 2 1-65
POS register 3 1-65
POS register 4 1-65
POS register 5 1-65
POS register 6 1-66
POS register 7 1-66
residual buffer address 1-59, 5-20, 5-22
residual buffer address 1 1-51
residual buffer address 2 1-44, 1-52
residual byte count 1-59, 5-19, 5-22
residual byte count 1 1-50
residual byte count 2 1-44, 1-51
start address 1-57
termination status block address 1-40, 1-43
first-level interrupt-handling routine 3-20
flags field 1-67
format-identifier bit 1-72, 1-74

H

hardware failure — control-block command 5-6
hardware failure — non-control-block command 5-6

hardware support, subsystem 2-3
hardware support, system master 2-3
hardware-controlled command delivery 3-4
hardware-controlled command delivery service 3-8
hardware-controlled command delivery service parameters 3-8
hardware-controlled reset subsystem command
See reset subsystem command (hardware controlled)

I

immediate commands 1-72
no operation 1-76
reset control block interrupt 1-80
reset device 1-77
reset interrupt status port 1-79
reset subsystem (hardware controlled) 1-85, 1-90
reset subsystem (software controlled) 1-82
resume 1-86
run immediate diagnostic test 1-87
suspend 1-88
immediate-command delivery 3-4
immediate-command delivery service 3-7
immediate-command delivery service parameters 3-7
immediate-command structure 1-73
disable-command-interrupt bit 1-73
disable-device-interrupt bit 1-73
format-identifier bit 1-74
op code dependent field 1-73
operation code field 1-74
operation-code bits (format 1) 1-75
indirect list 4-10, 4-12, 4-14, 4-16
See also processing level, data chaining
indirect-list structure 1-12, 1-53
address field 1-53
count field 1-53
indirect-list-1 bit 1-33, 1-37, 1-59
indirect-list-2 bit 1-42
initialization 5-1
initialize device command 1-55
options 1-57
interrupt 2-8
interrupt identification 3-14
interrupt identifier field 1-22
interrupt level field 1-66
interrupt queuing 3-16
interrupt status port 1-9, 1-21, 1-79, 2-3, 3-38, 5-2
device number field 1-24
interrupt identifier field 1-22
interrupt-identifier codes 1-22
interrupt-requested bit 1-49, 1-60, 1-72
interrupt-support bit 1-68
interrupt-valid bit 1-9, 1-27, 1-91
interrupts, resetting 3-38
invalid command 5-7
invalid device number 5-8
I/O address space 1-15, 2-4
I/O space 1-15

L

load bit 1-68
locate mode overview 1-1
location bit 1-57
logical interrupt 3-15
long-length exception 4-14, 5-5, 5-11
long-length-exception bit 1-49

M

major error on data transfer 5-6, 5-16
major-error bit 1-47
major-error/exception bit 1-47, 1-60, 1-72
major-exception bit 1-47
maximum command/busy status time field 1-69
maximum device reset time field 1-67
maximum number of queued interrupts field 1-69
maximum reset control block interrupt time field 1-68
maximum reset interrupt status port time field 1-68
maximum size for completion status field 1-69
maximum size for diagnostic status field 1-69
maximum subsystem reset time field 1-66
memory address space 1-29, 2-4
memory address x 4-10
memory address 1 field 1-39
memory address 2 field 1-39, 1-56, 1-59, 1-71
memory-address-1-selected bit 1-36, 1-37
memory-address-2-selected bit 1-35, 1-37
microseconds bit 1-67
milliseconds bit 1-67
multiple-form interrupt
  See DIIP interrupt

N

nanoseconds bit 1-67
no chaining 4-7
no operation command 1-57, 1-76
  disable-command-interrupt bit 1-76
no operation command *(continued)*
  disable-device-interrupt bit 1-76
non-control-block interrupt 3-19, 3-21, 3-22, 3-30, 3-34
non-DIIP interrupt 3-16, 3-37
number of devices supported field 1-66

O

op code dependent field 1-73
op code field 1-32, 1-74
op-code values 1-32
operation code field 1-32, 1-74
operation-code bits (format 1) 1-75
operational characteristics, locate mode 1-3
option-1 bit 1-78, 1-84
option-2 bit 1-78, 1-84
options field 1-32
options, initialize device command 1-57
overview, locate mode 1-1

P

physical data block 4-16
physical interface 1-15
  type 1 1-15, 1-16
  type 2 1-15, 1-17
physical interrupt 3-14, 3-19, 3-20, 3-30, 3-31
physical level 2-1
physical-interrupt enablement 3-15
physical-level protocols 2-9
physical-level services 2-6
  data delivery 2-9
  interrupt 2-8
  interrupt-service parameters 2-8
  pull 2-6
  pull-service parameters 2-7
  push 2-6

physical-level services *(continued)*
push-service parameters 2-6
physical-level structure 2-1
hardware support, subsystem 2-3
hardware support, system master 2-3
I/O address space 2-4
memory address space 2-4
subsystem hardware support 2-3
support logic 2-4
system-master hardware support 2-3
port 1-9
ports
attention 1-9, 1-18, 1-48, 2-3, 5-2
attention code field 1-19
device number field 1-20
command 1-9, 1-18, 2-3
command busy/status 1-9, 1-24, 2-3, 5-2
command interface 5-2
device interrupt identifier 1-10, 1-28, 2-3, 3-39, 5-2
interrupt status 1-9, 1-21, 1-79, 2-3, 3-38, 5-2
device number field 1-24
interrupt identifier field 1-22
request ports 1-9
attention 1-9, 1-48, 2-3, 5-2
command 1-9, 2-3
command busy/status 1-9, 2-3
device interrupt identifier 1-10, 2-3, 3-39, 5-2
interrupt status 1-9, 1-79, 2-3, 3-38, 5-2
subsystem control 1-9, 1-20, 1-90, 2-3, 5-2
POS register 2 field 1-65
POS register 3 field 1-65
POS register 4 field 1-65
POS register 5 field 1-65
POS register 6 field 1-66
POS register 7 field 1-66
processing level 4-1
command chaining 4-1, 4-2
command-chaining options 4-7, 4-8
conditional command chaining 4-5
control-block execution 4-9
control-block chaining 4-1
data chaining 4-1, 4-10
base control block 4-12, 4-14
extended control block 4-14, 4-16
indirect list 4-12, 4-14, 4-16
physical data block 4-16
read indirect list 4-12, 4-14
write indirect list 4-14, 4-16
pull 2-6
push 2-6

R

read command 1-58
read completion status command 1-53, 1-60
read configuration command 1-63, 5-3
attachment ID field 1-65
attachment revision level field 1-66
device dependent area size field 1-69
device dependent configuration data area field 1-70
DIIP address field 1-69
DIIP bit position field 1-69
flags field 1-67
interrupt level field 1-66
maximum command/busy status time field 1-69

read configuration command
(continued)
maximum device reset time field 1-67
maximum number of queued interrupts field 1-69
maximum reset control block interrupt time field 1-68
maximum reset interrupt status port time field 1-68
maximum size for completion status field 1-69
maximum size for diagnostic status field 1-69
maximum subsystem reset time field 1-66
number of devices supported field 1-66
POS register 2 field 1-65
POS register 3 field 1-65
POS register 4 field 1-65
POS register 5 field 1-65
POS register 6 field 1-66
POS register 7 field 1-66
read indirect list 4-12, 4-14
reject bit 1-9, 1-26, 1-91
request ports 1-9
reset bit 1-68, 1-90
reset control block interrupt command 1-80
count field 1-81
disable-command-interrupt bit 1-81
disable-device-interrupt bit 1-81
reset device command 1-77
disable-command-interrupt bit 1-78
disable-device-interrupt bit 1-78
option-1 bit 1-78
option-2 bit 1-78
reset interrupt status port command 1-79
disable-command-interrupt bit 1-80
reset interrupt status port command
(continued)
disable-device-interrupt bit 1-80
reset reject command 1-9
reset subsystem command (hardware controlled) 1-85, 1-90
reset subsystem command (software controlled) 1-82
disable-command-interrupt bit 1-84
disable-device-interrupt bit 1-84
option-1 bit 1-84
option-2 bit 1-84
reset-reject bit 1-20
resetting interrupts 3-38
residual buffer address 1-34, 1-72
residual buffer address field 1-59, 5-20, 5-22
residual buffer address 1 field 1-51
residual buffer address 2 field 1-44, 1-52
residual byte count 1-34, 1-71
residual byte count field 1-59, 5-19, 5-22
residual byte count 1 field 1-50
residual byte count 2 field 1-44, 1-51
residual data values 5-19
number of bytes transferred 5-21
residual buffer address 5-20
residual byte count 5-19
restart bit 1-57
resume command 1-86
disable-command-interrupt bit 1-86
disable-device-interrupt bit 1-86
run diagnostic test command 1-70, 1-71, 1-87
run immediate diagnostic test command 1-87
diagnostic command field 1-87
disable-command-interrupt bit 1-87

run immediate diagnostic test command *(continued)*
disable-device-interrupt bit 1-87

S

second-level interrupt-handling routine 3-21
seconds bit 1-67
setup 5-1
shared memory 1-29
short-length exception 4-14, 5-5, 5-9
short-length-exception bit 1-49
signalling protocol 3-13
simple-form interrupt
See non-DIIP interrupt
software-controlled command delivery 3-4
software-controlled reset subsystem command
See reset subsystem command (software controlled)
specification exception 5-5, 5-13
specification-check bit 1-49
start address field 1-57
status bit 1-9, 1-25
subsystem control port 1-9, 1-20, 1-90, 2-3, 5-2
subsystem does not respond 5-7
subsystem hardware support 2-3
subsystem-reset bit 1-20
support logic 2-4
suppress-exception-long bit 1-34
suppress-exception-short bit 1-34
suspend command 1-47, 1-88
disable-command-interrupt bit 1-89
disable-device-interrupt bit 1-89
suspended bit 1-47
system-master hardware support 2-3

T

termination status block 1-43
termination status block address field 1-40, 1-43
termination-status-block structure 1-11
base termination status block 1-44
device dependent data area field 1-53
device dependent data size field 1-53
end status word 1 1-57, 1-60
end status word 1 field 1-44, 1-46, 1-72
end status word 2 field 1-47, 1-50
extended termination status block 1-45
residual buffer address 1-59
residual buffer address 1 field 1-51
residual buffer address 2 field 1-44, 1-52
residual byte count 1-59
residual byte count 1 field 1-50
residual byte count 2 field 1-44, 1-51
time-field specifier bits 1-67
TSB-available bit 1-48
two-way chaining 4-8
type 1 physical interface 1-15, 1-16
type 2 physical interface 1-15, 1-17

W

write command 1-71
byte count 1 field 1-71
end status word 1 field 1-72
memory address 2 field 1-71
write indirect list 4-14, 4-16

Move Mode

Chapter 1. Overview 1-1
Operational Characteristics 1-4
Peer to Peer Relationships 1-7
Queueing Control Elements 1-9
Control Element Delivery 1-11
I/O Address Space 1-15
Physical Interface Support 1-15
Attention Port 1-16
Subsystem Control Port 1-17
Command Busy/Status Port 1-18
Memory Address Space 1-19
Delivery Pipe 1-19
Local Enqueue Control Area 1-21
Surrogate Enqueue Control Area 1-24
Local Dequeue Control Area 1-25
Surrogate Dequeue Control Area 1-28
Signalling Control Area 1-29
Control Elements 1-32
Basic Structure 1-32
Type Field 1-33
Length Field 1-33
Value Field 1-33
General Structure 1-33
Common Indicators Field 1-34
Source Field 1-37
Destination Field 1-37
Correlation Field 1-38
Entity to Entity Field 1-38
Function Codes 1-39
Request, Reply, and Error Control Elements 1-40
Initialize 1-40
Read 1-42
Read List 1-43
Read Immediate 1-45
Write 1-47
Write List 1-49
Write Immediate 1-50
Execute List 1-52
Mark 1-54

© Copyright IBM Corp. 1991.

Cancel 1-54
Reset 1-57
Read Configuration 1-58
Diagnose 1-60
Event Control Elements 1-62
Resume 1-62
Notification 1-63
Inform 1-63
Wrap 1-64

Chapter 2. Physical Level 2-1
Structure 2-1
System 2-3
Adapter 2-3
Support Logic 2-3
Push and Pull 2-4
Signal 2-5
Memory Address Space 2-8
I/O Address Space 2-9
Physical Level Services 2-10
Push 2-10
Pull 2-11
Signal 2-12
Data Delivery 2-13
Physical Level Protocols 2-13
Control Areas 2-13
Physical Interface 2-14
Feature Adapter to System Unit Protocols 2-14
Feature Adapter to Feature Adapter Protocols 2-16

Chapter 3. Delivery Level 3-1
Structure 3-1
Delivery Pipes 3-3
Signalling 3-5
Send and Receive Interfaces 3-5
Delivery Level Services 3-6
Enqueue Service 3-6
Dequeue Service 3-7
Delivery Level Protocols 3-8
Enqueue Protocol 3-11
Enqueue Initialization 3-14
Enqueue Receive Signal 3-14
Enqueue Function 3-15

Enqueue State Change Function 3-18
Dequeue Protocol 3-19
Dequeue Initialization 3-20
Dequeue Receive Signal 3-21
Dequeue Function 3-21
Dequeue State Change Function 3-23

Chapter 4. Processing Level 4-1
Structure 4-1
Processing Level Operation 4-2
Send 4-6
Receive 4-6
Interrupt 4-8
Processing Level Services 4-8
Processing Level Protocols 4-9
Command Chaining 4-9
Data Chaining 4-9
Direct Data Chaining 4-10
Indirect Data Chaining 4-10
Notification and Wait 4-11

Chapter 5. Design Considerations 5-1
Configuration 5-1
Unit Level 5-1
System Level 5-2
Peer Level 5-3
Initialization 5-4
Unit Level Initialization 5-4
System Level Initialization 5-4
Peer Level Initialization 5-10
Exception Handling 5-13
Management Relationships 5-14

Chapter 6. Architectural Compliance 6-1
Physical Level 6-2
Base Protocols 6-2
Additional Architected Protocols 6-2
Delivery Level 6-2
Base Protocols 6-2
Additional Architected Protocols 6-3
Processing Level 6-3
Base Protocols 6-3
Additional Architected Protocols 6-4

Management 6-4
Base Protocols 6-4
Additional Architected Protocols 6-4

Appendix A. C Language A-1

Appendix B. Assembler Language B-1

Index X-1

Figures

1-1. Control Element Delivery Structure 1-3
1-2. Peer to Peer Delivery Model 1-8
1-3. Handling Control Elements 1-10
1-4. Control Element Delivery Flow 1-12
1-5. I/O Address Space Map - Physical Interface 1-16
1-6. Attention Port 1-17
1-7. Subsystem Control Port 1-17
1-8. Command Busy/Status Port 1-18
1-9. Pipe Area 1-20
1-10. Control Areas Associated with a Single Pipe 1-21
1-11. Local Enqueue Control Area Structure 1-22
1-12. Enqueue Status Field 1-22
1-13. Surrogate Start of Free Field 1-24
1-14. Surrogate Enqueue Status Field 1-25
1-15. Local Dequeue Control Area Structure 1-26
1-16. Dequeue Status Field 1-26
1-17. Surrogate Start of Elements Field 1-28
1-18. Surrogate Dequeue Status Field 1-29
1-19. Signalling Control Areas 1-30
1-20. Signalling Control Area Structure 1-30
1-21. Basic Control Element Structure 1-33
1-22. Control Element Structure - FID 0 1-34
1-23. Control Element Common Indicators Field 1-35
1-24. Control Element Source and Destination Fields 1-38
1-25. Architected Function Codes in the Common Indicators Field 1-39
1-26. Initialization Request Control Element 1-40
1-27. Initialization Reply Control Element 1-41
1-28. Initialization Error Control Element 1-41
1-29. Read Request Control Element 1-42
1-30. Read Reply Control Element 1-43
1-31. Read Error Control Element 1-43
1-32. Read List Request Control Element 1-44
1-33. Read List Error Control Element 1-45
1-34. Read Immediate Request Control Element 1-45
1-35. Read Immediate Reply Control Element 1-46
1-36. Read Immediate Error Control Element 1-47
1-37. Write Request Control Element 1-47
1-38. Write Reply Control Element 1-48

© Copyright IBM Corp. 1991.

1-39. Write Error Control Element 1-48
1-40. Write List Request Control Element 1-49
1-41. Write List Error Control Element 1-50
1-42. Write Immediate Request Control Element 1-51
1-43. Write Immediate Reply Control Element 1-51
1-44. Write Immediate Error Control Element 1-52
1-45. Execute List Request Control Element 1-53
1-46. Execute List Reply Control Element 1-53
1-47. Mark Request Control Element 1-54
1-48. Cancel Request Control Element 1-55
1-49. Cancel Reply Control Element 1-56
1-50. Cancel Error Control Element 1-56
1-51. Reset Request Control Element 1-57
1-52. Reset Reply Control Element 1-58
1-53. Reset Error Control Element 1-58
1-54. Read Configuration Request Control Element 1-59
1-55. Read Configuration Reply Control Element 1-59
1-56. Read Configuration Error Control Element 1-60
1-57. Diagnose Request Control Element 1-60
1-58. Diagnose Reply Control Element 1-61
1-59. Diagnose Error Control Element 1-62
1-60. Resume Event Control Element 1-63
1-61. Notification Event Control Element 1-63
1-62. Inform Event Control Element 1-64
1-63. Wrap Event Control Element 1-64
2-1. Physical Level 2-2
2-2. Signalling 2-6
2-3. Interrupt Support Logic 2-7
3-1. Overall Delivery Level Structure 3-2
3-2. Delivery Pipe and Associated Control Areas 3-3
3-3. Multiple Unit Structure 3-4
3-4. Delivery Pipe and Associated Control Areas 3-9
3-5. Signalling Control Area Use 3-10
3-6. Various Examples of Pipe Use 3-11
4-1. Device Driver or Handler Model 4-3
4-2. Device Driver or Handler Model 4-5
4-3. Indirect Data Chaining 4-11
4-4. Notify and Wait Actions 4-12
5-1. System Configuration Record Format 5-5
5-2. Configuration Field Format 5-7
5-3. Peer Configuration Record Format 5-11
5-4. Unit Management Relationships 5-15

Chapter 1. Overview

The Move mode supports the delivery of control information between cooperating entities using a variable-length control element. This control element can contain requests, replies, errors, or event notifications for an entity in another adapter or system unit, or for entities in the same adapter or system unit.

At the processing level, operation of the Move mode is similar to that of the Locate mode (client entities build requests, requests are delivered to server entities, server entities build replies, and replies are delivered to client entities).

However, in the Move mode, a client entity can be in an adapter as well as in a system unit, which means an entity acting as a client in an adapter can send requests to a server entity in another adapter or to the system unit. This capability provides a peer-to-peer relationship between entities independent of their role (client or server) or their physical location (adapter or system unit). A server can be in the system unit, which means that the system unit can receive requests from an adapter.

The Move mode defines a method of delivery that permits a varying number of control elements to be moved from an entity in one unit to a correspondent entity in another unit through a pair of delivery pipes. An entity uses one pipe for sending control elements and the other pipe for receiving control elements. Each pipe can contain requests, replies, errors or event notifications for a specific entity in another unit, or for different entities in the same unit.

The Move mode defines the pipe-related protocols and the services that support the control element delivery.

Figure 1-1 on page 1-3 shows how various levels of delivery-support relate to each other and to the various control areas in I/O space and shared memory. This figure also shows that there is send-interface logic and receive-interface logic between the delivery support and the client and server entities. These interfaces provide the support necessary to adapt the entity-interface requirements to those of delivery support, taking into account the specific local operating

© Copyright IBM Corp. 1991.

environment. There is, however, no unit-to-unit protocol associated with the send-interface logic and receive-interface logic.

The architecture definition of the control areas, control elements, services, and protocols used in Move mode are presented in the remainder of this manual.

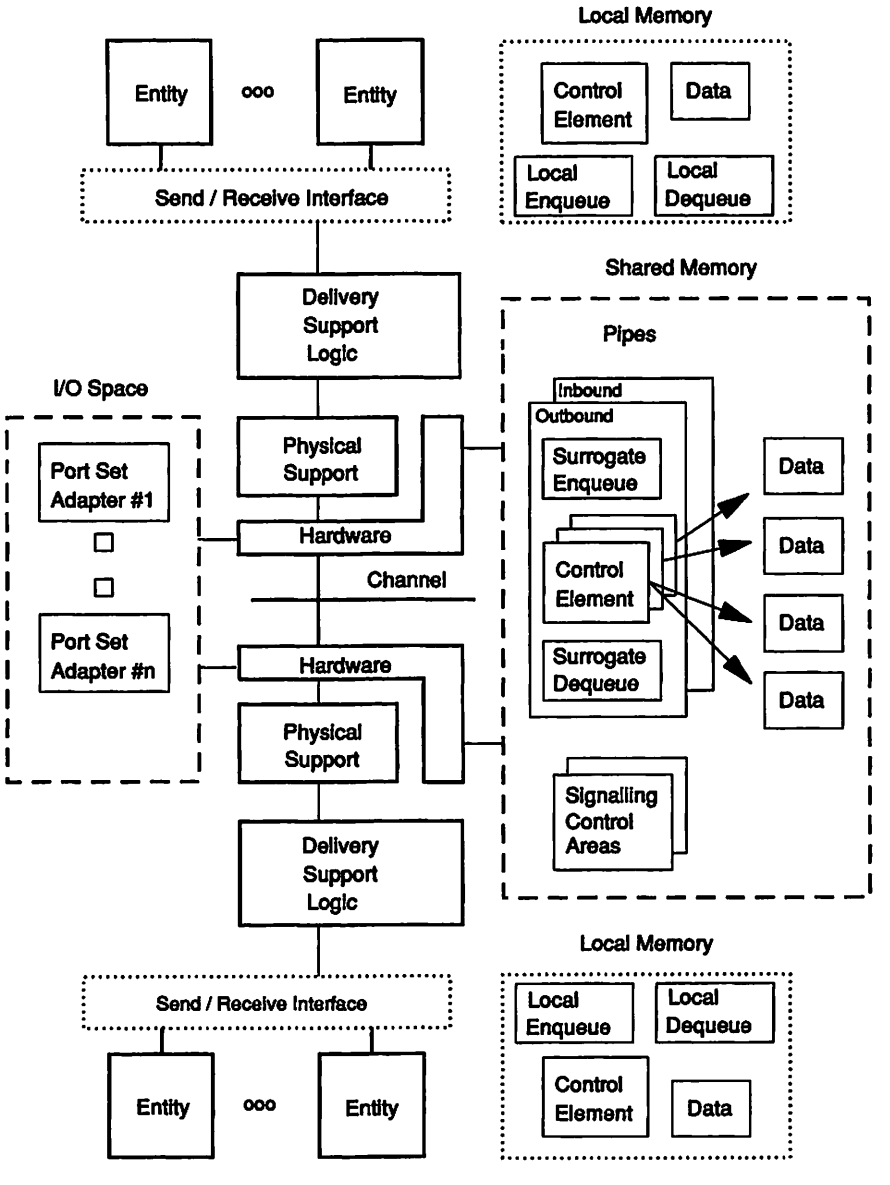


Figure 1-1. Control Element Delivery Structure

Operational Characteristics

The following are some of the key operational characteristics of the Move mode.

- Request and Reply Support

  A request and reply protocol is used to support the delivery of control elements. An adapter can send requests to the system unit and receive replies from the system unit because an adapter can contain a client entity or a server entity. This support includes a means to:

  - Send one or more requests and receive one or more replies in response.
  - Correlate a request with one or more replies.
  - Identify the source and destination of both requests and replies.
  - Send and receive unsolicited information.

- Bidirectional Delivery

  Requests and replies can flow in both directions.

- Peer-to-Peer Delivery Support

  An entity in a system unit or adapter can send requests directly to, and receive replies directly from the entities in a system unit or another adapter attached to the channel. This means an entity in an adapter can send requests and receive replies directly from an entity in another adapter without having to go through a system unit.

- Variable Length Requests and Replies

  Requests and replies are exchanged using variable-length control elements. The control elements allow a variable number of parameters of any size to be passed between client and server entities. This provides a simple way of tailoring control elements to the task to be performed rather than using fixed-length control elements.

- Delivery Pipes

  Request and reply control elements are moved between entities in a source unit and entities in a destination unit using a delivery pipe. A delivery pipe is similar to a first-in, first-out queue, which makes the control elements appear to be contiguous. This allows for the chaining of multiple control elements into a single request.

- Asynchronous Operation

  The use of a delivery pipe allows multiple control elements to be delivered and processed asynchronously by the entities within each unit (system unit or adapter).

- Full Duplex

  A pipe for each direction of delivery between each unit (system unit or adapter) allows control elements to be delivered in one direction independent of the flow of control elements in the other direction.

- Multiplexed

  Control elements for multiple entities in the same destination unit can be found within the same delivery pipe.

  The delivery pipe supports the intermixing of requests and replies as well as other types of control elements (errors and events) within the same pipe for different entities.

- Continuous Running

  The structure of the delivery pipe allows control element delivery to appear as a continuous stream of control elements flowing between client and server entities in the source and destination units. There are elements of the architecture that assist in the management of this continuous stream of control elements:

  - Suspending the delivery of control elements at the entity-to-entity level.
  - Notifying the destination entity that a control element is available.
  - Synchronizing the exchange of control elements between source and destination entities.

- Shared Memory

  Shared memory allows control structures that convey control elements, as well as the control structures supporting the delivery pipes, to be placed in the memory of either an adapter or system unit. This greatly enhances the ability to make cost and performance trade-offs when designing advanced function adapters.

- Reduction of Interrupts

  It is possible to reduce or eliminate the use of physical interrupts between units (system unit or adapter). Each delivery pipe can specify whether interrupts are used and what conditions require the use of interrupts.

Peer to Peer Relationships

In a peer-to-peer relationship, entities performing the roles of clients and servers are not restricted to being in an adapter or a system unit, but can be located in either place. Therefore, control elements can be delivered directly between a system unit and an adapter, or between any two adapters on the channel.

To operate in a peer-to-peer relationship, the delivery support must allow requests and replies to flow in either direction and allow control elements for entities in the same destination unit to be intermixed on the same delivery pipe. In control-element delivery, this is supported by having independent delivery of control elements in either direction, and by allowing clients to be located in system units or adapters, and servers to be located in system units or adapters. A peer-to-peer relationship also requires support in both system units and adapters to resolve contention when two or more system units or adapters attempt to deliver control elements to the same destination at the same time.

The term *peer-to-peer* must be qualified by the level of support being discussed and not used as a unit-to-unit term without qualification.

The peer-to-peer delivery support has a pair of delivery pipes for each unit-to-unit pair with entities that communicate with each other. An example of this is shown in the following figure. The labels R1, R2, and so on, on the client and server entities indicate the entity pairings. The dotted lines indicate the delivery support portion in each unit.

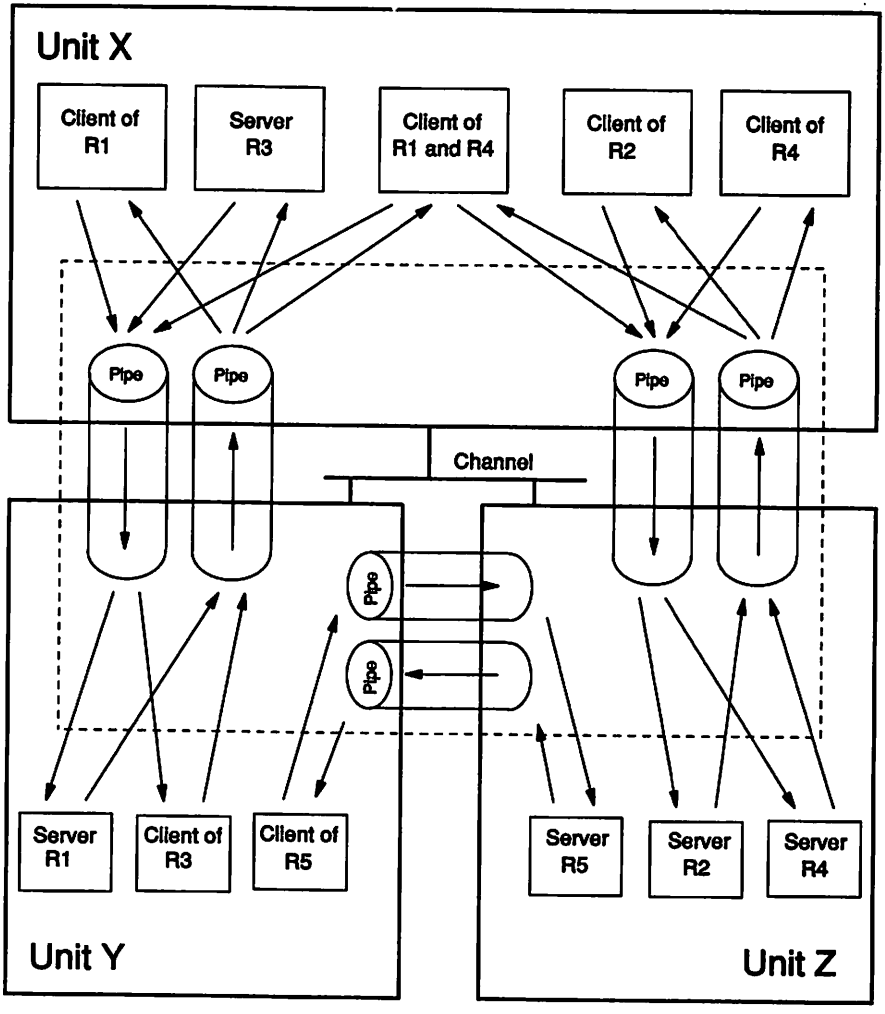


Figure 1-2. Peer to Peer Delivery Model

Queueing Control Elements

Figure 1-3 on page 1-10 shows an example of the flow of a request through the delivery pipe to the specific server queues. Each entity pair is responsible for ensuring that a pending-receive indication is sent so that one entity cannot block others from using the pipe. If the pending-receive indication is not sent, the delivery-level support can discard the control element and notify the entity that sent the control element. Queueing by the server is an optional function, which is performed within the processing level.

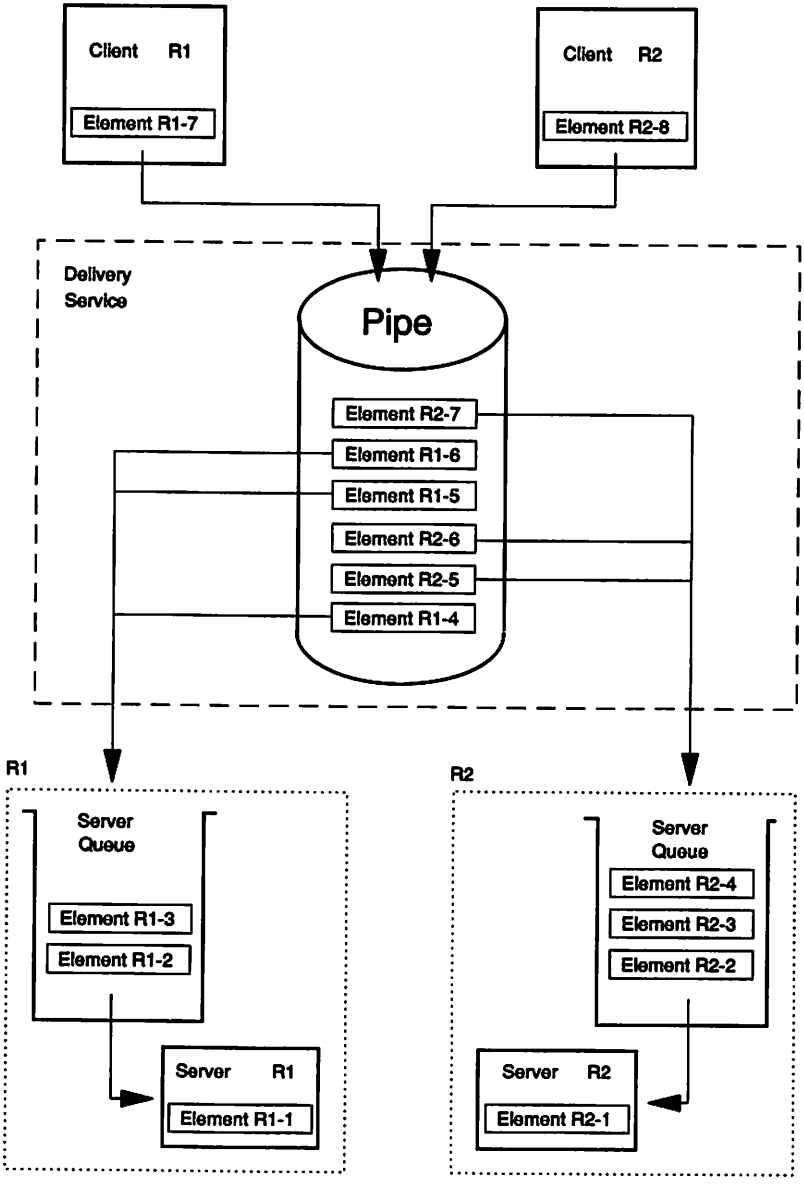


Figure 1-3. Handling Control Elements

Control Element Delivery

The following figure shows the overall flow of a request or a reply between the server entity (device attachment or processing software) in an adapter and the client entity in another adapter or system unit.

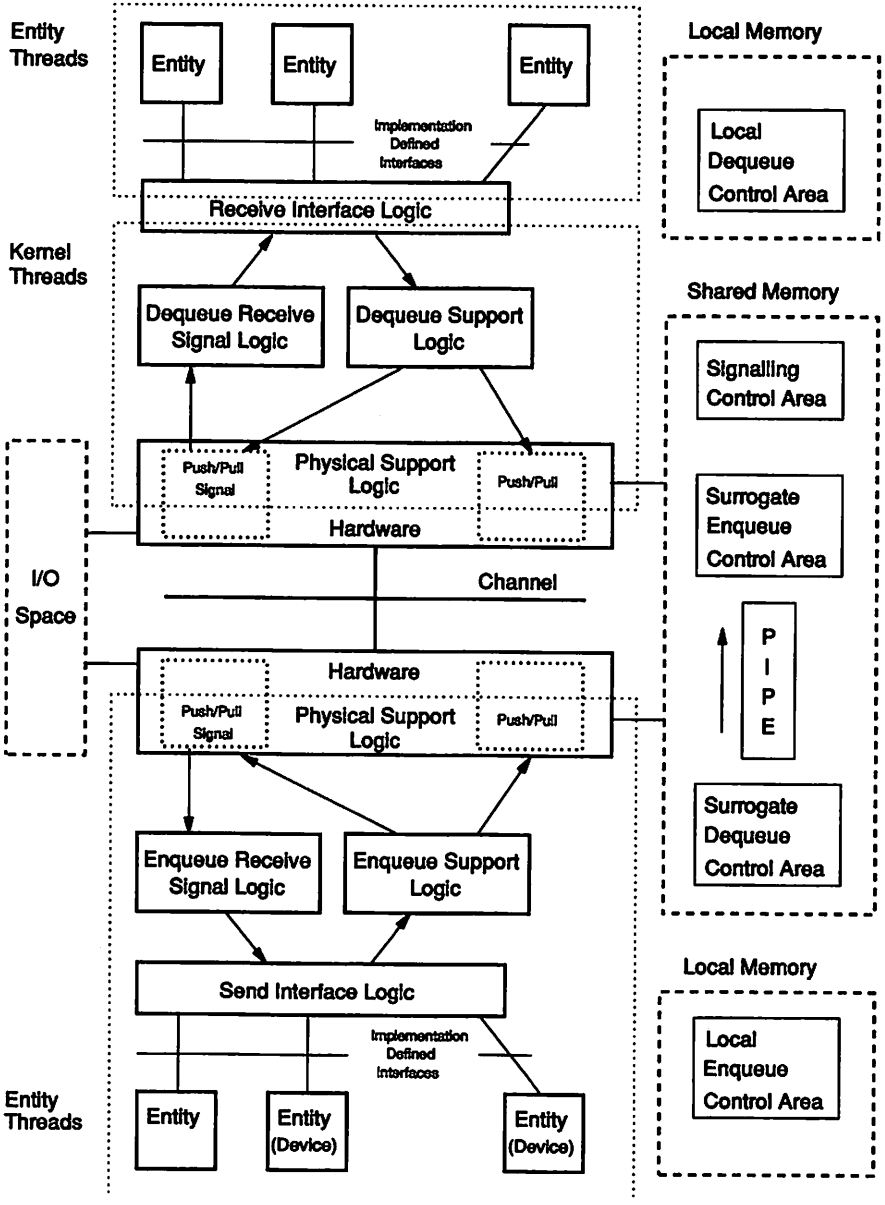


Figure 1-4. Control Element Delivery Flow

The following describes the flow and logic for delivering a request to the server. The flow and logic for delivering a reply to a client is identical but inverted.

The send-interface logic is responsible for sending control elements from an entity in one unit to a correspondent entity in the same or different unit. This interface enforces the policies of the send operation that are consistent with a particular implementation and operating environment. For example, it defines the form that the call takes, and how waits and sharing of the delivery pipe are handled. The send-interface logic also handles the changing of addresses from the local address space to the address space in shared memory (this logic is determined by, and specific to, the implementation).

The send-interface logic calls the enqueue logic with the address and count of the number of control elements to be sent. Using the information in the Destination and Length fields of each control element, the enqueue logic places the control elements one at a time into the appropriate delivery pipe and updates the associated pointers. This continues until all the control elements are placed into the delivery pipe or the pipe becomes full.

If the pipe becomes full, the enqueue logic stops enqueueing control elements, updates the Status fields in the Local and Surrogate Enqueue Control areas, sends a signal to the dequeue logic at the other end of the delivery pipe, and returns a pipe-full condition to the send-interface logic

If the delivery pipe is empty when a control element is enqueued, the enqueue logic updates the Status fields in the Local and Surrogate Enqueue Control areas, and sends a signal to the dequeue logic at the other end of the delivery pipe.

The enqueue logic sends a signal to the dequeue logic by first setting the dequeue state-change indicator in the Signalling Control area assigned to the unit and then invoking the signalling services of the physical-support logic.

If the system unit is being signalled, then the physical-support logic activates the appropriate interrupt-request line. If the adapter is being signalled, then the physical-support logic writes the appropriate attention code into the Attention port of the adapter.

Note: Each state-change indicator (enqueue, dequeue, and management) is assigned to a separate byte in the Signalling Control area. This allows each state-change indicator to be written separately and requires only the posting of the appropriate delivery-support or management-support logic when a state-change indicator is set.

On the receive side, the receive-interface logic is responsible for providing the interface between the dequeue logic and the entities that receive control elements from correspondent entities. This interface enforces the policies of the receive operation that are consistent with a particular implementation and operating environment. For example, it defines:

- What form the call takes
- How waits are handled
- Whether a read pending is indicated to remove control elements or a queue of the removed control elements is maintained.

The receive-interface logic also handles the transformation of addresses and the movement of control elements from the shared memory address space to the local address space of the receiving entity or queue (these are determined by, and specific to, the implementation).

The receive-interface logic initiates the removal of control elements from a delivery pipe by calling the dequeue logic. If there is a control element in the delivery pipe, the dequeue logic calls the receive-interface logic with the address of the control element to be dequeued. The receive-interface logic, using information in the control element (Destination and Length fields), determines where the control element should be sent. Depending on the policy of the receive-interface logic, it can choose to:

- Route it directly to the destination entity
- Place it on a queue maintained by the receive-interface logic for the destination entity
- Return it to the originating entity
- Discard it.

When the receive-interface logic returns to the dequeue logic, the information in the Local and Surrogate Control areas associated with the delivery pipe is updated and the process is repeated until all control elements in the delivery pipe are removed. At this time, the

dequeue logic returns to the receive-interface logic at the point of the original invocation.

Receiving a signal sent by the enqueue logic at the other end of the delivery pipe can also initiate removing control elements from a delivery pipe. In this case, the interrupt logic associated with the physical-support logic examines all Signalling Control areas and, based on the state-change indicators, posts the appropriate dequeue receive-signal logic.

The posting, in effect, allows the receive-interface logic and dequeue logic to run under the kernel thread. It also allows the logic to synchronize with the entity thread portion of the receive-interface logic for the entity indicated by the Destination field of each control element removed from the delivery pipe.

I/O Address Space

Multiple ports are defined in the architecture for the physical interface. In this context, a *Port* describes a byte or set of contiguous bytes located in I/O address space.

Since several adapters can be present on the channel, the I/O addresses are shown as offsets from the base I/O address of the adapter. The base I/O address used by each adapter is determined during system configuration.

Physical Interface Support

The physical interface supports the delivery of control elements between a system unit and an adapter as well as between adapters. It uses the following subset of the Locate mode ports:

- Attention port
- Subsystem Control port
- Command Busy/Status port.

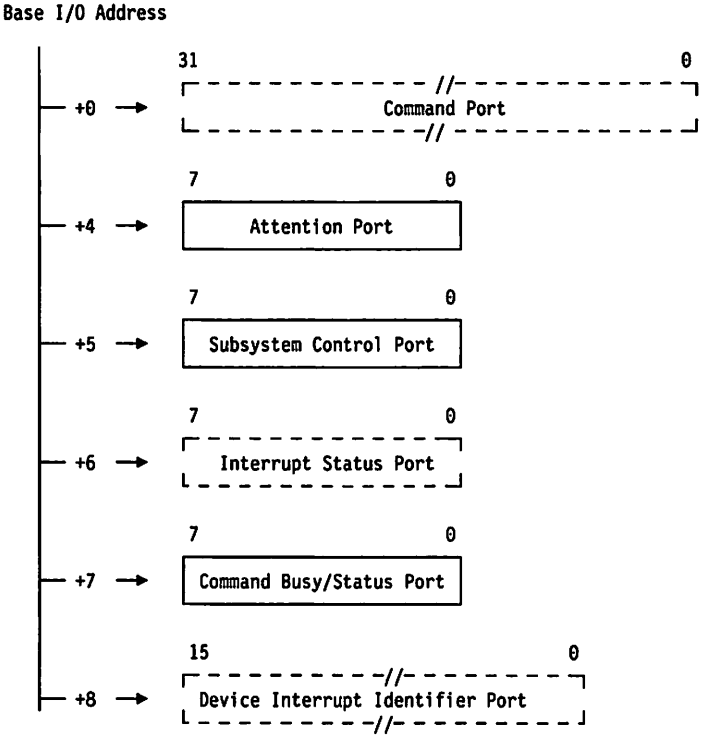


Figure 1-5. I/O Address Space Map - Physical Interface

The Command, Interrupt Status, and Device Interrupt Identifier ports are present, but are not used by Move mode after the physical adapter has reached operational state.

Attention Port

The Attention port is an 8-bit port used by the physical-support logic in a system unit or adapter to signal, or request the attention of, another adapter. Writing to the Attention port causes the adapter to be interrupted and indicates that signalling information has been placed into the Signalling Control area assigned to the system unit or adapter writing to the Attention port. (See "Signal" on page 2-5 for additional information on the Signalling operation and "Signalling Control Area" on page 1-29 for a description of the Signalling Control areas.)

The format and content of the Attention port are shown in the following figure.

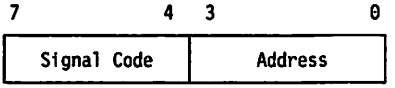


Figure 1-6. Attention Port

The Signal Code field (bits 7-4) must be set to hex D, and the Address field (bits 3-0) must be set to 0.

Subsystem Control Port

The Subsystem Control port is an 8-bit port that is used by the system unit to provide direct hardware control of several adapter functions. This port contains control bits that are used to:

- Enable interrupts from the adapter
- Enable the bus-master operations by the adapter
- Reset the adapter.

The following figure shows the format and content of the Subsystem Control port and is followed by a description of the individual bits.

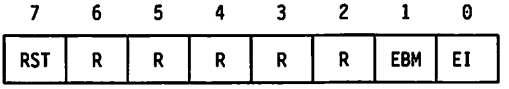


Figure 1-7. Subsystem Control Port

Bit 7 The reset bit (RST) is used to provide a hardware-controlled reset of the adapter and its associated resources and attached devices. When set to 1, all subsystem and device activities on the adapter are halted and the subsystem is placed in the reset state.

Bits 6–2 These bits are reserved.

Bit 1 The enable-bus-master bit (EBM) is used to control bus-master operations from the adapter. Setting this bit to 1 enables the adapter to initiate bus-master operations. Setting this bit to 0 disables the initiation of bus-master operations by the adapter. The enable-bus-master bit is initialized to 0 when the adapter is powered-on or reset.

Note: The enable-bus-master bit is primarily used to debug hardware. However, it can also be used to

force the adapter to halt adapter-initiated channel activity.

Bit 0 The enable-interrupt bit (EI) is used to control the interrupts generated by requests from the adapter. Setting this bit to 1 allows the adapter to send interrupt requests to the system unit. Setting the bit to 0 prevents the adapter from sending physical interrupts. The enable-interrupt bit is set to 0 when the adapter is powered-on or reset.

Command Busy/Status Port

The Command Busy/Status port is an 8-bit port that has three functions:

- It indicates that an interrupt to the system unit is pending.
- It indicates that a signal to the adapter is pending when used in conjunction with the Attention port.
- It indicates that a reset request is in progress in the adapter when used in conjunction with the Subsystem Control port.

The following figure shows the format and content of the Command Busy/Status port and is followed by a description of the individual bits.

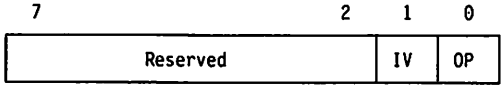


Figure 1-8. Command Busy/Status Port

Bits 7–2 These bits are reserved.

Bit 1 The interrupt-valid bit (IV) indicates that there is a valid interrupt request pending from this adapter to the system unit. It is set as part of the signalling logic and is reset when the physical-support logic in the system unit reads the Command Busy/Status port.

Bit 0 The operation-pending bit (OP) indicates the current state of a signalling operation or the progress of a hardware reset of the adapter (for example, the toggling of the reset (RST) bit in the Subsystem Control port).

Memory Address Space

Shared memory is used to hold control elements (request, reply, error, and event) within the in-bound and out-bound pipes. It also contains:

- Surrogate Control areas
- Signalling Control areas
- Data areas.

The format, content, and use of control elements are described in "Control Elements" on page 1-32. This section describes the format, content, and use of the following control areas:

- Delivery pipe areas
- Enqueue Control areas
- Dequeue Control areas
- Signalling Control areas.

Delivery Pipe

A delivery pipe is a control area maintained in shared memory. It is used to support the exchange of control elements between cooperating entities in different units (system units or adapters). The pipe space is managed as a first-in first-out (FIFO) stream in which entities in one unit insert control elements at one end and entities in another unit take the control elements out at the other end.

A system unit or adapter has a pair of pipes for each adapter or system unit it exchanges control elements with. One pipe handles the flow of control elements in one direction; the other pipe handles the flow of control elements in the opposite direction.

Each pipe represents a circular queue and contains an area to hold the actual control elements. The structure of this area is shown in the following figure.

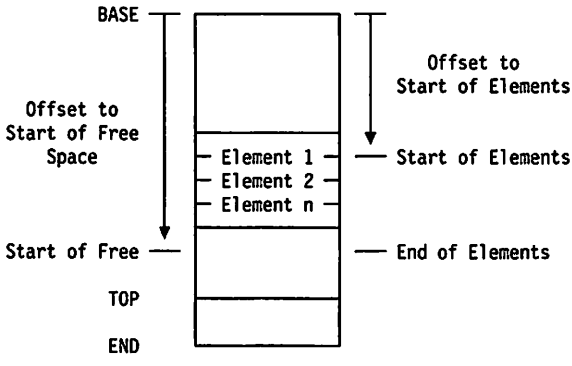


Figure 1-9. Pipe Area

Additional control areas are used to manage the operation of the pipe. For each pipe there is a Local Enqueue Control area, Surrogate Enqueue Control area, Local Dequeue Control area, and Surrogate Dequeue Control area. The relationship of these control areas to the pipe and to each other is shown in Figure 1-10 on page 1-21.

Note: The Surrogate Enqueue Control area is write only to the enqueue logic and read only to the dequeue logic at the other end of the pipe. The Surrogate Dequeue Control area is read only to the enqueue logic and write only to the dequeue logic at the other end of the pipe.

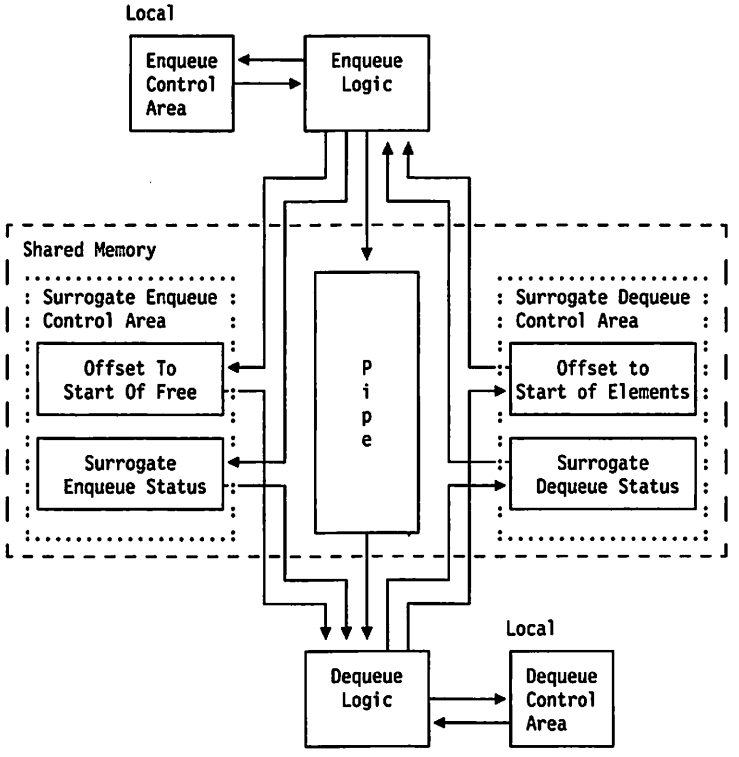


Figure 1-10. Control Areas Associated with a Single Pipe

Local Enqueue Control Area

Within a given unit, there is a Local Enqueue Control area for each pipe. Each control area is identical in structure.

The Local Enqueue Control area is constructed and maintained in either local or shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Fields within each Local Enqueue Control area identify the location of the pipe, indicate its current status (full, empty, or wrap), and provide state information identifying the starting and ending offsets of control elements in the pipe. The following shows the structure and format of the Local Enqueue Control area and describes each of the fields.

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0	
Pipe Address (BASE)	
Enqueue Status (ES)	Wrap Element Offset (WE)
End of Free Space (EF)	Start of Free Space (SF)
Offset to Top (TOP)	Offset to End (END)

Figure 1-11. Local Enqueue Control Area Structure

The Pipe Address field is a 32-bit, doubleword-aligned field containing the 32-bit physical address of the area in shared memory where the circular queue of control elements is maintained.

The Enqueue Status field is a 16-bit, word-aligned field containing bits used to maintain current status information of the pipe as perceived by the local enqueue logic. The format and content of this field are shown below and are followed by a description of each of the bits.

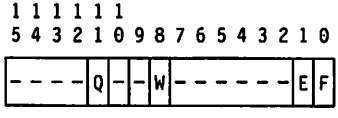


Figure 1-12. Enqueue Status Field

Bits 15–12 These bits are reserved.

Bit 11 The queued bit (Q) indicates that a control element has been successfully placed into the pipe.

Bits 10–9 These bits are reserved.

Bit 8 The wrap bit (W) indicates that a control element would not fit into the space remaining between the start-of-free space (SF) and the top-of-pipe space (TOP). The enqueue logic toggles the current setting of this bit. The bit is initially set to 0 during initialization or after management services re-establish the synchronization of both ends of a delivery pipe.

Bits 7–2 These bits are reserved.

Bit 1 The empty bit (E) is set to reflect the setting of the empty bit in the Surrogate Dequeue Status field.

Bit 0 The full bit (F) indicates whether the pipe is full. The enqueue logic sets this bit to 1 (full) when it tries to place a control element into the pipe and there is insufficient space. The enqueue logic resets this bit each time it successfully places a control element into the pipe. This bit is also reset to 0 (not full) during initialization or after re-establishing the synchronization of the pipe.

Note: The queued and empty bits are implementation specific and are shown here to help in understanding the enqueue algorithm that is presented later.

The Wrap Element Offset field is a 16-bit, word-aligned field containing the offset, in bytes, from the pipe address (BASE) to the location of the wrap control element in the pipe. This field is updated each time a wrap control element is placed into the pipe.

The End of Free Space field (EF) is a 16-bit, word-aligned field containing the offset, in bytes, to the end-of-free space (TAIL) in the pipe. This field represents the end of the available space for placing control elements into the pipe. This field is updated each time the enqueue logic is called. It is checked against the Start of Elements field in the Surrogate Dequeue Control area to insure control elements that have not been dequeued are not overwritten.

The Start of Free Space field (SF) is a 16-bit, word-aligned field containing the offset, in bytes, to the start-of-free space (HEAD) in the pipe. It is the location for the next control element that is placed into the pipe. This field is updated each time a control element is placed into the pipe.

The Offset to Top field (TOP) is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the pipe that is used for holding control elements. This field is set at initialization time and is a copy of information contained in the configuration record.

The Offset to End field (END) is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the pipe. This field represents the

physical size of the pipe when allocating space in shared memory for the pipe and is based on product requirements.

Note: Offset to Top and Offset to End fields ensure that sufficient space is reserved at the end of the pipe to hold a wrap control element.

Surrogate Enqueue Control Area

There are two fields in the Surrogate Enqueue Control area associated with the enqueue operations of a pipe: Surrogate Start of Free (SSF) and Surrogate Enqueue Status (SES). In this context, *Surrogate* means that the information in these two fields is a copy of the information found in the Local Enqueue Control area. This surrogate information is used by the dequeue logic, at the other end of the pipe, in conjunction with the information from the Local Enqueue Control area to manage the distributed aspects of the pipe.

Surrogate Enqueue Control areas are constructed and maintained in shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Surrogate Start of Free: The Surrogate Start of Free field is a 16-bit, word-aligned field containing the offset in bytes from the pipe address (BASE) to the location in the pipe where the enqueue logic places the next control element. The format of the field is shown in the following figure.

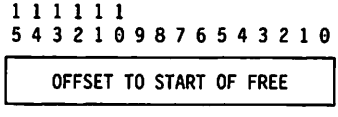


Figure 1-13. Surrogate Start of Free Field

The Surrogate Start of Free field is updated by the enqueue logic after each control element is placed into the pipe. The enqueue logic writes to the field and the dequeue logic reads from the field.

Surrogate Enqueue Status: The Surrogate Enqueue Status field is a 16-bit, word-aligned field that contains bits used to inform the dequeue logic of the current state of the pipe, as perceived by the enqueue logic. The enqueue logic writes to this field, and the dequeue logic reads from this field.

Note: The enqueue logic updates all of the bits in the Surrogate Enqueue Status field each time a control element is place into the pipe.

The following shows the format and content of the field and describes each of the bits.

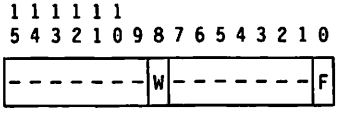


Figure 1-14. Surrogate Enqueue Status Field

Bits 15 – 9 These bits are reserved.

Bits 8, 0 These bits are copies of the bits in the Enqueue Status field. (See the description of the Enqueue Status field on page 1-22.)

Bits 7 – 1 These bits are reserved.

Local Dequeue Control Area

Like the Local Enqueue Control area described previously, there is one Local Dequeue Control area in a given unit for each incoming pipe. They are all identical in structure.

The Local Dequeue Control area is constructed and maintained in either local or shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Fields within the Local Dequeue Control area identify the physical location of the pipe, indicate the current status of the pipe (full, empty, or wrap), and provide state information identifying the starting and ending offsets of control elements in the pipe. The following

figure shows the structure of the Local Dequeue Control area. It is followed by a description of the content and use of each of its fields.

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

Pipe Address (BASE)	
Dequeue Status (DS)	Wrap Element Offset (WE)
End of Elements (EE)	Start of Elements (SE)
Offset to Top (TOP)	Offset to End (END)

Figure 1-15. Local Dequeue Control Area Structure

The Pipe Address field (BASE) is a 32-bit, doubleword-aligned field, containing the 32-bit physical address of the area in shared memory where the circular queue of control elements is maintained.

The Dequeue Status field (DS) is a 16-bit field containing bits used to maintain status information about the current state of the pipe, as perceived by the local dequeue logic. The following figure shows the format and meaning of the bits and is followed by a description of each bit.

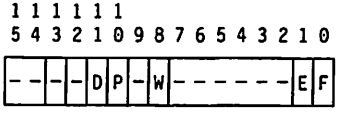


Figure 1-16. Dequeue Status Field

Bits 15 – 12 These bits are reserved.

Bit 11 The dequeued bit (D) is an internal bit that indicates a control element has been removed from the pipe. It is reset to 0 prior to exiting the dequeue logic.

Bit 10 The pre-empt bit (P) is an internal bit that indicates no more control elements will be removed from the pipe, even if the pipe is not empty. This bit is set and reset by the dequeue logic. It is set depending upon the value returned from the receive-interface logic.

Bit 9 This bit is reserved.

Bit 8 The wrap bit (W) indicates that a wrap control element has been placed in the pipe because there was insufficient space between the top of the pipe and the start of free space for the control element. The dequeue logic toggles the current setting of this bit after removing a wrap control element. The bit is initially set to 0 during initialization or after re-establishing the synchronization of the pipe.

Bits 7–2 These bits are reserved.

Bit 1 The empty bit (E) indicates whether the pipe is empty. The dequeue logic sets this bit to 1 when it determines that the pipe is empty. The dequeue logic resets this bit to 0 (not empty) when it determines that the empty bit in the Surrogate Enqueue Status area is set to 0. The bit is also set to 1 (empty) during initialization or after re-establishing the synchronization of the pipe.

Bit 0 The full bit (F) is set to reflect the setting of the full bit in the Surrogate Enqueue Status field.

Note: The dequeued, pre-empt, and full bits are implementation specific and are shown here to help in understanding the dequeue algorithm that is presented later.

The Wrap Element Offset field is a 16-bit, word-aligned field containing the offset, in bytes, from the pipe address to the location of the wrap control element in the pipe.

The End of Elements field is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the control elements in the pipe. This points to the byte immediately following the last control element in the pipe. This field is updated each time the dequeue logic is called; it is a copy of information maintained as an offset in the Start of Free field in the Surrogate Enqueue Control area.

The Start of Elements field is a 16-bit, word-aligned field containing the offset, in bytes, to the start of the next control element in the pipe. This field is updated each time a control element is removed from the pipe.

The Offset To Top field is a 16-bit, word-aligned field containing the offset, in bytes, to the end of the usable area in the pipe for holding control elements. This field is set at initialization time using

information maintained in the configuration record. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

The Offset To End field is a 16-bit, word-aligned field containing the offset, in bytes, to the physical end of the pipe. This field represents the physical size of the pipe when allocating space in shared memory for the pipe and is based on product requirements.

Note: Offset to Top and Offset to End fields ensure that sufficient space is reserved at the end of the pipe to hold a wrap control element.

Surrogate Dequeue Control Area

There are two fields in the Surrogate Dequeue Control area associated with the dequeue operations of a pipe: Surrogate Start of Elements (SSE) and Surrogate Dequeue Status (SDS). In this context, *Surrogate* means that the information in these two fields is a copy of information found in the Local Dequeue Control area. This surrogate information is used by the enqueue logic, at the other end of the pipe, in conjunction with information from the Local Enqueue Control area to manage the distributed aspects of the pipe.

Surrogate Start of Elements: The Surrogate Start of Elements field (SSE) is a single 16-bit, word-aligned field containing the offset, in bytes, from the pipe address to the location in the pipe where the dequeue logic finds the next control element to be removed.

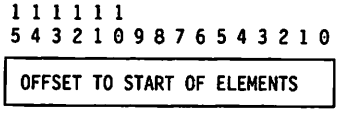


Figure 1-17. Surrogate Start of Elements Field

The Surrogate Start of Elements field is updated by the dequeue logic each time a control element is removed from the pipe. The Surrogate Start of Elements field is read, but never written to by the enqueue logic.

Surrogate Dequeue Status: The Surrogate Dequeue Status field is a single 16-bit, word-aligned field containing bits used to inform the enqueue logic of the current state of the pipe, as perceived by the dequeue logic. The format and content of the field is shown in the following figure. It is followed by a description of the meaning and use of each of its bits.

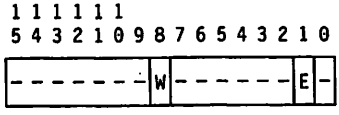


Figure 1-18. Surrogate Dequeue Status Field

Bits 15–9 These bits are reserved.

Bits 8, 1 These bits are copies of the bits in the Local Dequeue Status field. (See the description of the Local Dequeue Status field on page 1-26.)

Bits 7–2 These bits are reserved.

Bit 0 This bit is reserved.

The Surrogate Dequeue Control areas are constructed and maintained in shared memory from information provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Signalling Control Area

A system unit or adapter that receives signals from other units maintains a Signalling Control area in shared memory. It assigns a specific area within this Signalling Control area to each unit that will be signalling it (see Figure 1-19 on page 1-30).

These Signalling Control areas serve two purposes: they provide the means for identifying the source of the signal or interrupt and the means for indicating the reason for the signal or interrupt.

The location of these Signalling Control areas is provided in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details.)

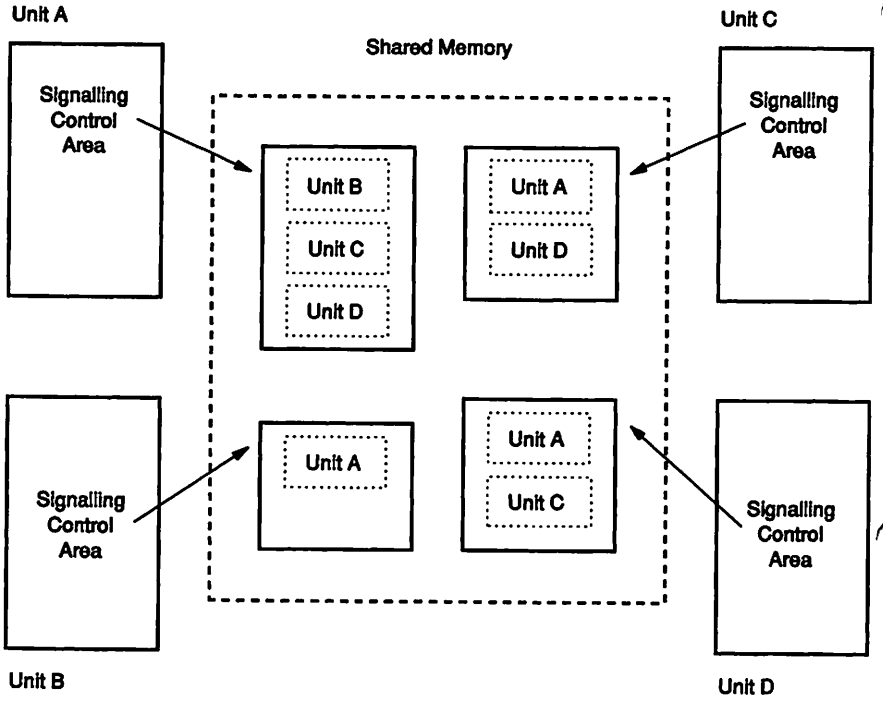


Figure 1-19. Signalling Control Areas

The following figure shows the structure of the Signalling Control area. It is followed by a description of the content and use of each of its fields.

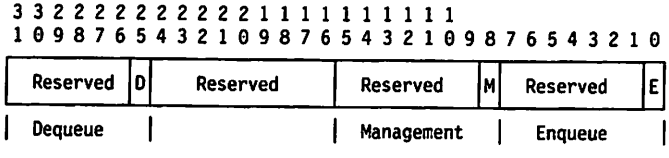


Figure 1-20. Signalling Control Area Structure

The Enqueue field is an 8-bit field; bit 0 indicates that a change in the state of the pipe is being signalled to the enqueue logic by the delivery-support logic at the other end of the pipe. The reason for the state change can be determined by examining the contents of the Surrogate Dequeue Status field associated with the pipe. Some state changes that can cause setting bit 0 of this field are:

- A control element has been placed into a previously empty delivery pipe.
- A control element could not be placed into a delivery pipe due to a full condition.
- A control element has been placed into a delivery pipe.
- A request to re-establish synchronization for the control structures at the other end of the delivery pipe.

Except for the last item, all of the above are optional and are specified in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Bits 7 – 1 of the Enqueue field are reserved.

The Management field is an 8-bit field; bit 0 is used to indicate a change in the state of the delivery logic at one end of the delivery pipe which requires the attention of the management logic at the other end of the delivery pipe. An example of such a state change is the timer expiration.

Bits 7 – 1 of the Management field are reserved.

The Dequeue field is an 8-bit field; bit 0 indicates that a change in the state of the pipe is being signalled to the dequeue logic by the delivery-support logic at the other end of the pipe. The reason for the state change can be determined by examining the contents of the Surrogate Enqueue Status field associated with the pipe. Some state changes that can cause setting bit 0 of this field are:

- A control element has been removed from a previously full delivery pipe.
- The last control element has been removed from a delivery pipe resulting in an empty condition.
- A control element was removed from a delivery pipe.

- A request to re-establish synchronization for the control structures to the other end of the delivery pipe has been requested.

Except for the last item, all of the above are optional and are specified in the configuration record at initialization time. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Bits 7 – 1 of the Dequeue field are reserved.

In addition to identifying the reason for a signal, the Signalling Control area is also used to identify the source of the signal. The system unit and adapters can organize the Signalling Control area as a contiguous table. Each entry in the table would correspond to a Signalling Control area. This allows the location of each Signalling Control area to be assigned sequentially at initialization time, when configuration records are exchanged and updated (see "Configuration" on page 5-1 and "Initialization" on page 5-4 for details on Configuration and Initialization.)

Control Elements

In the Move mode, control elements are used to exchange control information and data between client and server. Some fields of a control element are used by the delivery service, as well as the client and server, while other fields contain information meaningful only to the client and server.

Note: Control elements are not used in Locate mode. In the Move mode, control elements must be aligned in doubleword boundaries.

Basic Structure

Each control element contains three components: type, length, and value. The type component is used to identify the format of the control element. The length component specifies the length of the control element. The value component contains information used by both the delivery service and the cooperating peer entities to

coordinate the exchange and processing of command and control information.

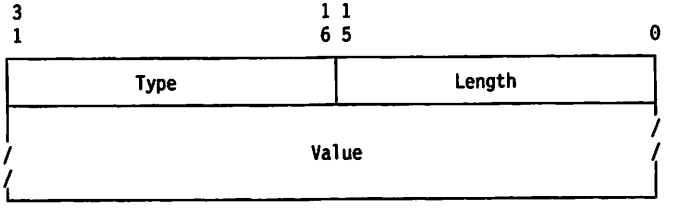


Figure 1-21. Basic Control Element Structure

Type Field

The type component of the control element contains a single field, the Format Identifier field (FID). This field is a 16-bit, word-aligned field used to specify the format and content of the architected portion of the control element. Control elements with a format-identifier value of 0 identify control elements defined for the Move mode.

Length Field

The length component of the control element contains a single field, the Length field. This field is a 16-bit, word-aligned field used to specify the total length of the control element. The length is specified in bytes and includes two bytes for the Length field, and a variable number of bytes for the Value field.

Value Field

The value component of the control element is variable in length and contains information whose structure and content is determined by the value of the Format Identifier field.

General Structure

Move mode control elements are identified by a format identifier of 0. The value component of these control elements consists of a fixed-length header and a variable-length body. The header part is common to all the Move mode control elements and contains the following fields:

- 16-bit Common Indicators field

- 16-bit Reserved field
- 16-bit Source field
- 16-bit Destination field
- 32-bit Correlation field.

The body consists of 0 or more variable-length Entity-to-Entity fields. The presence, format, content, and use of these fields are determined by the content of the Common Indicators field.

The Format Identifier and Length fields along with the content of the Value field are used by the delivery service, as well as the client and server. The remaining field, the Entity-to-Entity field, contains information meaningful only to the client and server.

The following describes, the format and content of each of these fields as it pertains to the delivery service. It also describes a set of common control elements that can be used to set up and manage the exchange of data between client and server.

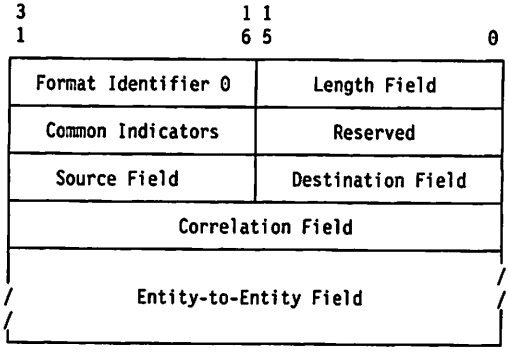


Figure 1-22. Control Element Structure - FID 0

Common Indicators Field

The Common Indicators field is a 16-bit, word-aligned field used to identify a control element and indicate how to interpret the remaining fields within the basic control element. The format and content of the Control Indicators field is illustrated in the following figure.

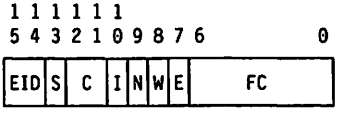


Figure 1-23. Control Element Common Indicators Field

The Common Indicators field consists of the following bits and subfields:

Bits 15, 14 The Element Identifier (EID) is a 2-bit subfield used to identify the control element type. The value in this subfield identifies one of four possible control element types:

ID	Description
00	The request is a control element sent by a client to a server asking it to perform a unit of work on its behalf. It allows the transfer of data and control information between client and server.
01	The reply is a control element sent by a server in response to certain request control elements. Not all request control elements are answered by replies, only those that request information.
10	The event is a control element that contains information about the progress of a request, the state of a previous request, or the status of a client or server. Events are unsolicited and can be sent by either a server or a client.
11	The error is a control element that provides error information about a previous request, or about the status of a server or delivery agent.

Bit 13 The suppress-reply bit (S) indicates whether the client wants the server to return a reply control element when the processing of this request control element has been completed successfully. When set to 1, the reply is suppressed and no reply control element is returned. When set to 0, the entity-to-entity protocol determines whether a reply control element is returned. The

suppress-reply bit is only valid in a request control element.

Bits 12, 11 The chaining bits (C) indicate a group of two or more control elements which represent a single unit of work to be processed in the order in which they appear. The following indicates how the chaining bits are interpreted. The chaining bits are only valid in a request control element.

ID	Description
00	No chaining (each control element is a separate unit of work)
01	First control element in a chain
11	Intermediate control element in a chain
10	Last control element in a chain.

Bit 10 The indirect bit (I) indicates whether the Entity-to-Entity field of a control element contains the parameters for the specified function, or just the length of and a pointer to the area in shared memory where the actual parameters are stored. When set to 1, the Entity-to-Entity field contains the length of and a pointer to the area containing the parameters. The indirect bit is only valid in a request control element.

Bit 9 The notification bit (N) indicates whether a client has requested notification by the server when the processing of this control element begins. When set to 1, a notification is returned to the client in an event control element when processing begins. The notification bit is only valid in a request control element (see "Notification and Wait" on page 4-11 for additional information).

Bit 8 The wait bit (W) indicates whether a client has requested the server to wait before processing this request control element, or any other request control elements from this client. The server suspends processing of the control element from that client until that client instructs the server to resume processing. When set to 1, all request control element processing is suspended, pending the receipt of a resume event control element. The wait bit is

only valid in a request control element (see "Notification and Wait" on page 4-11 for additional information).

Bit 7 The expedite bit (E) identifies the control element that the client wants the server to process as soon as possible (before any other control elements that might be waiting to be processed). When this bit is set to 1, the control element is expedited. The expedite bit is only valid in a request control element.

Bits 6 – 0 The Function Code subfield (FC) is a 7-bit subfield used to identify the function to be performed and indicate how to interpret the contents of the Entity-to-Entity field. Each of the Function Codes are described in detail in "Function Codes" on page 1-39.

Source Field

The Source field is a 16-bit, word-aligned field used to identify the originator or source of a control element. It consists of an 8-bit unit identifier and an 8-bit entity identifier (see Figure 1-24 on page 1-38).

The unit identifier in the Source field contains an identifier used to indicate which physical unit (system unit or adapter) the entity sending the control element is located. The value assigned to the unit identifier is determined by the delivery service during initialization. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for additional information.)

The entity identifier in the Source field contains an identifier used to indicate which of the possible 256 entities in the source unit is the originator of the control element. The value assigned to the entity identifier is determined by unit-level management and is implementation dependent.

Destination Field

The Destination field is a 16-bit, word-aligned field used to identify the target or destination of a control element. Like the Source field, it is a structured field containing an 8-bit unit identifier and an 8-bit entity identifier (see Figure 1-24 on page 1-38).

The unit identifier part of the Destination field contains an identifier used to indicate the location of the destination entity (system unit or

adapter). The value assigned to the unit identifier is determined by the delivery service during initialization. (See "Configuration" on page 5-1 and "Initialization" on page 5-4 for additional information.)

The entity identifier part of the Source field contains an identifier used to indicate which of the possible 256 entities in the destination unit is the recipient of the control element. The value assigned to the entity identifier is determined by unit-level management and is implementation dependent.

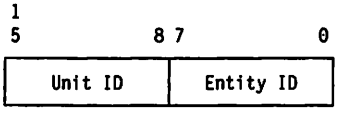


Figure 1-24. Control Element Source and Destination Fields

Correlation Field

The Correlation field is a 32-bit, word-aligned field used to provide an identifier for associating or correlating a reply control element, error control element, or an event control element with a previous request control element. The format, content, and use of the Correlation field are determined by the entity that originates the request control element. The field can contain a sequence number, the address of a control block, or the address of a data buffer. The content of the Correlation field is returned to the originator of a request in a reply, error, or event control element.

Entity to Entity Field

The Entity-to-Entity field of a control element is a variable-length field used to hold information such as status, parameters, or data required by the clients and servers to perform the operation identified by the function code in the Type field. The length, format, and type of information the field contains varies depending on the function specified. The information in the Entity-to-Entity field is meaningful only to the clients and servers and is defined as part of the entity-to-entity protocol.

Function Codes

Within the Common Indicators field of a Move mode control element, a function code is used to identify not only the operation to be performed but also to indicate how the contents of the Entity-to-Entity field should be interpreted. The high-order bit of the function code is used to distinguish between commonly-used function codes and implementation-defined function codes. When the high-order bit (6) is set to 1, it indicates that the remaining bits (5–0) contain an implementation-defined function code. When set to 0, bit 6 indicates that the remaining bits (0–5) contain one of the following commonly-used function codes.

Function Code	Name
1	Reserved
2	Initialize
3	Reserved
4	Read
5	Read List
6	Read Immediate
7	Write
8	Write List
9	Write Immediate
10	Execute List
11	Mark
12	Cancel
13	Reset
14	Read Configuration
15	Diagnose
16	Reserved
17	Resume
18	Notify
19	Inform
20-31	Reserved
32-39	Reserved
40	Reserved
41-62	Reserved
63	Wrap

Figure 1-25. Architected Function Codes in the Common Indicators Field

The resume, notify, inform, and wrap function codes are used exclusively with the event control element. "Event Control Elements" on page 1-62 provides a detailed description of these function codes. All other function codes are used with request, reply, and error control elements and are described in the next section.

Request, Reply, and Error Control Elements

The architecture defines a number of control elements for sending and receiving requests and replies. The following section describes these control elements. Each control element provides a set of primitives that can be used to create the specific protocol required for the task at hand. The rules determining protocol requirements such as the number, type, order of exchange, and required responses are determined by the two cooperating entities. This becomes the user-defined processing-level protocol.

Initialize

The Initialize function is used in request, reply, and error control elements. In a request control element, it invokes the Initialize function of the server. The Source, Destination, and Correlation Identifier fields, and Entity-to-Entity field, containing any additional parameters required by the server Initialize function, are passed to the server in the request control element.

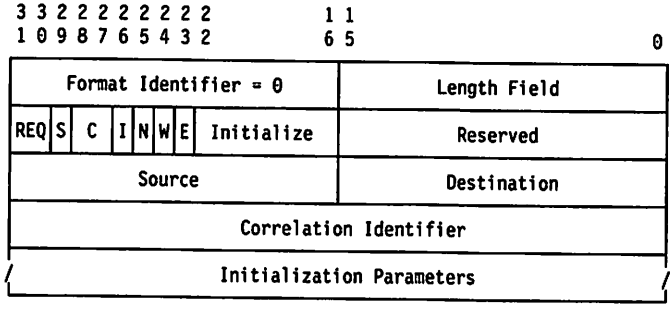


Figure 1-26. Initialization Request Control Element

The initialization parameters passed to the server in the Entity-to-Entity field of the request control element are server dependent and need to be defined on a server-by-server basis.

If the indirect bit is set in the Common Indicators field of the initialization request control element, then the Entity-to-Entity field does not contain the actual initialization parameters. Instead, it contains the location and the length of a data area in shared memory where they are stored.

A server reports the successful completion of an initialization request by returning a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields from the initialization request control element, along with any return values, are returned to the requestor in the reply control element.

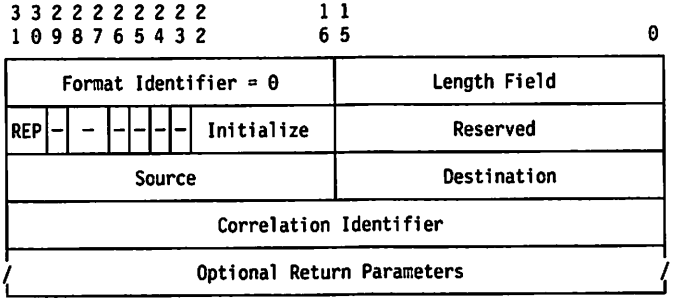


Figure 1-27. Initialization Reply Control Element

The optional return parameters passed in the Entity-to-Entity field of a reply control element are server dependent.

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the initialization request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier fields from the initialization request control element, along with status information identifying the cause of the failure, are returned to the requestor.

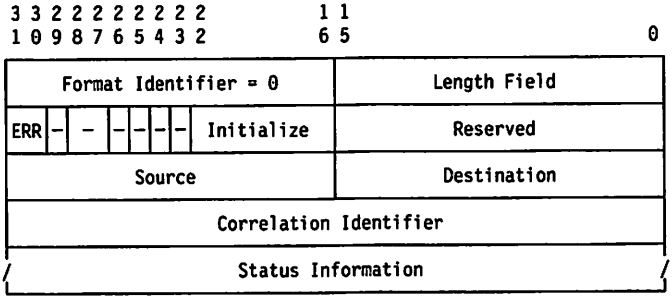


Figure 1-28. Initialization Error Control Element

Read

The Read function is used in request, reply, and error control elements. In a request control element, it is used to set up and initiate the transfer of data from a server to a client. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the parameters required by the server to perform the read, are passed to a server in the request control element.

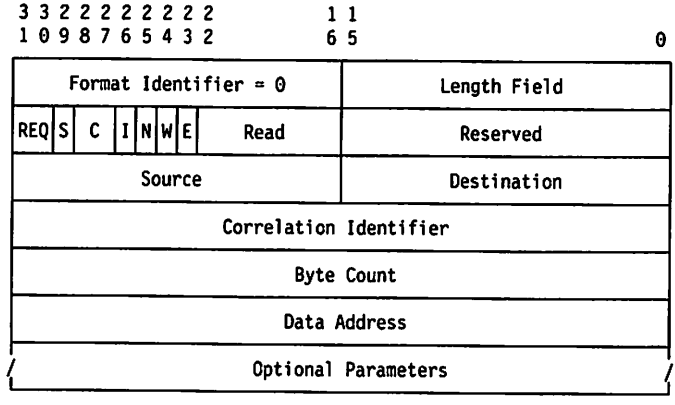


Figure 1-29. Read Request Control Element

The parameters contained in the Entity-to-Entity field of a request control element are not limited to those identified above. Additional parameters can be supplied to meet the requirements of each client and server relationship.

If the indirect bit is set in the Common Indicators field of the read request control element, the Entity-to-Entity field does not contain the byte count, data address, and optional parameters. Instead, it contains the location and the length of a data area in shared memory where these parameters are stored.

When a server has successfully completed the transfer, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the read request control element and the actual number of bytes transferred during the read operation.

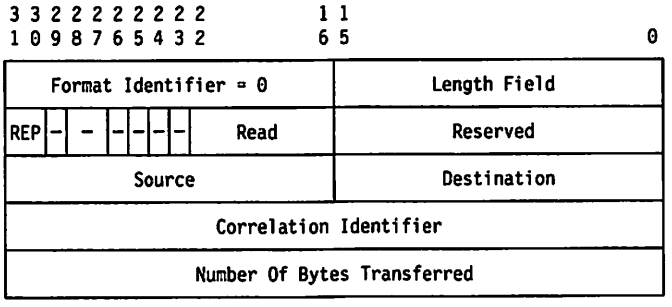


Figure 1-30. Read Reply Control Element

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the read request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier from the read request control element, along with status information identifying the cause of the failure, is returned to the requestor.

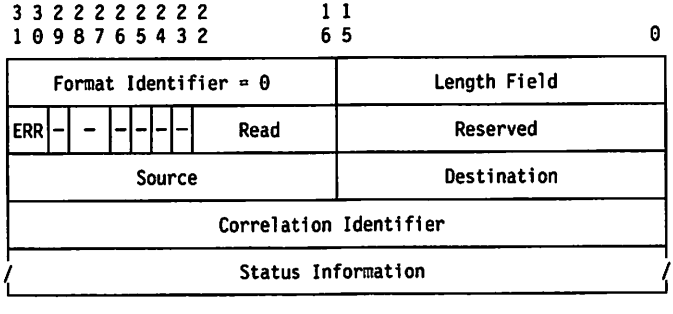


Figure 1-31. Read Error Control Element

Read List

The Read List function is used in request, reply, and error control elements. It is used to establish and initiate the transfer of data and control information from a server into several, possibly noncontiguous, areas in shared memory. This is often referred to as data chaining. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the list of byte counts and data addresses required by the Read List function, are passed to the server in the request control element.

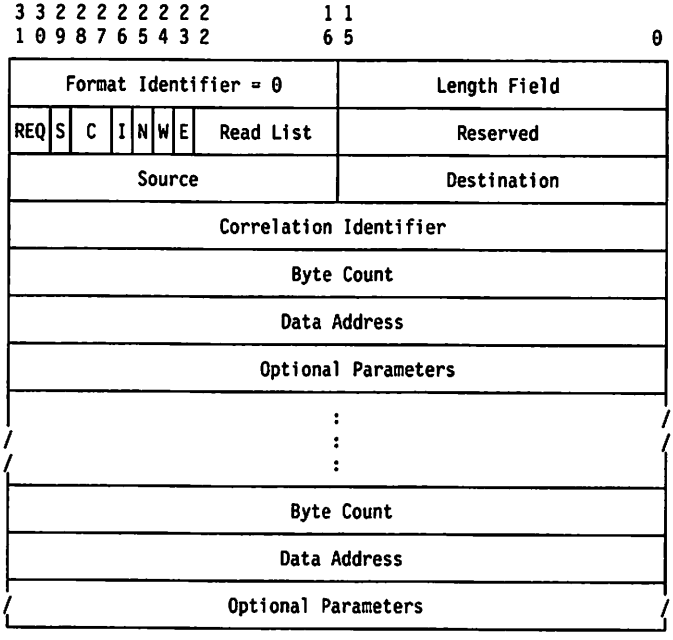


Figure 1-32. Read List Request Control Element

If the Indirect list is indicated in the Common Indicators field, the Entity-to-Entity field does not contain the byte count, data address, or optional parameters. Instead, it contains the location and the length of an area in shared memory where the actual list is stored.

When a server has successfully completed the transfer, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the read list request control element, and residual-byte count in the reply control element. The format and content of the read list reply control element is the same as the read reply control element (see "Read" on page 1-42).

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the read list request control element.

If unsuccessful, the following are returned to the requester in an error control element.

- The Source, Destination, and Correlation Identifier fields from the read list request control element

- The status information identifying the cause of the failure
- The pointer to the last data address that was processed
- The actual number of bytes read.

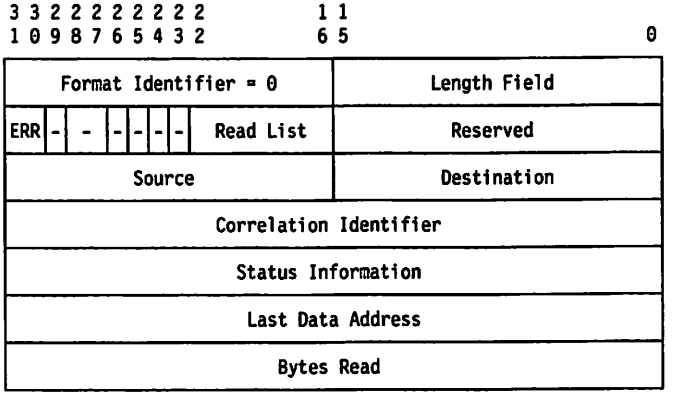


Figure 1-33. Read List Error Control Element

Read Immediate

The Read Immediate function is used in request, reply, and error control elements. It is used to request the transfer of data and control information from a server to a client. It differs from the Read Request function in that the data is to be returned in the Entity-to-Entity field of the reply control element where it becomes immediately available to the client. The Source, Destination, Correlation Identifier fields, and the Entity-to-Entity field, containing the parameters required by the Read Immediate function, are all passed to a server in the request control element.

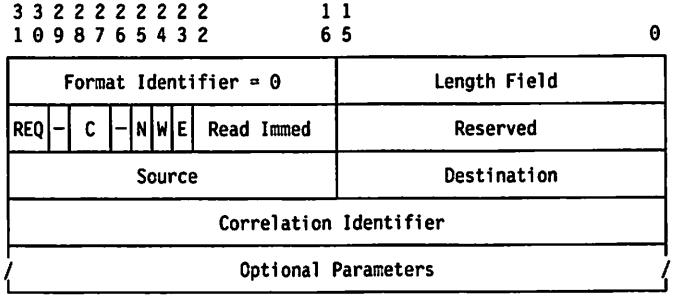


Figure 1-34. Read Immediate Request Control Element

Note: The amount of data that can be returned using the read immediate request control element is configuration dependent. That is, it is directly related to the size of the delivery pipe (queue). Therefore, care should be exercised when using this request control element.

To complete the requested operation, the server returns a reply control element with the Source, Destination, and Correlation Identifier fields from the read immediate request control element. In the Entity-to-Entity field, it returns the immediate data.

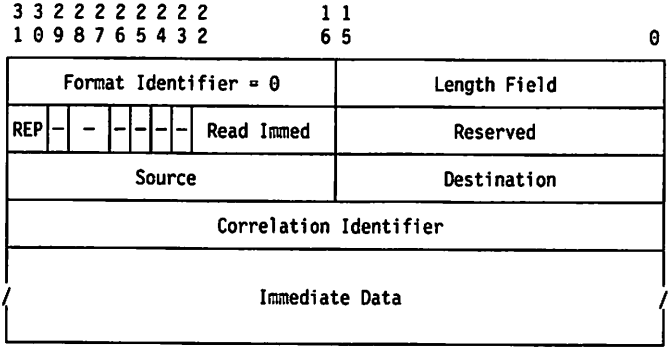


Figure 1-35. Read Immediate Reply Control Element

The Read Immediate function ignores the suppress bit if it is set in the Common Indicators field of the read immediate request control element because it always returns a reply control element.

If unsuccessful, an error control element containing the Source, Destination, Correlation Identifier fields from the read immediate request control element, and status information identifying the cause of the failure, are returned to the requestor.

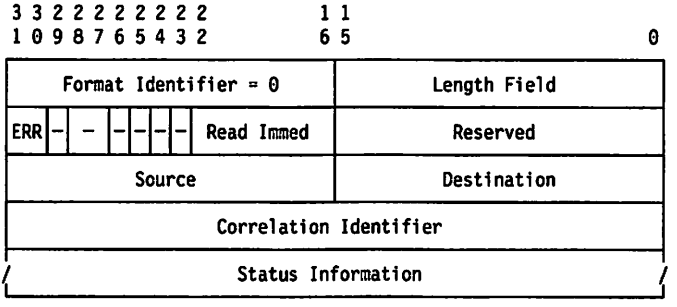


Figure 1-36. Read Immediate Error Control Element

Write

The Write function is used in request, reply, and error control elements. It is used to set up and initiate the transfer of data and control information from a client to a server. The Source, Destination, Correlation Identifier, and the Entity-to-Entity fields are all passed to a server in the request control element.

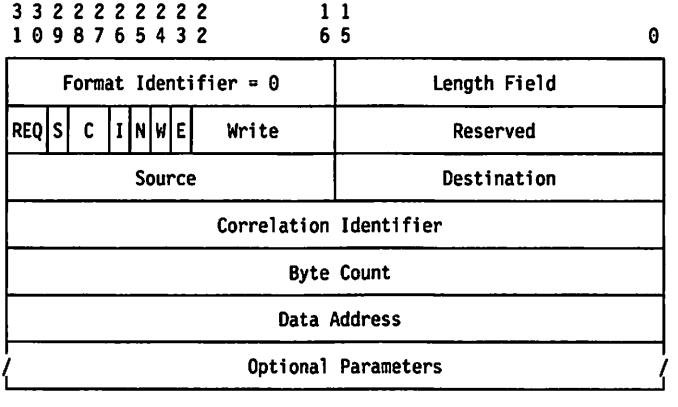


Figure 1-37. Write Request Control Element

The Entity-to-Entity field contains the amount and location of the data to be transferred by the Write function. The Entity-to-Entity field is not limited to these parameters. Additional parameters can be supplied to meet the requirements of each client and server relationship.

If the indirect bit is set in the Common Indicators field of the write request control element, the Entity-to-Entity field does not contain the byte count, data address, or optional parameters. Instead, it contains

the location and length of the area in shared memory where data is stored.

When the server completes the transfer, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the write request control element.

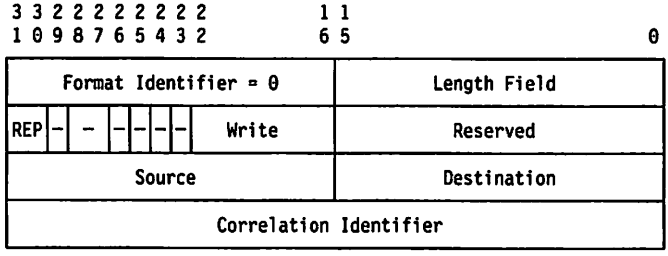


Figure 1-38. Write Reply Control Element

A client can suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the write request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier fields from the write request control element, and status information identifying the cause of the failure, is returned to the requestor.

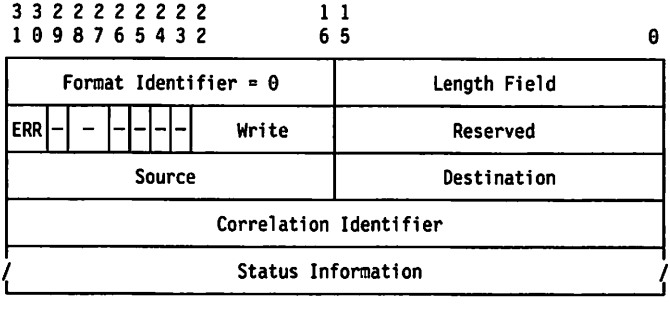


Figure 1-39. Write Error Control Element

Write List

The Write List function is used in request, reply, and error control elements. It is used to set up and initiate the transfer of data and control information to a server from several possibly noncontiguous areas of shared memory. This is often referred to as data chaining. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the list of byte counts, data addresses, and optional parameters required by the Write List function, are all passed to the server in the request control element.

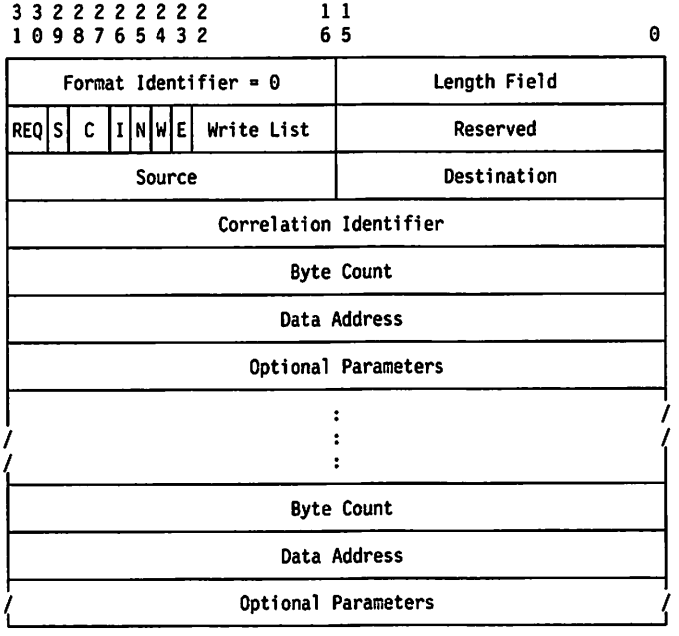


Figure 1-40. Write List Request Control Element

If the indirect bit is set in the Common Indicators field of the write list request control element, the Entity-to-Entity field does not contain the byte count, data address, and optional parameters. Instead, it contains the location and length of the area in shared memory for the list.

When the server has completed the transfer, it returns a reply control element with the Source, Destination, Correlation Identifier fields from the write list request control element, and residual-byte count in the Entity-to-Entity field of the reply control element. The format and

content of the write list reply control element is the same as the write reply control element (see "Write" on page 1-47).

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the write list request control element.

If unsuccessful, the following are returned to the requester in an error control element:

- The Source, Destination, and Correlation Identifier fields from the write list request control element
- The status information identifying the cause of the failure
- The pointer to the last data address that was processed
- The actual number of bytes transferred.

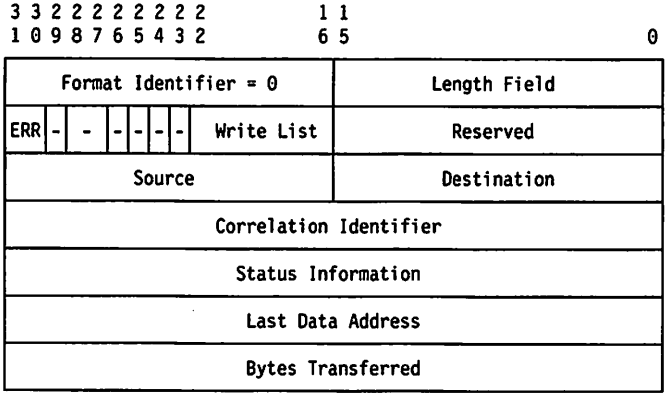


Figure 1-41. Write List Error Control Element

Write Immediate

The Write Immediate function is used in request, reply, and error control elements. It is used to carry data and control information from a client to a server. It differs from the write request in that the data is present in the Entity-to-Entity field of the request control element and is immediately available to the server. The Source, Destination, and Correlation Identifier fields, and data are passed to a server in the request control element.

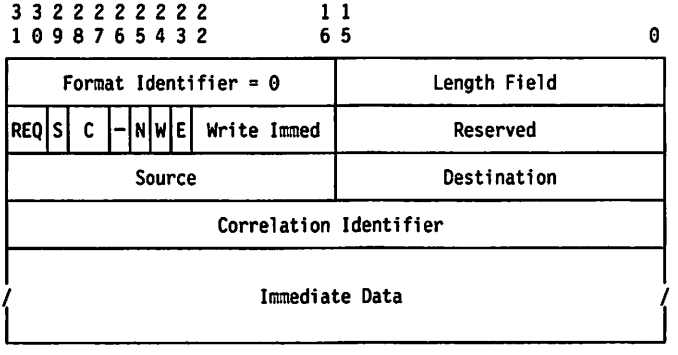


Figure 1-42. Write Immediate Request Control Element

The format and meaning of the data contained in the Entity-to-Entity field of the request control element are client and server dependent and, as such, are not defined here.

Note: The amount of data that can be transferred using the write immediate request control element is configuration dependent. That is, it is directly related to the size of the delivery pipe (queue). Therefore, care should be exercised when using this request control element.

When the server has successfully received the data, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the write immediate request control element and, in the Entity-to-Entity field, a count of the actual number of bytes received.

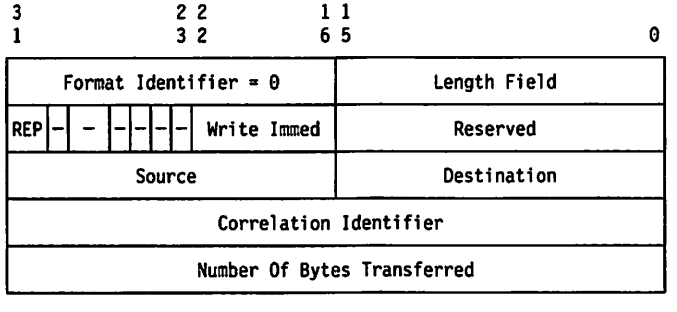


Figure 1-43. Write Immediate Reply Control Element

A client can suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the write immediate request control element.

If unsuccessful, an error control element is returned to the requestor. This control element contains the Source, Destination, and Correlation Identifier fields from the write immediate request control element, and status information identifying the cause of the failure.

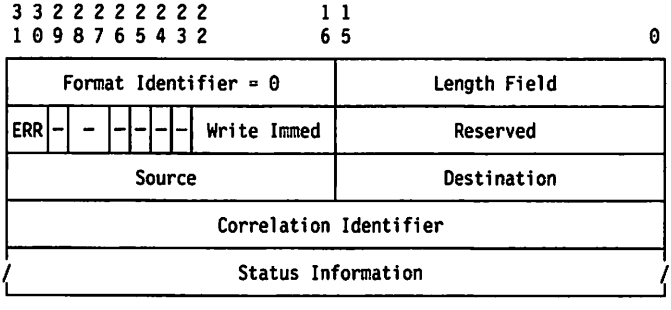


Figure 1-44. Write Immediate Error Control Element

Execute List

The Execute List function is used in request and reply control elements. It is used to set up a repetitive loop for executing a list of request control elements one or more times. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the location, length, and repetition count required by the Execute List function are all passed to a server in the request control element.

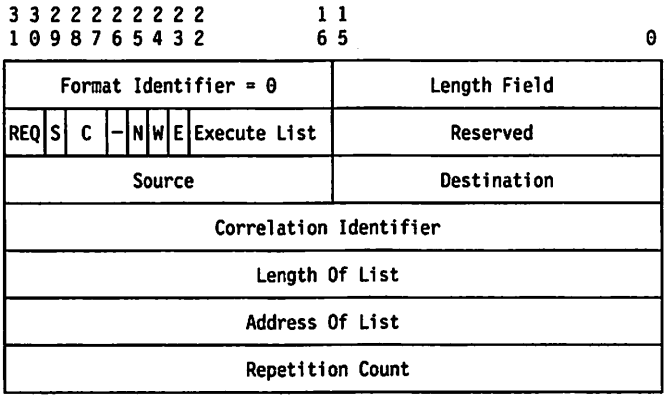


Figure 1-45. Execute List Request Control Element

The list can consist of one or more request control elements that must be contiguous in shared memory. Because each control element in the list contains its own length field, the start of the next control element can be determined. The Length of List field determines where the list ends. On each pass through the list, the repetition count is decremented by 1, and if not 0, the list is executed again. If the initial repetition count is negative or 0, the request terminates.

The request control elements contained in the list can use any available function code and common indicator.

When the repetition count goes to 0, the list is terminated and the server returns a reply control element with the Source, Destination, and Correlation Identifier fields from the original execute list request control element.

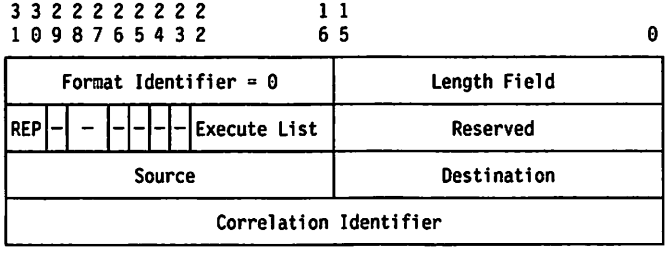


Figure 1-46. Execute List Reply Control Element

A client can choose to suppress the returning of a reply control element by setting the suppress bit in the Common Indicators field of the execute list request control element.

If an error occurs while processing any control element within the list, the list is terminated and the control element with the error is returned.

Mark

The Mark function is used only in a request control element to provide a means of synchronization between a client and a server. Its only function is to pass the Correlation Identifier field to the server.

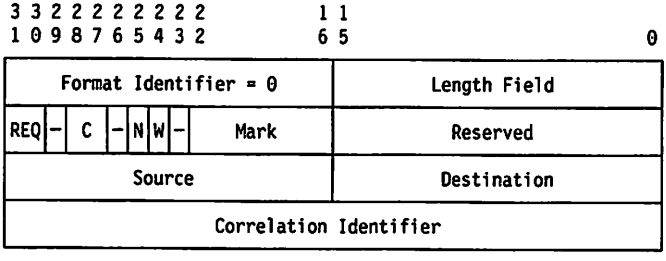


Figure 1-47. Mark Request Control Element

The Mark function can utilize the notify and wait bits to cause a server to inform a client of the current processing state of the server. For more details refer to "Notification and Wait" on page 4-11.

The Mark function ignores the indirect, suppress, and expedite bits if set in a mark request control element.

Cancel

The Cancel function is used in the request, reply, and error control elements. In a request control element, it is used to cancel one or more outstanding requests. The Source, Destination, Correlation Identifier, and Entity-to-Entity fields contain the cancellation qualifier and the cancellation list of request control element correlation identifiers.

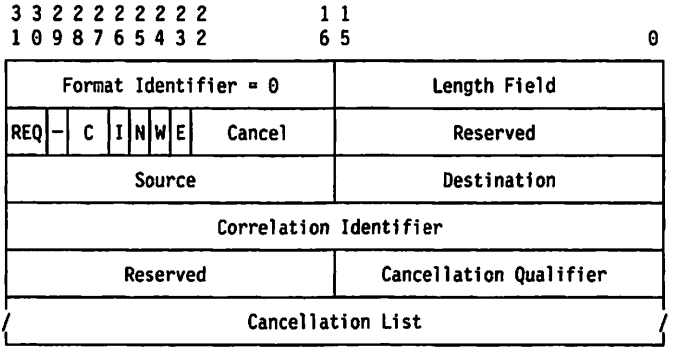


Figure 1-48. Cancel Request Control Element

The cancellation qualifier bits identify the request control elements to be cancelled:

- All outstanding requests (bit 0 = 1)
- Requests that match the Source ID (bit 1 = 1)
- First request matching the cancellation list value (bit 2 = 1)
- All requests matching the cancellation list value (bit 3 = 1).

If bits 2 or 3 of the cancellation qualifier are set to 1, a cancellation list is present. If bit 1 is set, then bit 2 or 3 must also be set.

If the indirect bit is set in the Common Indicators field of the cancel request control element, the Entity-to-Entity field does not contain the cancellation list. Instead, it contains the location and length of the area in shared memory for the list.

When the server has successfully completed the Cancel function, it returns a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields from the cancel request control element are returned in the reply control element.

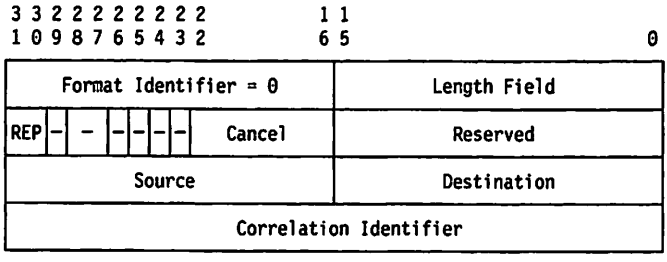


Figure 1-49. Cancel Reply Control Element

The server tries to cancel all the requests whose Correlation Identifier fields are in the Entity-to-Entity field of the cancel request. If the server cannot cancel all requests, it returns an error control element with the status and the list of Correlation Identifiers fields from the cancelled requests. An empty list indicates that it was unable to perform the cancel on any of the identified control elements.

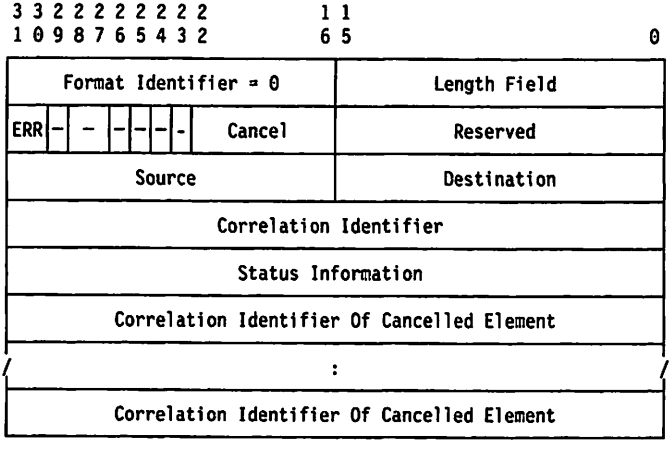


Figure 1-50. Cancel Error Control Element

However, a cancel request control element with the *all outstanding control element* qualifier set does not return an error control element with the list of cancelled control elements.

Reset

The Reset function is used in request, reply, and error control elements. In a request control element, it is used to request a server to place itself into a known state. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the desired state, are all passed to a server in the request control element.

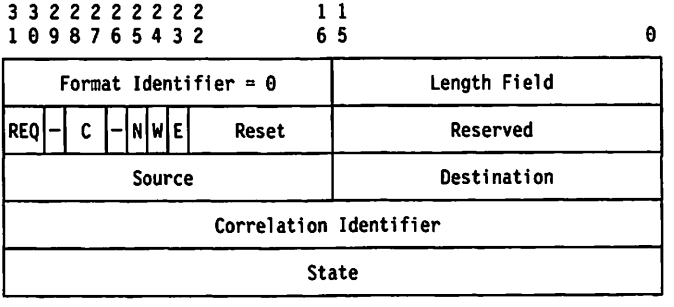


Figure 1-51. Reset Request Control Element

A reset request control element with a value of 0 for the State field requests the device to return to the initialization state. All other values are implementation dependent.

When reset to the initialization state, the server purges all work in all queues, and any outstanding requests are terminated with an error control element. All requests except for the initialize control element are rejected with an error control element when in the reset-to-initialization state.

When the server has completed the reset, it returns a reply control element with the Source, Destination, and Correlation Identifier fields from the request control element to the client.

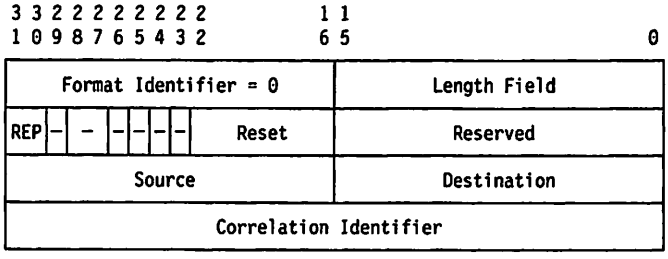


Figure 1-52. Reset Reply Control Element

The Reset function ignores the suppress and indirect bits, if the bits are set in a reset request control element.

If unsuccessful, an error control element containing the Source, Destination, and Correlation Identifier fields from the reset request control element, and status information identifying the cause of the failure, are returned to the requestor.

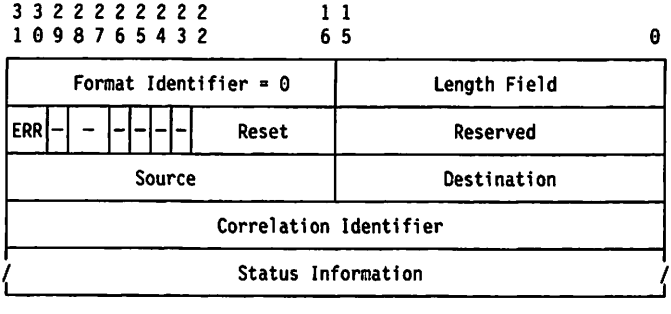


Figure 1-53. Reset Error Control Element

Read Configuration

The Read Configuration function is used in request, reply, and error control elements. When used in a request, it causes the server to return configuration information to the client. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field containing the byte count, data address, and optional parameters are passed to the server in the request control element.

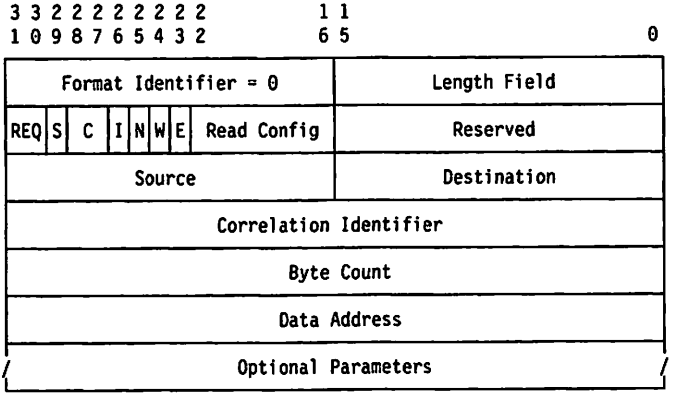


Figure 1-54. Read Configuration Request Control Element

The parameters contained in the Entity-to-Entity field of a read configuration request control element are not limited to those identified above. Additional parameters can be supplied to meet the requirements of each client and server relationship.

If the indirect bit is set to 1 in the Common Indicators field of the read configuration request control element, the Entity-to-Entity field does not contain the byte count, data address, and optional parameters. Instead, it contains the location and the length of the data area in shared memory where the parameters are stored.

A server reports the successful completion of a read configuration request by returning a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields are returned in the reply control element.

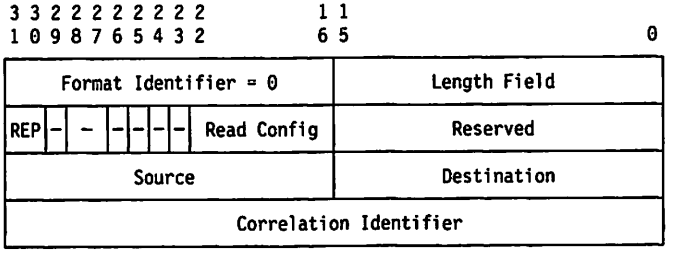


Figure 1-55. Read Configuration Reply Control Element

A client can suppress returning a reply control element by setting the suppress bit to 1 in the Common Indicators field of the read configuration request control element.

If the server is unable to report configuration information, an error control element is returned to the requestor. This control element contains the Source, Destination, and Correlation Identifier fields from the read configuration request control element, along with the reason.

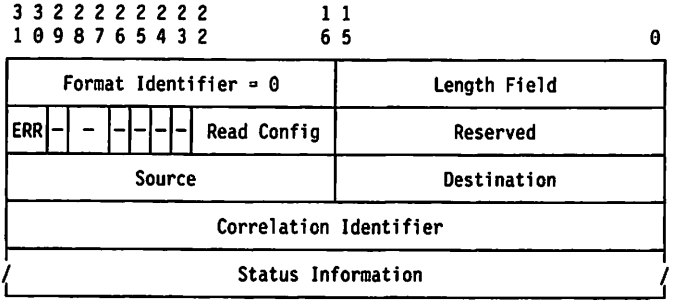


Figure 1-56. Read Configuration Error Control Element

Diagnose

The Diagnose function is used in request, reply, and error control elements. In a request, the Diagnose function initiates diagnostic routines by a server. The Source, Destination, and Correlation Identifier fields, and the Entity-to-Entity field which identifies the specific diagnostic tests to be run, as well as any operational parameters, are passed to the server in the request control element.

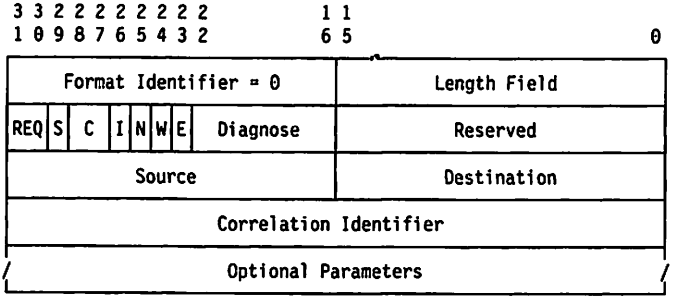


Figure 1-57. Diagnose Request Control Element

If the indirect bit is set in the Common Indicators field of the diagnose request control element, the Entity-to-Entity field does not contain the optional parameters. Instead, it contains the location and length of the area in shared memory for the parameters.

A server reports the results of running the diagnostic tests by returning a reply control element to the requestor. The Source, Destination, and Correlation Identifier fields from the diagnose request control element, and the Entity-to-Entity field containing the test results, are returned to the reply control element.

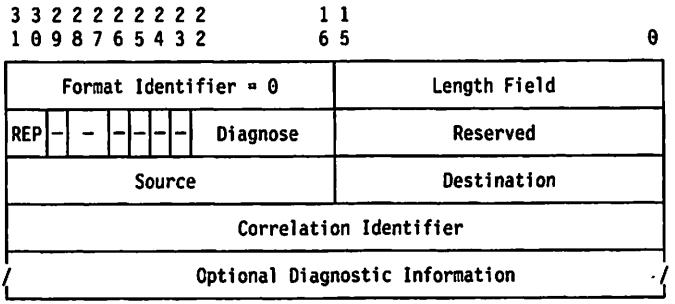


Figure 1-58. Diagnose Reply Control Element

A client can suppress returning a reply control element by setting the suppress bit in the Common Indicators field of the diagnose request control element.

If a server is unable to perform the requested diagnostic test, an error control element is returned and the Entity-to-Entity field indicates the cause of the error. This error control element contains the Source, Destination, and Correlation Identifier fields from the diagnose request control element.

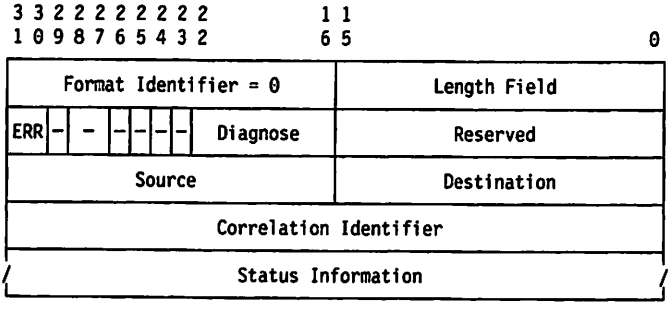


Figure 1-59. Diagnose Error Control Element

Event Control Elements

The event control element is provided by the delivery service in addition to the request, reply, and error control elements. It provides information about the progress of a request, or about the side-effects of a previous request. The event control element differs from the other control elements in that it reports a change in the state of one of the communicating entities (client or server). Because of the impact of these state changes, the entity must be informed of the change as quickly as possible. The following sections describe the format, content, and use of event control elements.

Resume

A resume event is used to notify a server of a change in a clients processing state. A client uses this event to notify a server that a client is now in a state that allows the processing of request control elements to be resumed. The Source, Destination, and Correlation Identifier fields of the original request control element causing the suspension are returned to the server in the resume event control element.

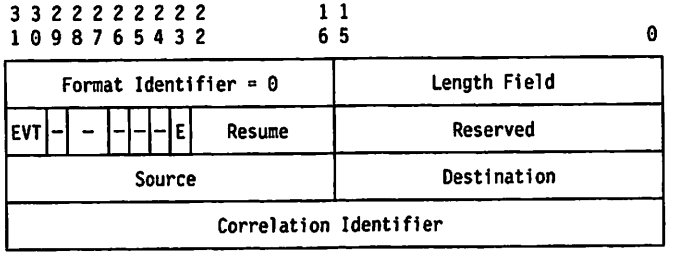


Figure 1-60. Resume Event Control Element

A resume event is ignored if it is received from a client other than the one responsible for the suspended state or from a client that is not in the suspended state.

Notification

A notification event is used to confirm the receipt of a request control element with the notification bit of the Common Indicators field set to 1 (see "Notification and Wait" on page 4-11). The Source, Destination, and Correlation Identifier fields of the request control element containing the notification bit, are returned to the client in the Entity-to-Entity field of the event control element.

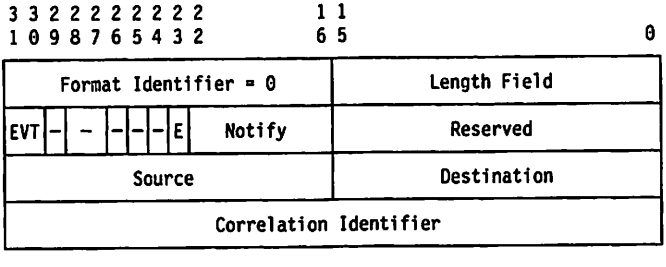


Figure 1-61. Notification Event Control Element

Inform

An inform event is used to provide a means of conveying information regarding the processing state of either a client or a server from one to the other.

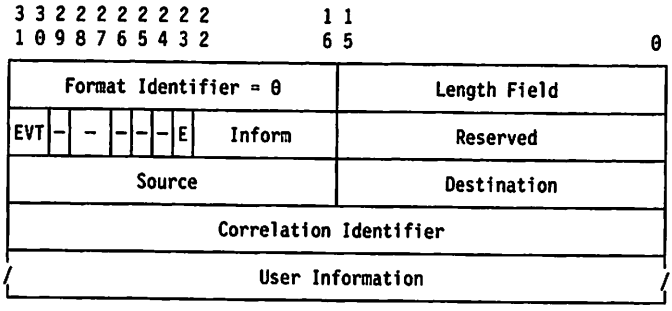


Figure 1-62. Inform Event Control Element

Wrap

A wrap event is a management and control-initiated event used internally by the delivery service. It is used to synchronize the operations associated with the in-bound or out-bound delivery pipes.

A wrap event control element can be sent only by the delivery service. This control element informs the delivery service that there was insufficient space in the pipe for the next control element. Therefore, the control element is placed in the start of the pipe.

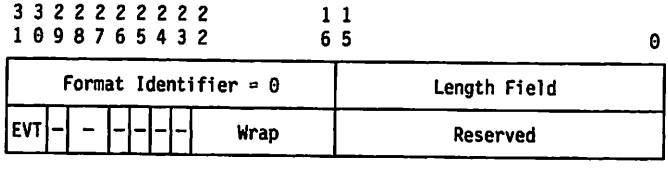


Figure 1-63. Wrap Event Control Element

Chapter 2. Physical Level

At the physical level, Move mode defines the services, functions, and protocols that are used by the delivery-support logic to physically manage control-element delivery between units attached to the channel.

The physical level provides a set of services for gaining access to I/O and memory address space, and for signalling between units that are attached to the channel that is independent of the hardware. This allows different forms of system unit and adapter hardware to be used by the same delivery-support logic.

The physical level allows for differences in the way services are provided by various system units and adapters. The location and use of control areas; the method of generating and handling interrupts; and the protocols used to access I/O and memory address space are transparent to the logic at the delivery level. Only the physical level needs to be aware of how the services are mapped to the underlying hardware.

Physical-level services can be used to push or pull data between local memory address space and control areas in shared memory. The data can be signalling information, surrogate-state information, a control element, or data pointed to by the control elements.

Structure

The following figure shows the physical-level structure for a configuration with a single system unit (Unit X) and two adapters (Unit Y and Unit Z). The services of the physical level are provided by a combination of hardware, physical-support logic in each unit, control areas in I/O address space, and control areas (signalling) in shared memory.

© Copyright IBM Corp. 1991.

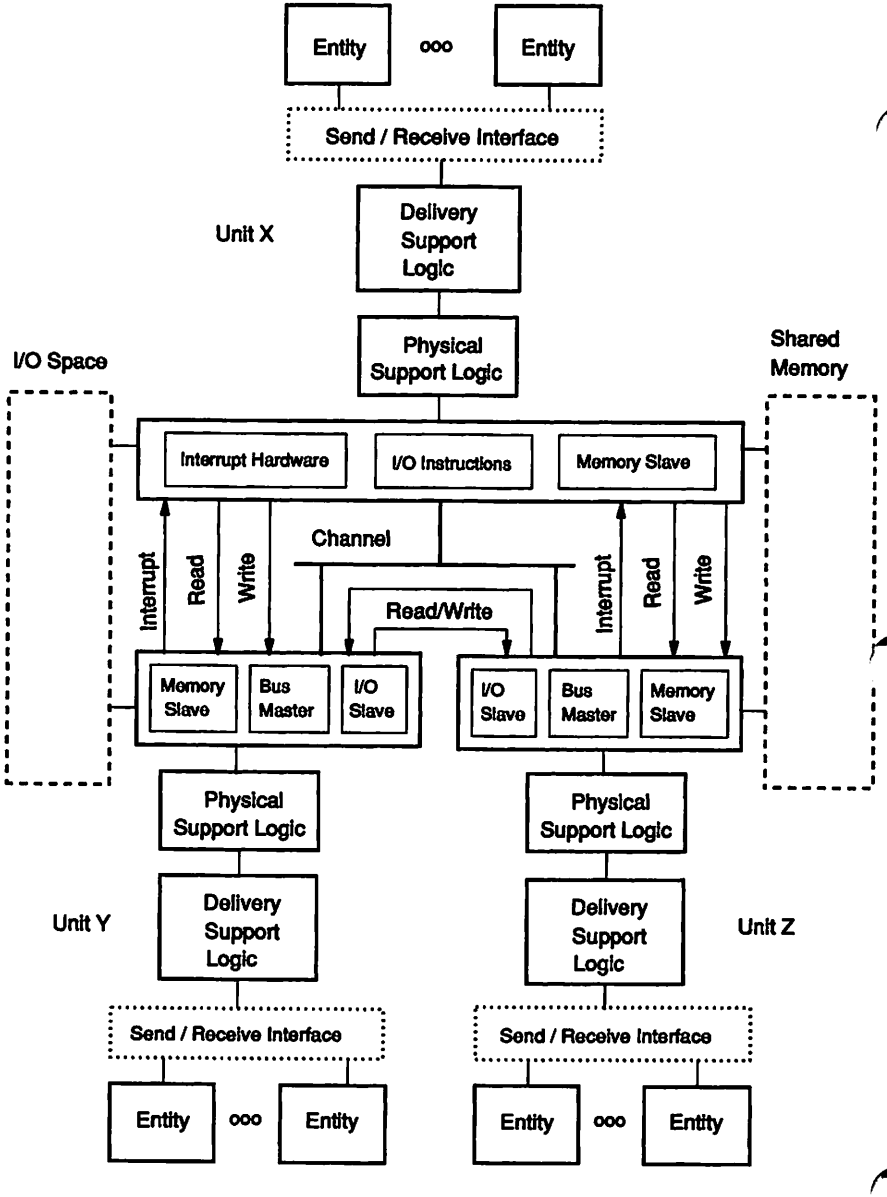


Figure 2-1. Physical Level

System

System unit designs typically provide support for system interrupts, memory slaves, and I/O slaves.

The interrupt controller provides hardware support for signalling system interrupts from an adapter to a system unit. The memory-access instructions (such as Load and Store) provide a means for gaining access to data and control information located in shared memory. The I/O-access instructions (such as In and Out) provide a means for gaining access to control information located in control areas in I/O address space. The bus-master support allows adapters to control the channel.

Adapter

Adapter designs typically provide support for I/O slaves, bus masters, and optional memory slaves. The I/O slave provides hardware support for initializing an adapter, enabling access to an adapter, signalling between adapters, and identifying the adapter that is the source of an interrupt. The memory slave provides hardware support for gaining access to data and control areas in shared memory from a system unit or from another adapter with bus-master capabilities. The bus master provides hardware support for gaining access to data and control areas in either shared or local memory and control areas in I/O address space in other system units or adapters.

Support Logic

The support logic is responsible for implementing the set of services provided to the delivery level for the specific hardware used by an adapter or system unit. The protocols used between two units are determined by the hardware and must be implemented by the support logic in the source and destination units. Support logic must be provided for the following services: push, pull, and signal.

Push and Pull

The support logic for the push and pull service must establish the addresses and the method to be used to move the data into or out of shared memory. The protocols can vary for different combinations of system units and adapters. The following sequence is used to establish the addressing for performing the Push or Pull functions.

- During system configuration, the specific I/O and shared memory address ranges are defined for the various units.
- During initialization, specific control areas in shared memory are established. Initialization includes establishing any address translations between the channel address form and the local memory address form.
- Adapters can establish the addresses just before performing the push or pull operation. In this case, establishing the address can be done locally or it can be visible on the channel.

The data can be moved in one of several ways, depending on the functions available in the adapter. The data migration includes hardware to provide an interface from the unit to the channel, as well as providing the following functions.

- If a unit (such as a system) has hardware that supports memory-mapped I/O instructions, these instructions are used to push and pull data.
- If a unit (such as an adapter) has bus-master capabilities, it is used to push and pull data between units.

The hardware in each unit that is used to support the push and pull operations can also be shared with the protocols that move data that is pointed to by the control elements.

The architecture requires control areas in memory to be aligned on word (2-byte) boundaries so that they are accessed as words in shared memory. This eliminates the possibility of having word-aligned surrogate information split when it is accessed by the delivery-support logic.

Signal

The context for describing the signalling logic is shown in Figure 2-2 on page 2-6. The signalling service is supported by a combination of hardware and support logic in the sending and receiving units. The support logic in the receiving unit uses the information in the Signalling Control areas to determine which units have interrupts pending and the reasons for those interrupts.

The send-signal logic and receive-signal logic operate in support of the signalling protocols defined at the delivery level.

- The send-signal logic is used by the delivery or management-support logic to cause the interrupt of another unit after it has pushed signalling information into the Signalling Control area assigned to it in the other unit.
- The receive-signal logic examines all Signalling Control areas whenever a hardware interrupt is received. This ensures that all signalling indications are captured even though an interrupt could have been missed by a hardware-busy or masked condition.

  Using the information in the Signalling Control areas, the receive-signal logic posts the appropriate delivery-support logic at the delivery level and clears the contents of the Signalling Control area to 0. For management, it posts the unit-management logic and resets the management indicator in the appropriate Signalling Control area.

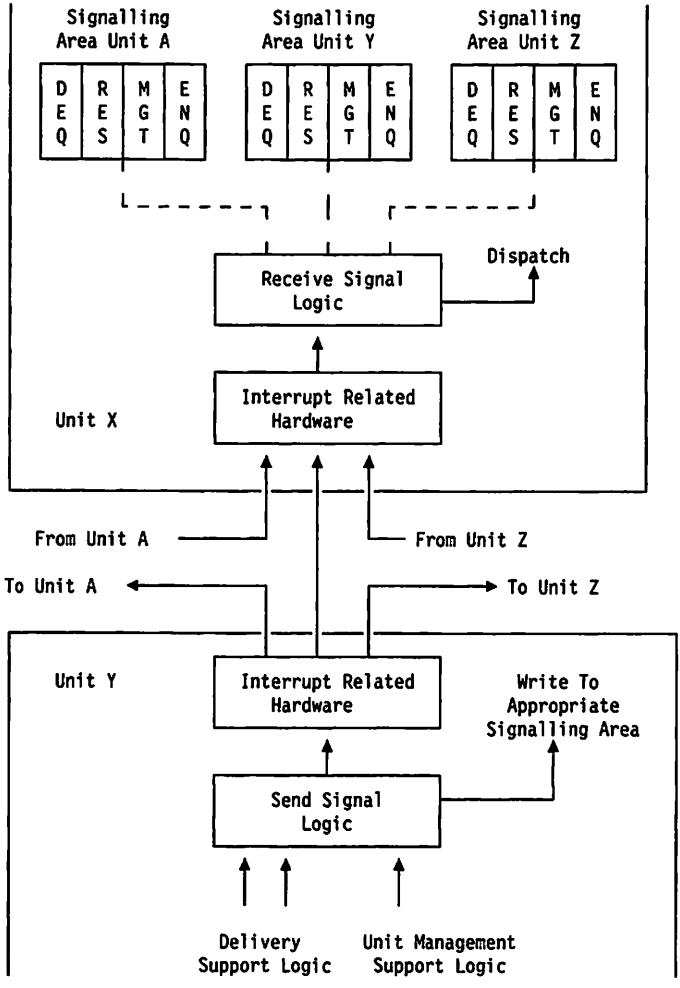


Figure 2-2. Signalling

The signalling service operates in a variety of physical implementations by using Signalling Control areas in shared memory and simple signalling primitives. To support peer-to-peer delivery, each sending unit has a separate Signalling Control area that identifies it to the receiving unit. An indication in a particular Signalling Control area defines which unit is signalling. The receiving unit has a Signalling Control area for each unit it communicates with.

The structure and content of the Signalling Control area is described in detail in "Signalling Control Area" on page 1-29.

The support logic for the signalling portion can be viewed as having a part that establishes access to the interrupt hardware and a part that causes an interrupt in the destination unit.

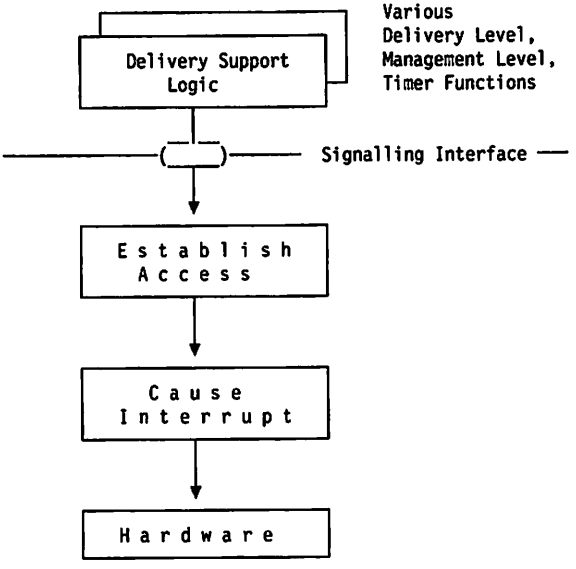


Figure 2-3. Interrupt Support Logic

The protocols can vary for differences:

- Between unit types (adapters and systems)
- In types of adapters (such as LAN, COMM, and SCSI)
- In the processor architectures used for the specific hardware.

Access to the interrupt hardware is established at specific times:

- During system configuration, the interrupt level is allocated along with the I/O addresses used by the adapter.
- During adapter initialization, the locations of the specific control areas, including the Signalling Control area, are established. The type of unit determines the location of the control area and the value that causes an interrupt.

- During operation, when the interrupt is shared, then some form of interrupt-sharing protocol could be required.

The specific protocol for generating the interrupt is determined by the type of unit being signalled.

- If the unit is a system unit, an interrupt is caused by hardware (IRQ) lines in the channel that connect to the interrupt controller hardware.
- If the unit is an adapter, an interrupt is caused by writing a predefined value into a specific control area (port) in I/O address space. The value and location are specified as part of the configuration and initialization process.

Memory Address Space

The physical level provides access to data and control areas in shared memory.

The physical-level and delivery-level protocols in the source and destination units must have access to (and in some instances share) data and control information in the same address space.

At the delivery level, this includes the control elements in the data areas associated with each delivery pipe, as well as their associated Surrogate Control areas.

At the physical level, this includes the Signalling Control areas associated with each unit (system and adapter). The information in the Signalling Control area is also accessed by the support logic implementing the delivery-level protocol in the source and destination units.

Assigning these control areas to locations in shared memory allows them to be accessed, and the information they contain to be shared, by different units on the channel.

The data buffers used to support certain forms of entity-to-entity data transfers (defined by addresses within specific request and reply control elements) are also assigned locations in shared memory. The location of these data buffers is determined by the logic at the

processing level, which supports entity-to-entity data transfers, and can change from control element to control element.

The shared memory structure utilizes the physical addresses on the channel to define the address space that is shared between system units and adapters.

The control areas in shared memory are used for the delivery pipes and their associated Surrogate Control areas. The location of each control area is assigned either at system configuration or adapter initialization time and remains constant throughout the operation of the system. The shared memory can be physically located in either the system unit, the adapter, or both. Each of these areas is accessed using the memory-slave support in their respective units.

The physical-support logic handles the access to all control information in the various shared memory locations using the appropriate hardware for the specific unit (system unit or adapter) performing the access.

Note: The ability to assign shared memory to various units and the ability to assign control areas to any location in shared memory allows for various cost-performance tradeoffs to be implemented using the same delivery-level and processing-level protocols.

I/O Address Space

The physical-support logic provides access to a set of control areas (ports) in I/O address space that are confined to, and shared at, the physical level between system units and adapters. This space is shared across units and is supported by the I/O slave support in adapters.

The range of I/O addresses assigned to each adapter is defined by programmable options select (POS) parameters provided during system setup. The location and use of control areas in this I/O address space for a given adapter type can be found in "I/O Address Space" on page 1-15.

Physical Level Services

Operations provided by the physical level are used to access control information in locations in shared memory or I/O address space and to support the sending and receiving of signals by any system unit or adapter on the channel. The operations provided are:

- Push
- Pull
- Signal.

The delivery-support logic uses the I/O address space to move control parameters between the system unit and adapter. It uses the shared memory to move state information and variable-length control elements between delivery-support logic in different units (system unit as well as adapters).

A system unit can use processor memory-mapped I/O or programmed I/O instructions to access control areas in I/O address space.

In an adapter, these operations can be provided by memory-mapped instructions or by bus-master operations, depending upon the specific system unit or adapter implementation.

Push

The push operation moves control information from a location in the caller's memory to a location in shared memory.

Note: The caller's memory can be located in local address space or in shared memory address space.

The push operation is an internal function and is not defined by the architecture. However, the following conceptual service primitive illustrates the parameters required when invoking the service:

```
PUSH (unit id, address space, from address, to address, length, return code)
```

Unit ID This parameter identifies the system unit or adapter that is the target of the push operation.

Address space This parameter indicates whether the to address argument refers to a location in shared memory or I/O address space.

From address This parameter identifies the location in the caller's memory that is the source of the information to be pushed into the location in shared memory or I/O address space indicated by the to address parameter.

To address This parameter identifies the location in shared memory or I/O address space that is to be the destination for the information indicated by the from address parameter.

Length This parameter indicates the number of bytes to be pushed into the location in shared memory or I/O address space identified by the to address parameter.

Return code This parameter indicates the success or failure of the push operation.

Pull

The pull operation moves information from a location in I/O or shared memory to a location in the caller's memory.

Note: The caller's memory can be located in local address space or shared memory address space.

The pull operation is an internal function and is not defined by the architecture. However, the following conceptual service primitive illustrates the parameters required when invoking the service:

```
PULL (unit id, address space, from address, to address, length, return code)
```

Unit ID This parameter identifies the system unit or adapter that is the target of the pull operation.

Address space This parameter indicates whether the from address argument refers to a location in shared memory or I/O address space.

From address This parameter identifies the location in I/O space or shared memory that is the source of the information to be pulled into the caller's memory as indicated by the to address parameter.

To address This parameter identifies the location in the caller's memory that is the destination for the information indicated by the from address parameter.

Length This parameter indicates the number of bytes to be pulled from the location in I/O or shared memory identified by the from address parameter.

Return code This parameter indicates the success or failure of the pull operation.

Signal

In addition to the push and pull operation primitives, the physical level also provides a service primitive for signalling between units. The physical-level logic at the source unit maps the signalling primitive into the appropriate set of hardware signals to cause an interrupt at the destination unit (system unit or adapter).

The signal operation is an internal function and is not defined by the architecture. However, the following conceptual service primitive illustrates the parameters required when invoking the service.

```
SIGNAL (unit id, area, value, return code)
```

Unit ID This parameter identifies the system unit or adapter.

Area This parameter identifies the specific area within the Signalling Control area (the enqueue area, the dequeue area, or the management area).

Value This parameter identifies the value to be written to the specified area.

Return code This parameter indicates the success or failure of the signal operation.

Data Delivery

In adapters, bus-master support can also be used to transfer data to or from entities in the adapter. The request and reply control elements contain pointers to the location in shared memory of the data to be accessed by the adapter entities. The push and pull operations can be used by the entities at the processing level, and the support-logic at the delivery level, to access data in shared memory.

Physical Level Protocols

This section defines the architecture for the various types of physical-level protocols supported. The definition for each type includes the protocols for both system unit to adapter and for adapter to adapter.

Control Areas

The physical level uses the following parameters that are part of the system configuration:

- Memory address space
- I/O address space
- Interrupt level
- Arbitration level.

The assignment and uses of specific control areas in I/O space for each interface type are included as part of the architecture definition of each unit type.

Signalling Control areas use shared-memory and are used by the delivery level in conjunction with the signalling protocols. These operations are defined in "Signal" on page 2-5.

Physical Interface

An adapter can support either the Locate or Move mode architecture using hardware with the same set of control areas in I/O address space and with setup support to tailor the operation of the interrupt-valid bit and reset the protocol to match either mode.

At the physical level, the Move mode architecture defines the number and location of the control areas in I/O address space and the rules or protocols for their use in:

- Establishing access to the interrupt
- Causing the interrupt to an adapter
- Identifying the cause of the interrupt
- Resetting the interrupt
- Establishing addressability for the push or pull operation.

The base address of the I/O address space is defined during system configuration (Setup). The control areas within that I/O address space for a unit with a physical interface are defined in "Physical Interface Support" on page 1-15.

Note: Before using a unit with a physical interface, the enable-interrupt and the enable-bus-master bits in the Subsystem Control port (defined in "Subsystem Control Port" on page 1-17) must be set to 1.

Feature Adapter to System Unit Protocols

The following section defines the protocols used with the physical interface for the push, pull, and signal operations.

- Pushing and Pulling

  The protocol used to push and pull to or from shared memory is done differently depending upon whether it is being done from a system unit or adapter.

  - A system unit uses either Load and Store instructions or DMA operations to push and pull data, depending on the data being transferred.
  - Adapters perform the push and pull operation to or from shared memory using the bus-master function when the shared memory is not physically located on the adapter.

- Signalling (which causes a hardware interrupt)

  The protocol used to generate an interrupt to a system unit differs from that used to generate an interrupt to an adapter.

  - An interrupt of a system unit by an adapter or another system unit is caused by raising the interrupt level (IRQ line) defined at system configuration time (Setup).

    Because system units allow an interrupt level to be shared by several adapters, Signalling Control areas are used to identify the unit that generates the interrupt.

    The interrupt-valid bit in the Command Busy/Status port (defined in "Command Busy/Status Port" on page 1-18) also indicates which unit generated the interrupt.

  - An interrupt of an adapter is caused by writing to the Attention port. The reason code of hex D and the device number of hex 0 must be written to the Attention port as defined in "Attention Port" on page 1-16. Only reason code D and device number 0 are used by the physical protocol.

    When the Attention port is written to, the operation-pending bit in the Command Busy/Status port is set to 1. The receive-signal logic in the adapter is responsible for resetting this bit. Interrupt support in the adapter-control program invokes the receive-signal logic .

    The send-signal logic and receive-signal logic operate in support of the signalling protocols defined at the delivery level.

    - The send-signal logic (at the physical level) is used by the send-signal logic (at the delivery level) after it has pushed the signalling information into the appropriate Signalling Control area.

    - The receive-signal logic examines all Signalling Control areas when a hardware interrupt is received. This ensures that all signalling indications are captured even though a hardware interrupt could have been missed by some hardware-busy or masked condition.

      Using the information in the Signalling Control areas, the receive-signal logic posts the appropriate delivery-signal logic (at the delivery level) and clears the contents of the Signalling Control area to 0. For management, it posts

the unit-management logic and resets the management indicator in the appropriate Signalling Control area.

Note: In order to prevent the loss of interrupts when the operation-pending bit in the Command/Busy Status port is set to 1, the processing of any interrupt should include the processing of all Signalling Control areas. Then, following the reset of the operation-pending bit, a final recheck of all the Signalling Control areas should be performed.

Feature Adapter to Feature Adapter Protocols

Push, pull, and signal operations are also defined between two adapters, each having a physical interface.

The push and pull operations use the bus-master-DMA operations in both directions.

The signal operation (which causes a hardware interrupt) is done in the same manner as defined from adapter to system unit. The Signalling Control area identifies the cause and source of the interrupts.

Chapter 3. Delivery Level

The delivery level defines the services and protocols for managing the delivery of one or more control elements between a system unit and adapter or between adapters. The delivery level uses the services provided by the underlying physical level.

The primary function of the delivery level is to support the sending and receiving of control elements. Control elements are exchanged using a pair of delivery pipes. One delivery pipe is used for sending control elements, the other for receiving them.

The delivery level provides a set of services that can place control elements into a delivery pipe (enqueue), take control elements out of a delivery pipe (dequeue), and signal between units.

The delivery level defines the protocols used in the delivery of control elements. These protocols operate with the control areas located in shared memory, and are built on top of the physical-level support.

Structure

Figure 3-1 on page 3-2 shows the delivery-level structure for a configuration supporting control element delivery between a pair of units. The Local Enqueue and Local Dequeue Control areas are in memory that is local to each unit, and the Surrogate Enqueue and Surrogate Dequeue Control areas are in shared memory. The pipes, one in-bound and one out-bound, are also located in shared memory. There can be multiple pairs of delivery pipes in a user's system containing multiple units (system units and adapters).

Figure 3-1 on page 3-2 also shows that delivery support consists of enqueue logic, dequeue logic, and management logic and provides a set of services to these entities through the send and receive interfaces.

© Copyright IBM Corp. 1991.

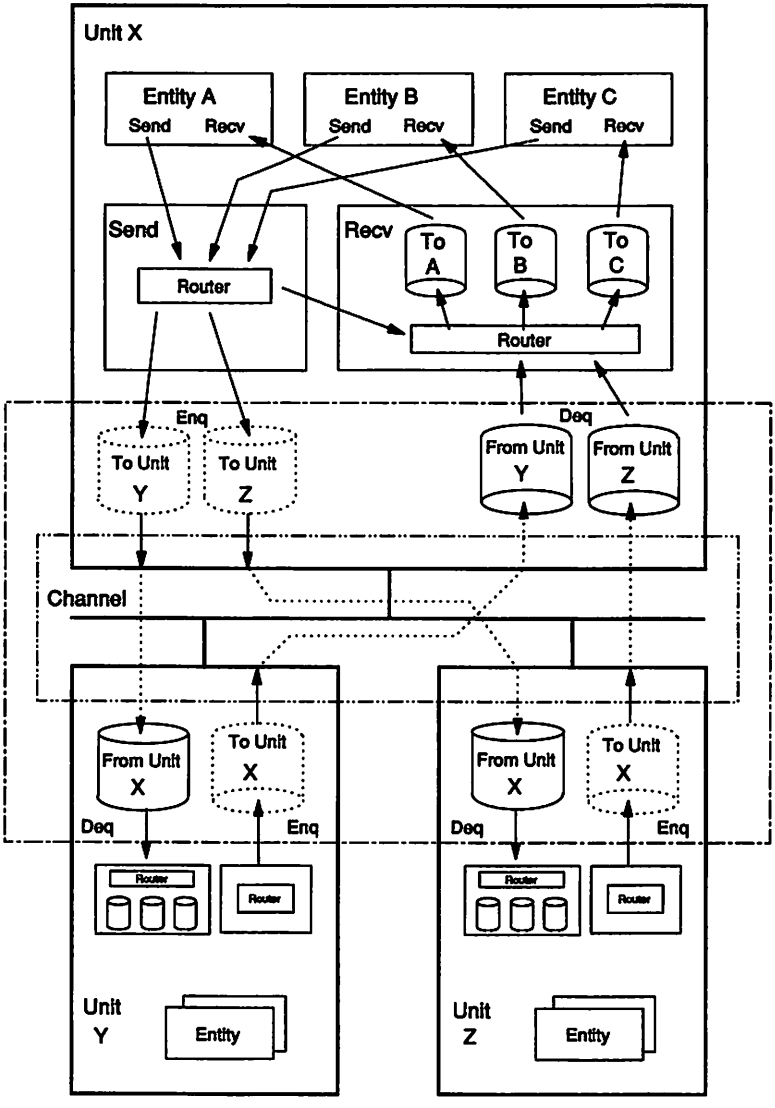


Figure 3-1. Overall Delivery Level Structure

Delivery Pipes

The following figure shows the relationship of the delivery pipe to the enqueue logic and dequeue logic (at either end of the delivery pipe) and the individual control areas associated with each pipe. For more information on the delivery pipe and the associated control areas, see "Delivery Pipe" on page 1-19.

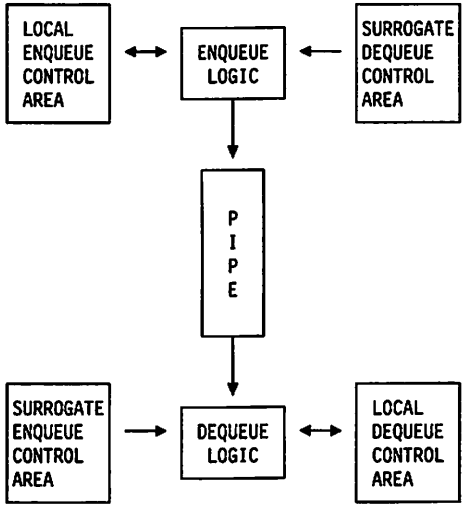


Figure 3-2. Delivery Pipe and Associated Control Areas

The enqueue logic places the control elements into the delivery pipe at the source; the dequeue logic removes the control elements from the delivery pipe at the destination. When a system contains several units, each pair of units has its own pair of delivery pipes (one in-bound and one out-bound). An example of a multiple unit structure is shown in the following figure.

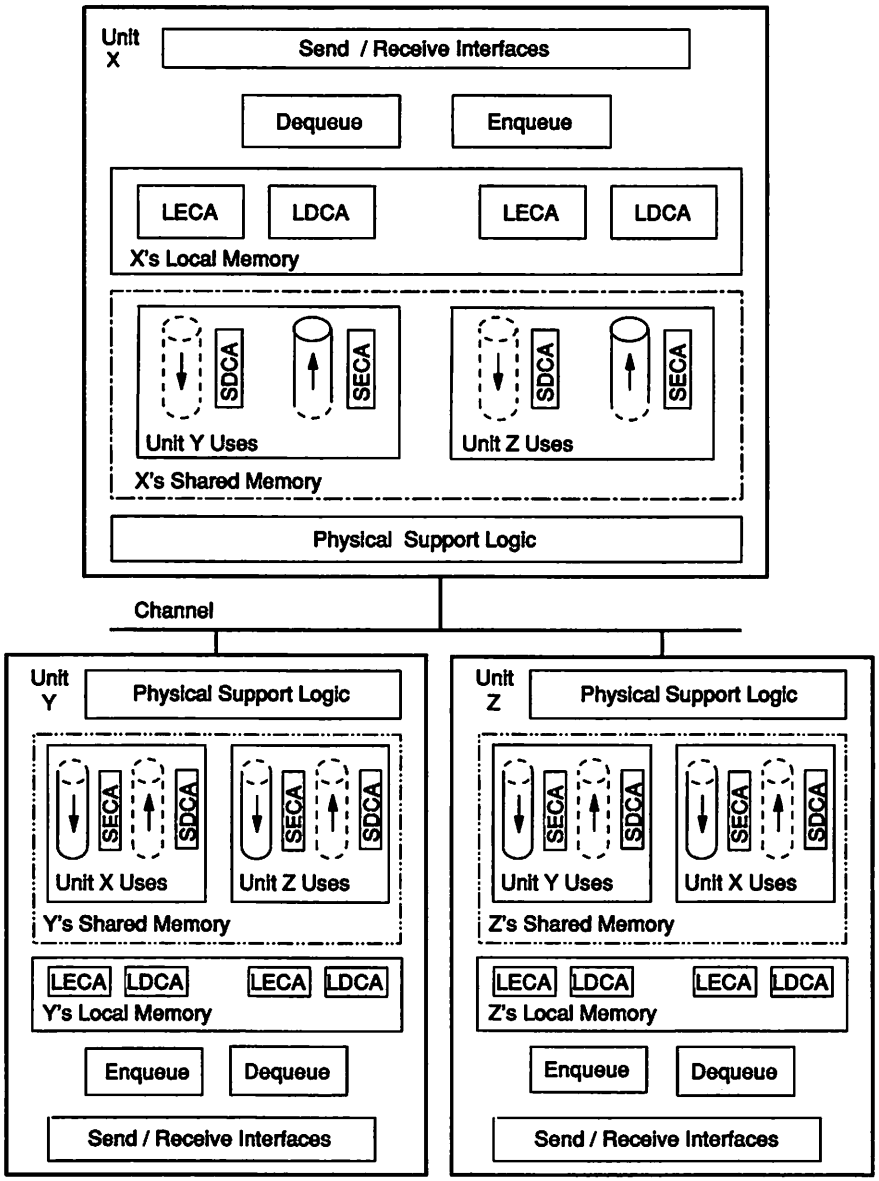


Figure 3-3. Multiple Unit Structure

In the example, the control areas and the delivery pipes are shown within dotted lines to indicate that they are located in shared memory.

Signalling

The enqueue and dequeue protocols operate using status information maintained in shared memory and do not require signalling operations. However, for certain implementations and protocols, there might be a need to signal the other unit when certain changes of the delivery pipe occur. The following are optional signalling conditions:

- When a delivery pipe goes from empty to not empty
- When a delivery pipe goes from not empty to empty
- When a delivery pipe goes from full to not full
- When a delivery pipe goes from not full to full
- When an control element is placed into a delivery pipe
- When an control element is removed from a delivery pipe.

The Move mode architecture also defines signalling protocols which support the management of the delivery-support logic.

Send and Receive Interfaces

Figure 3-1 on page 3-2 shows the send-interface logic between an entity sending a control element and the enqueue logic. The send-interface logic separates the various forms of operating-support interfaces and scheduling policies from the enqueue protocols used between units.

Figure 3-1 on page 3-2 shows the receive-interface logic between an entity receiving a control element and the dequeue logic. The receive-interface logic separates the various forms of operating-support interfaces and scheduling policies from the dequeue-logic protocols used between units.

Delivery Level Services

The services provided by the delivery level are used by entities at the processing level to exchange requests and replies in the form of control elements between correspondent entities. They include services to:

- Enqueue control elements
- Dequeue control elements.

Enqueue Service

The enqueue service is used to place one or more variable-length control elements into the delivery pipe. The enqueue logic uses the services provided by the underlying physical level to access the control areas associated with the destination unit and to place control elements into the delivery pipe in shared memory.

The specifics for calling this service are an internal function and are not defined by the architecture. However, the following illustrates the parameters required when calling this service:

```
ENQUEUE (address, count, urgency, return code)
```

The ***address*** parameter indicates the location in the address space of the caller of the control element to be placed into the delivery pipe.

The ***count*** parameter indicates the number of control elements to be enqueued. If more than one control element is specified, all the control elements must be contiguous to one another in the memory space identified by the address parameter.

The ***urgency*** parameter indicates the relative priority of the request to enqueue a control element: normal or expedited.

The ***return code*** parameter is returned to the caller and indicates the success or failure of the enqueue operation.

The Destination field in the control element determines the destination unit and the delivery pipe that deliver the control element.

The Source field in the control element identifies the source unit and delivery pipe used when returning a reply.

Dequeue Service

The dequeue service is used to initiate the removal of control elements from the delivery pipe. Control elements are always removed from the head of the delivery pipe. The dequeue logic uses the services provided by the underlying physical level to access the control areas associated with the delivery pipe, as well as the delivery pipe.

Control elements are passed to the receive-interface logic using a call-back mechanism. Using this approach, the receive-interface logic calls the dequeue logic, which in turn calls back the receive-interface logic with the address and length of the control element at the head of the delivery pipe. The receive-interface logic uses the contents of the control element to determine which entity receives the control element and how to handle the delivery of the control element to the destination entity.

This process repeats itself until the delivery pipe is empty or until the receive-interface logic indicates it cannot accept any more control elements.

Local policy determines the means of delivery. The control element can be passed directly to the entity, or it can be queued for processing later. If the entity is not active, does not exist, or does not have an active read pending, the control element can be returned to the sender.

The specifics for calling this service are an internal function and are not defined by the architecture. However, the following conceptual service illustrates the parameters required when calling the dequeue service:

```
DEQUEUE (unit ID, return code)
```

The ***unit ID*** parameter identifies which delivery pipe is used.

The ***return code*** parameter is returned to the caller and indicates the success or failure of the service call.

Delivery Level Protocols

The delivery-level protocols define the operations of the delivery-level support as they appear across the channel. This includes defining the push and pull operations used with the physical-level services, and the signalling operations used with the signalling services.

The operation of the delivery-level protocols is defined for a given pipe. At the delivery level, all pipes operate the same. An overview of the push and pull operations between the enqueue logic and its control areas and the dequeue logic and its control areas is shown in Figure 3-4 on page 3-9. The specific definitions for these control areas can be found in "Delivery Pipe" on page 1-19. The locations in shared memory for the control areas shown are defined at initialization by the configuration records. The layout of the configuration record is shown in Figure 5-1 on page 5-5.

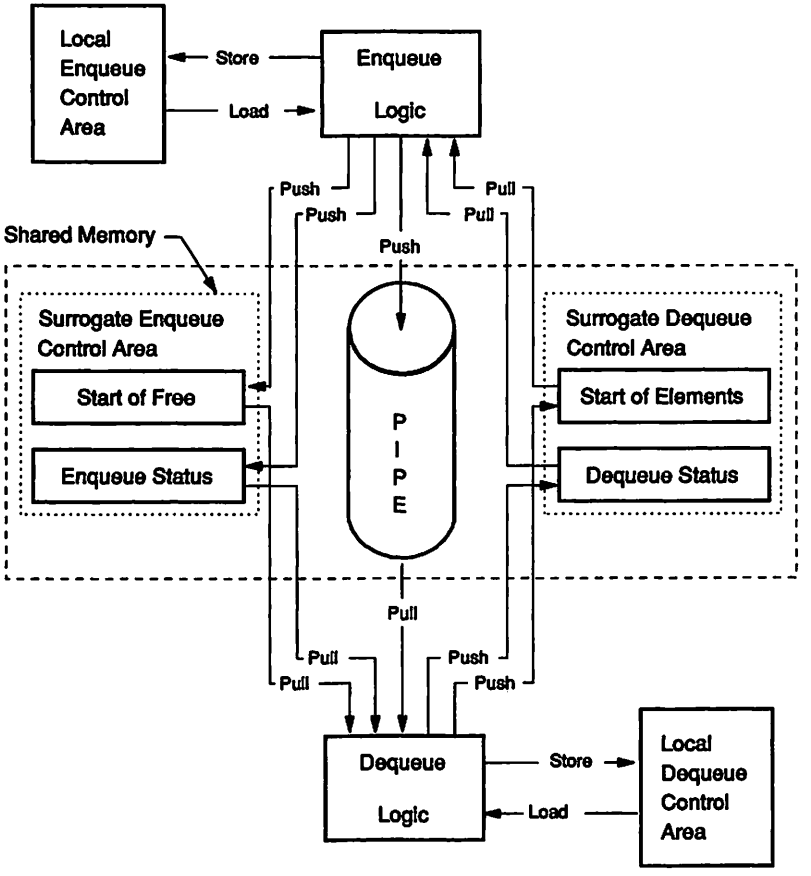


Figure 3-4. Delivery Pipe and Associated Control Areas

The overall operation of the signalling portion of the enqueue and dequeue logic is shown in Figure 3-5 on page 3-10. The specific definition of the Signalling Control area can be found in "Signalling Control Area" on page 1-29.

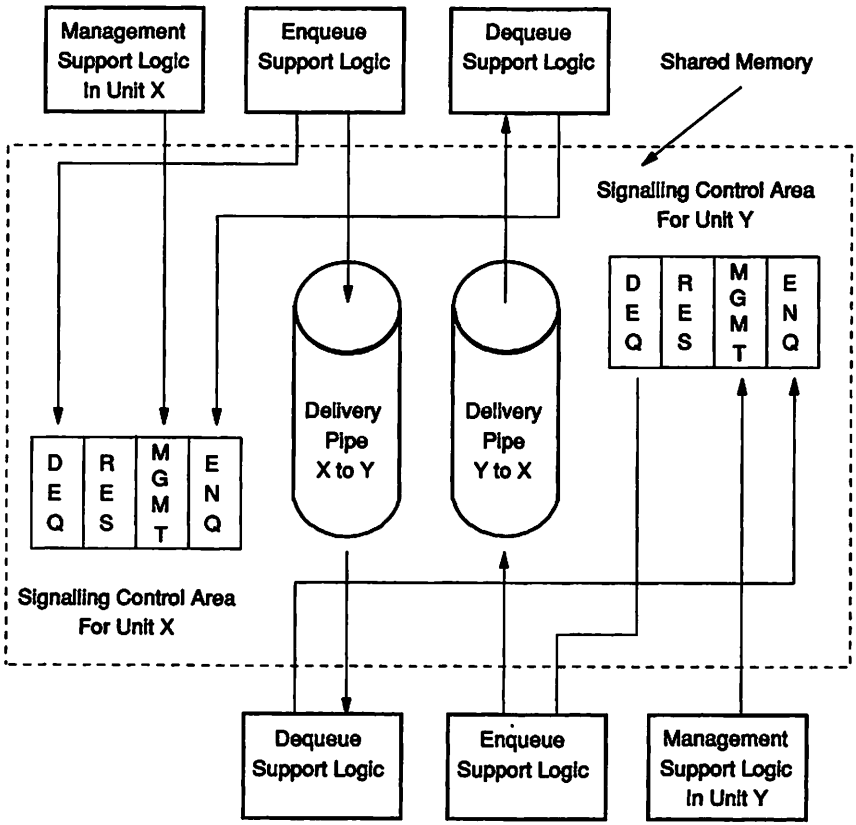


Figure 3-5. Signalling Control Area Use

Each delivery pipe has a set of pointers in the Local Enqueue Control area and in the Local Dequeue Control area. The pointers in the Local Enqueue Control area indicate the start-of-free space and the end-of-free space. The pointers in the Local Dequeue Control area indicate the start-of-elements and the end-of-elements. Both control areas have pointers that indicate the limits of the delivery pipe:

- TOP - the threshold point for normal enqueue operations.
- BASE - the beginning of the usable space in a delivery pipe.
- END - the end of the usable space.

Examples of the delivery pipe in different states are shown in the following figure. An example of a control element that is being wrapped is included. This occurs when a control element does not fit into the space remaining between the current end-of-free space (EF) and the top-of-pipe (TOP).

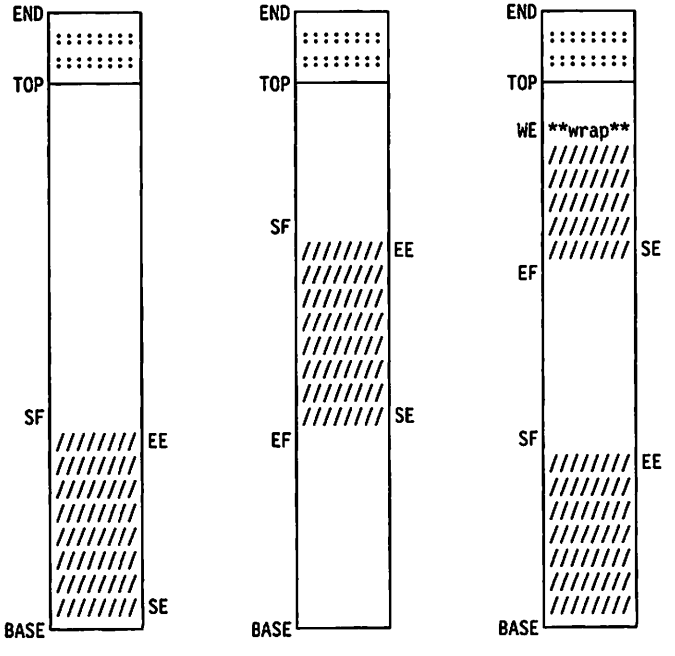


Figure 3-6. Various Examples of Pipe Use

The following sections define the specific protocols for the enqueue logic and dequeue logic. The logic is defined in pseudo code, which operates on the information in the control areas and calls the services of the physical level.

Enqueue Protocol

The protocol defines the operation of the enqueue logic that provides control elements with the enqueue service.

The following discusses how the enqueue logic uses the control areas in shared memory to manage the operation of the delivery

pipe. The individual bits and fields are described in "Delivery Pipe" on page 1-19 and "Signalling Control Area" on page 1-29.

- The size of a control element is specified in bytes and can be found in the Length field of the control element. Several control elements can be enqueued with a single invocation of the enqueue service. However, all of the control elements involved must be contiguous in memory (see "Enqueue Service" on page 3-6).
- The location of the top-of-pipe (TOP) is determined by the implementation but must be sufficient to hold at least one wrap control element.
- The enqueueing of a control element is a local operation when the delivery pipe is physically located in the source unit. It is a distributed operation and involves a push of the control element across the channel when the delivery pipe is physically located in the destination unit.
- The type of operation (local or distributed) required to access the Surrogate Start of Free and Surrogate Enqueue Status fields depends on where they are located in shared memory.
- For the enqueue logic, the receive-signal logic is used to synchronize the state of the empty bit in the Surrogate Dequeue Control area with the state of the empty bit in the Local Enqueue Control area.
- To allow the tailoring of signalling conditions, several signalling options can be specified during configuration and initialization. These options are used by the state-change function of the enqueue logic to indicate the conditions under which a signal is sent by the enqueue logic to the dequeue logic at the other end of the delivery pipe. The following describes the signalling-option bits and the actions taken when they are set to 1.
  - If the not-full to full option is selected, a signal is sent when the pipe goes from a not-full state to a full state.
  - If the empty to not-empty option is selected, a signal is sent when the pipe goes from an empty state to a not-empty state.
  - If the signal-on-enqueue option is selected, a signal is sent each time a control element is placed into the pipe.

    Note: Care should be exercised when configuring the signal on-enqueue option. Use of this option with high-speed

adapters could result in an overload of the destination unit's interrupt handler.

- Several bits in the Enqueue Status field of the Local Enqueue Control area are used internally to control the operation of the enqueue logic.
  - The queued bit, when set to 1, indicates a control element has been successfully placed into the delivery pipe.
  - The empty bit, when set to 1, indicates the empty bit in the Surrogate Dequeue Status field is set to 1.
- Several other bits in the Enqueue Status field of the Local Enqueue Control area hold information used to update the Status field in the Surrogate Enqueue Control area.
  - The full-status bit, when set to 1, indicates the delivery pipe is full from the perspective of the enqueue logic.
  - The wrap bit is used to indicate that a wrap control element has been placed into the delivery pipe by the enqueue logic. The enqueueing of a wrap control element causes a toggling of this bit (1 to 0 or 0 to 1).

The pseudo code in the following pages is an example of the enqueue protocols. These protocols are:

- Enqueue initialization
- Enqueue receive signal
- Enqueue function
- Enqueue state-change function.

Enqueue Initialization

The following occurs when the delivery service in a unit is initialized.

```
{Setup the fields in the local enqueue control area containing
 the starting and ending offset values for start_of_free and
 end_of_free.}

SET start_of_free = BASE
SET end_of_free = top_of_pipe

{Setup various flag bits in the enqueue status field of the
 local enqueue control area to their correct initial value.}

SET enqueue_wrap = 0
SET full = 0
SET queued = 0
SET empty = 1

{Setup the field in the surrogate dequeue control area containing
 the starting offset value for start_of_elements.}

SET surrogate start_of_elements = BASE

{Setup the various flag bits in the dequeue status field of the
 surrogate dequeue control area to their correct initial value.}

SET surrogate dequeue_wrap = 0
SET surrogate empty = 1
```

Enqueue Receive Signal

The following occurs when a signal is received from the dequeue logic at the other of the delivery pipe. This is done prior to invoking the enqueue logic.

```
IF surrogate empty = 1 THEN
   SET empty = 1
ENDIF
```

Enqueue Function

The following pseudo code represents the main-line code of the enqueue-support logic. It is called by the send-interface or receive-interface logic to place control elements into a delivery pipe. See "Enqueue Service" on page 3-6 for a description of the calling parameters and the return codes.

```
PROCEDURE Enqueue (control element address, number, urgency, return code)

   {Call the queue routine to place a control element in the pipe
    for as long as there are control elements to enqueue or until
    an exception condition is encountered. }

   DO
      CALL queue_element (control element address, return code)
      SET number = number - 1
   WHILE return = successful and number ≠ 0

   RETURN return code to caller
ENDPROC
```

The following pseudo code is called by the main-line code of the enqueue-support logic. It places control elements into the pipe one at a time, updates the appropriate Local and Surrogate Control areas, and signals the dequeue logic at the other end of the pipe, when necessary.

```
PROCEDURE queue_element(control element address, return code)

   {Update the end of free space (enqueue tail) by tracking
    where the start of elements (dequeue head) is currently
    located.}

   IF (end_of_free > surrogate start_of_element) OR
      (end_of_free = surrogate start_of_element AND
       enqueue_wrap = surrogate dequeue_wrap)
      THEN
         SET end_of_free = top_of_pipe
      ELSE
         SET end_of_free = surrogate start_of_elements
   ENDIF

   {Check to see if an element can be placed in the
    pipe.  If so, place it in the pipe, update the
    enqueue state information, and return indication
    that the enqueue was successful.}

   IF length of element ≤ end_of_free - start_of_free
      THEN

         {Element can be placed into the pipe.}

         move element onto pipe at start_of_free
         increment start_of_free by length of element
         SET queued = 1
         CALL state_change
         RETURN successful indication to caller
      ELSE

         {Element won't fit in the pipe.  Check if the pipe
          needs to wrap to the top_of_pipe.  If it does,
          build wrap control element, place it in the pipe,
          update the enqueue state information, and try
          again to place the element in the pipe.  If it
          does not, update enqueue state information and
          return pipe full indication to caller.}

         IF end_of_free = top_of_pipe
            THEN

               {A wrap condition exists.}

               push wrap element onto pipe at start_of_free
               SET end_of_free = surrogate start_of_elements
               SET start_of_free = base

               {Toggle the state of the wrap bit.}

               IF enqueue_wrap = 0
                  THEN
                     SET enqueue_wrap = 1
                  ELSE
                     SET enqueue_wrap = 0
               ENDIF
```

```
                {Now check to see if control element will fit
                 in the pipe after we have wrapped to BASE of
                 the pipe.  If it does, move control element
                 into the pipe and update the enqueue state
                 information.  If not, update state information
                 and return full condition to caller. }

                IF element length ≤ end_of_free - start_of_free
                   THEN

                      {Element will fit in the pipe.}

                      push element onto pipe at start_of_free
                      increment start_of_free by element length
                      SET queued = 1
                      CALL State_Change
                      RETURN successful indication to caller
                   ELSE

                      {Element still won't fit in the pipe.

                      SET full = 1
                      CALL State_Change
                      RETURN pipe full indication to caller
                ENDIF

             ELSE

                {Pipe has not wrapped, but it is full.  Update
                 the enqueue state information and indicate
                 control element not queued.}

                SET full = 1
                CALL State_Change
                RETURN pipe full indication to caller
          ENDIF
   ENDIF
ENDPROC
```

Enqueue State Change Function

The following pseudo code represents the state-change code of the enqueue-support logic. It is called by the enqueue function to update the status information associated with the Surrogate Enqueue Control area, and to determine if a signal must be sent to the unit at the other end of the delivery pipe.

```
PROCEDURE state_change()

   {Update the surrogate enqueue control area.}

   Move enqueue_status to surrogate enqueue_status
   Move start_of_free to surrogate start_of_free

   {Determine if a change in the state of the pipe has
    occurred which requires the sending of a physical
    signal to the dequeue logic in the unit at the other
    end of the pipe.}

   IF (queued = 1 AND signal on-enqueue = 1) OR
      (queued = 1 AND surrogate empty = 1 AND
      empty to non-empty = 1) OR
      (full = 1 AND not-full to full = 1)
      THEN

         {Physically send a signal to the dequeue logic in
          the specified unit indicating a change in state.}

         CALL signal(unit, dequeue logic, state_change)
   ENDIF
   SET empty = 0
   SET full = 0
   SET queued = 0
ENDPROC
```

Dequeue Protocol

The dequeue protocol defines the operation of the dequeue logic that provides the control element dequeue service.

The following discusses how the dequeue logic uses the control areas in shared memory to manage the operation of the delivery pipe. The individual bits and fields are described in "Delivery Pipe" on page 1-19 and "Signalling Control Area" on page 1-29.

- The dequeueing of a control element is a local operation if the delivery pipe is physically located in the destination unit. It is a distributed operation and involves a pull across the channel when the delivery pipe is physically located in the source unit.
- The type of operation (local or distributed) required to access the Surrogate Control areas depends upon where they are physically located in shared memory.
- The dequeue logic removes and discards the wrap control element when dequeueing so that the space between the wrap control element and the top-of-pipe (if any) can be made available to the enqueue logic.
- For the dequeue logic, the receive-signal logic is used to synchronize the state of the full bit in the Surrogate Enqueue Control area with the state of the full bit in the Local Dequeue Control area prior to invoking the dequeue logic.
- To allow the tailoring of signalling conditions, several signalling options can be specified during configuration and initialization. These options are used by the state-change function of the dequeue logic to indicate the conditions under which a signal is sent by the dequeue logic to the enqueue logic at the other end of the delivery pipe. The following describes the signalling-option bits and the actions taken when they are set to 1.
  - If the not-empty to empty option is selected, a signal is sent when the pipe goes from a not-empty state to an empty state.
  - If the full to not-full option is selected, a signal is sent when the pipe goes from a full state to a not-full state.
  - If the signal-on-dequeue option is selected, a signal is sent each time a control element is removed from the pipe.

Note: Care should be exercised when configuring the signal on-dequeue option. Use of this option with high speed adapters could overload the source unit's interrupt handler.

- Several bits in the Dequeue Status field of the Local Dequeue Control area are used internally to control the operation of the dequeue logic.
  - The dequeued bit, when set to 1, indicates a control element has been successfully removed from the delivery pipe.
  - The pre-empt bit, when set to 1, indicates the current dequeue operation is terminated even if the delivery pipe is not empty. The pre-empt bit is set based on the return code from the receive-interface logic. A value of 0 indicates that another control element can be passed to this logic. A value of 1 indicates that no more control elements are passed at this time.
- Several other bits in the Dequeue Status field of the Local Enqueue Control area hold information used to update the Status field in the Surrogate Dequeue Control area.
  - The empty bit, when set to 1, indicates the delivery pipe is empty from the perspective of the dequeue logic.
  - The wrap bit is used to indicate that a wrap control element has been removed from the delivery pipe by the dequeue logic. The dequeueing of a wrap control element causes a toggling of this bit (1 to 0 or 0 to 1).

The pseudo code in the following pages is an example of the dequeue protocols. These protocols are:

- Dequeue initialization
- Dequeue receive signal
- Dequeue function
- Dequeue state-change function.

Dequeue Initialization

The following occurs when the dequeue portion of the delivery service in a unit is initialized.

```
{Setup the fields in the local dequeue control area containing
 the starting and ending offset values for start_of_elements and
 end_of_elements.}

SET start_of_elements = BASE
SET end_of_elements = BASE

{Setup the various flag bits in the dequeue status field of the
 local dequeue control area to their correct initial value.}

SET dequeue_wrap = 0
SET pre-empt = 0
SET dequeued = 0
SET full = 0
SET empty = 1

{Setup the field in the surrogate enqueue control area containing
 the starting offset value for start_of_free.}

SET surrogate start_of_free = BASE

{Setup the various flag bits in the enqueue status field of the
 surrogate enqueue control area to their correct initial value.}

SET surrogate enqueue_wrap = 0
SET surrogate full = 0

ENDPROC
```

Dequeue Receive Signal

The following occurs when a signal is received from the enqueue logic at the other of the delivery pipe. This is done prior to invoking the dequeue logic.

```
IF surrogate full = 1 THEN
   SET full = 1
ENDIF
```

Dequeue Function

The following pseudo code represents the main-line code of the dequeue-support logic. It is called by the receive-interface logic to initiate removing control elements from a delivery pipe. See "Dequeue Service" on page 3-7 for a description of the calling parameters and the return codes.

```
PROCEDURE dequeue (unit, return code)

   IF start_of_elements = surrogate start_of_free AND
      dequeue_wrap = surrogate enqueue_wrap
      THEN
         CALL state_change
         RETURN pipe empty to caller
      ELSE

         {Remove all the elements from the pipe and pass them one at
          a time to the input router of the receive interface logic.}

         DO
            SET end_of_elements = surrogate start_of_free

            {Check the element at the head of the pipe.  If it is a wrap
             element, discard it, reset the start of elements pointer to
             the base of the pipe, and toggle the setting of the dequeue wrap
             state.  If not, call the input router function of the receive
             interface logic with the address of the element.}

            IF element at start_of_elements is a wrap element
               THEN
                  dequeue and discard the wrap element
                  SET start_of_elements = base of pipe
                  SET dequeued = 1

                  {Toggle the dequeue wrap state.}

                  IF dequeue_wrap = 0
                     THEN
                        SET dequeue wrap = 1
                     ELSE
                        SET dequeue wrap = 0
                  ENDIF
               ELSE
                  CALL receive_input_router (start_of_elements)
                  IF return_code = 1 THEN
                     SET pre-empt = 1
                  SET start_of_elements = start_of_elements + element length
                  SET dequeued flag = 1
                  CALL state_change
            ENDIF

         WHILE (pre-empt = 0) AND
               (start_of_elements ≠ end_of_elements OR
                dequeue_wrap ≠ surrogate_wrap)

         {Determine if we stopped because pipe was empty.  If so,
          set empty and call state_change.}

         IF pre_empt ≠ 1 THEN
            SET empty = 1
            SET pre-empt = 0
            CALL state_change
         ENDIF
   ENDIF
ENDPROC
```

Dequeue State Change Function

The following pseudo code represents the state-change code of the dequeue-support logic. It is called by the dequeue function to update the status information associated with the Surrogate Dequeue Control area, and to determine if a signal must be sent to the unit at the other end of the delivery pipe.

```
PROCEDURE state_change ();

   {Update the surrogate dequeue control area.}

   Push dequeue_status to surrogate dequeue_status
   Push start_of_elements to surrogate start_of_elements

   {Determine if a change in the state of the pipe has
    occurred which requires the sending of a physical
    signal to the enqueue logic in the unit at the other
    end of the pipe.}


   IF (dequeued = 1 AND signal on dequeue = 1) OR
      (surrogate full = 1 AND full to not-full = 1) OR
      (empty = 1 AND not-empty to empty = 1)
      THEN

         {Physically send a signal to the enqueue logic in
          the specified unit indicating a change in state.}

         CALL signal(unit, enqueue logic, state_change)
   ENDIF

   SET empty = 0
   SET full = 0
   SET dequeued = 0

ENDPROC
```

Notes:

Chapter 4. Processing Level

At the processing level, requests and the responses to those requests are exchanged between entities using the services provided by the underlying delivery level. These entities represent the clients and the servers in the various system units and adapters attached to the channel.

An entity sends a request to another entity or receives a reply to a previous request using control elements. The information contained in these control elements, their number, order of exchange, and meaning, constitute a protocol between the two entities. This protocol is used by the two entities to coordinate and control their activities. The architecture does not define these protocols. However, the architecture does define the structure and format of the control elements and the content and use of a common header that is required in all control elements.

The delivery service uses information in this common header to deliver control elements without knowing or understanding what is contained in the body of the control element.

The cooperating entities use information in the common header to identify the source, indicate the type and urgency of the information contained in the control element, and to correlate replies with previous requests.

Structure

The processing level is structured into two parts: an interface part and a processing part. The interface part provides a means for tailoring the interface between the entities at this level, the local operating environment, and the underlying enqueue and dequeue services provided by the delivery level. The processing part consists of the entities that use the underlying delivery service to communicate with one another in the performance of a common task.

The send-interface and receive-interface logics provide a means of tailoring the local interface in each unit (system unit or adapter) to

© Copyright IBM Corp. 1991.

the processing requirements of the entities, their operating environment, and the underlying delivery service.

This allows the receive-interface logic to place control elements in a local queue until the destination entity is ready to process them, instead of requiring the destination entity to wait for the arrival of a control element.

The operating environment at one end of the delivery pipe might not support the ability to block or suspend the execution of an entity pending the receipt of a control element, while the operating environment at the other end of the delivery pipe might be able to do so.

Each of these situations has an impact on the local behavior of the communicating entities, but not on their ability to send or receive control elements to each other.

The content of control elements is totally dependent on the entity pair. However, the architecture defines a set of common-use control elements and protocols at the processing level when either command or data chaining is used. The architecture also makes provisions for defining additional control elements used between entities whose requirements cannot be met by the control elements in the common-use set.

Processing Level Operation

The delivery-support logic operates in conjunction with the various entities and the underlying operating environment. The following shows an example of this interaction.

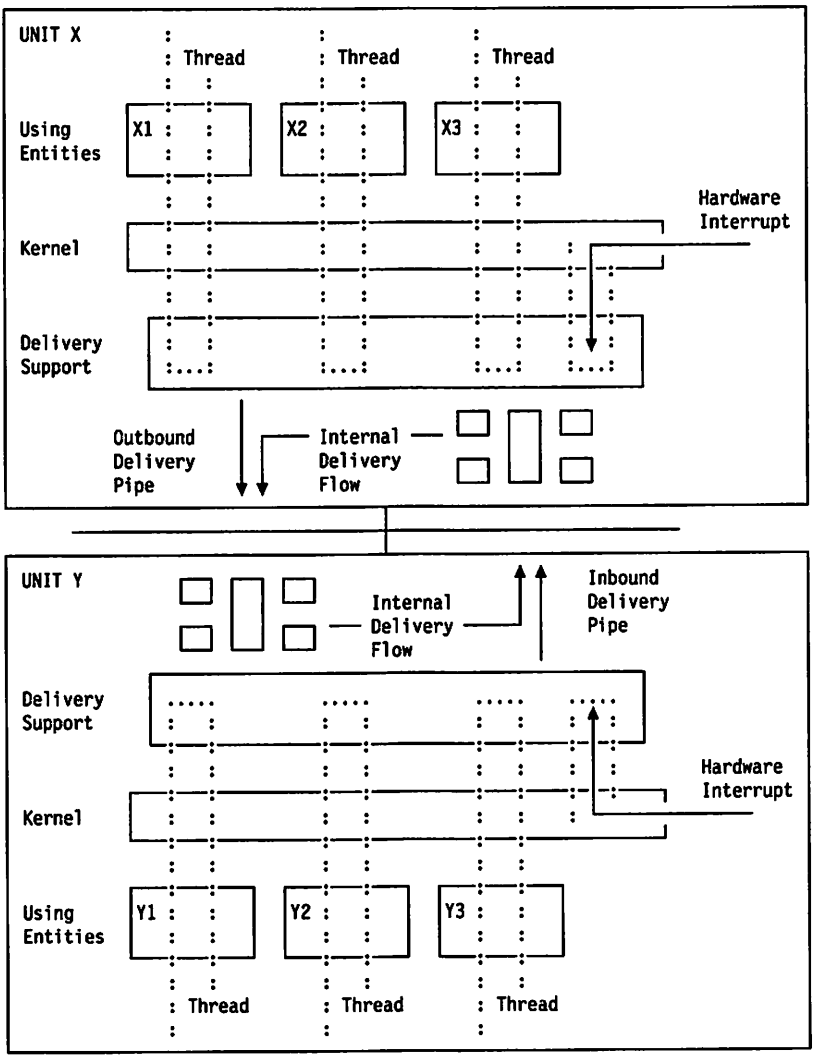


Figure 4-1. Device Driver or Handler Model

Each entity using the delivery support is operated on a unit of dispatching (called a thread). The delivery support is functionally similar to that of a device driver in operating-systems environments (Figure 4-2 on page 4-5).

- It has a strategy portion which runs under the same thread as the entity.
- It can allow only one thread at a time to use the delivery-support logic. This feature can be used to serialize the enqueueing of control elements into the shared-delivery pipe.
- It allows the delivery-support logic to cause a wait condition (block) on a specific thread. This can be used to handle pipe-full or receive-pending conditions.
- It has a local-data area for the delivery-support logic. This can be used for pipes as well as for delivery state and configuration.
- It has an interrupt handler which is called when a hardware interrupt occurs. This can be used to support the signalling protocol and initiate the dequeue logic.

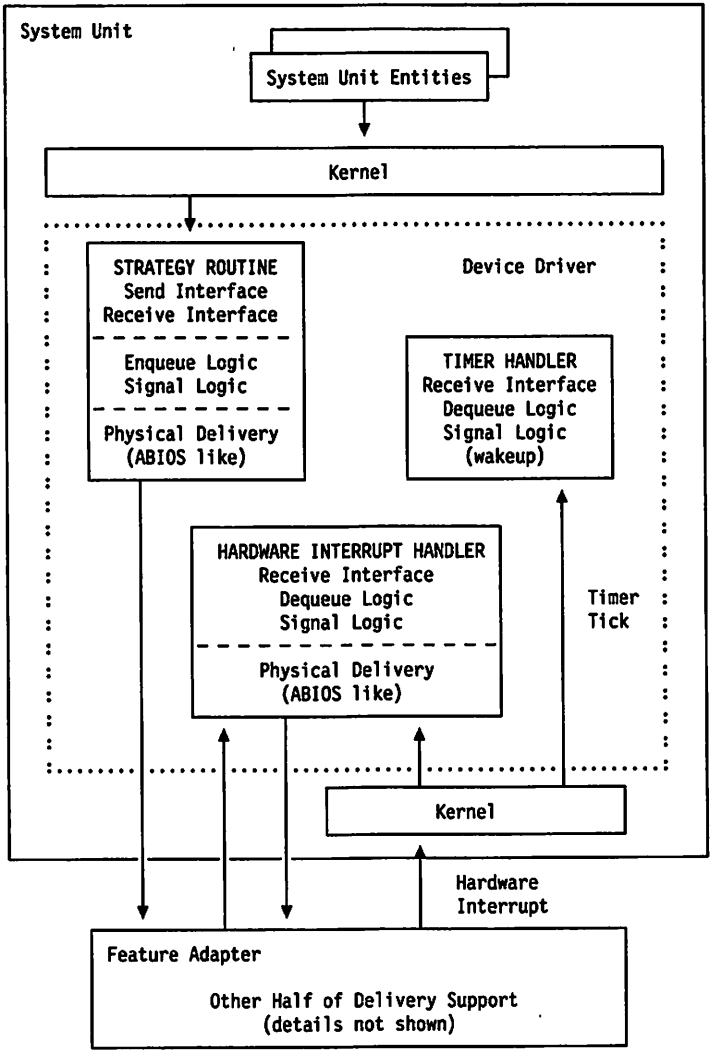


Figure 4-2. Device Driver or Handler Model

A hardware interrupt causes the scheduling of the signalling portion of the delivery support for that unit. The physical protocol that causes the interrupt differs for system units and adapters.

Figure 4-2 also shows that control elements can be removed from the pipe going in one direction between units and placed in the pipe going in the other direction between units (internal delivery flow).

This routing of control elements can be considered an internal operation. This capability is required to support the handling of certain error conditions. The figure does not show the physical-delivery support used to pass control elements, data, and signals between the two units.

Send

The enqueue operation is performed in the device driver or handler routine. The control elements move on the entity's thread when the Send command is executed. The details of the enqueue logic are defined in "Control Element Delivery" on page 1-11.

The architecture does not define policy. However, the send-interface logic provides a means of implementing local policy for each unit. The send-interface logic determines:

- Whether a single control element or multiple control elements are used.
- Whether control is returned directly to the calling entity when the pipe is full, or the thread is blocked until the full condition clears.

Receive

On the receive side, a receive pending exists for any Send command between an entity pair. The Send command can involve requests, replies to requests, or event type control elements. The specific entity-to-entity protocol determines how the Send and Receive commands are correlated. This correlation can be different for each entity pair. It is not restricted by the architecture.

Any queueing for the server is performed at the processing level. The delivery pipe is used only to mask the characteristics of the physical interface when multiplexing the delivery of control elements to multiple entities in the same unit. The model used for control element delivery is that of a free-flowing pipe running between the source and destination entities.

The receiving of control elements is handled the same as a read-pending operation. The space for the control element must be pinned and known by the receive-interface support so the control element can be moved from the delivery pipe into shared memory without performing a context switch. Placing the control element into

shared memory satisfies the receive pending at the receiving entity. If the thread with the receive pending was blocked, it becomes unblocked.

If the receiving entity does not have a receive pending for the control element sent, the receive-interface logic removes the control element from the delivery pipe, and either discards it or returns the common header (returning the header is device dependent).

When the control element is returned:

- The Source and Destination fields are reversed.
- The control element is set to an error type control element.
- The Status Information field is updated.
- The remainder of the request control element is discarded.
- The Length field is updated.

To prevent flooding of the delivery pipe in the opposite direction, and to prevent continuous wrapping of reply and error type control elements, additional request control elements and any other type of control elements received while a Receive command is not pending might be discarded. This prevents out-of-step protocols associated with one entity-to-entity flow from blocking the flow to other entities.

The space management for the delivery pipes can provide space within the pipe to accommodate this internal flow although the delivery pipe could be considered full for normal flow.

The architecture does not define what makes up a receive pending. It can be handled in a number of different ways because it depends on each unit's receive-interface-support logic. For example, some implementations choose to have the receive-interface logic maintain a set of queues for control elements. There might be a single queue for all entities at the destination, or there might be a separate queue for each entity. These queues can be arranged by control element type (request, reply, error, event), entity, or any other manner.

Interrupt

The use of the device driver or handler model for the control element delivery allows the hardware interrupts at the physical level to be separated from the control element delivery provided to entity-to-entity pairs.

Entity-to-entity protocols can operate using request, reply, error, and event control elements which are passed through a pair of delivery pipes. This separates the operation of the entity-to-entity protocols from the hardware interrupts.

The delivery-level state information for the enqueue and dequeue protocols is defined so that the pipe can operate without interrupts if the support observes the changes in state and acts accordingly.

Hardware interrupts are used to assist certain implementations in keeping the control elements in pipes flowing between units. Separating requests from replies allows interrupt overhead to be reduced because several control elements are dequeued when a single hardware interrupt is handled.

The delivery-level protocols related to interrupts are called signals. The architecture defines the signalling protocols if they are required by the implementation. Signalling can be done at user specified intervals to maintain the flow in the pipe and when certain state changes occur within the delivery pipe. The specific state changes are defined as part of the architected enqueue and dequeue protocols.

Processing Level Services

The Move mode architecture does not define the services used at the processing level. It only provides a means for tailoring the interfaces between the entities at this level, the local operating environment, and the underlying delivery level.

The functions provided by the send and receive interfaces depend on the implementation. Individual implementations can consider sending multiple control elements with a single call, and can decide to return on a pipe-full condition or wait for the condition to clear.

The receive-interface logic can provide the queueing of control elements for entities that do not have a read pending. Each entity can have its own individual queue or multiple queues corresponding to the type of control element (request, reply, error, event).

Processing Level Protocols

In general, the processing-level protocols are not defined by the architecture. However, the architecture does define common uses for the control elements used in the entity-to-entity flow. These include:

- Command chaining
- Data chaining
- Notification and wait
- Initialization (peer level).

Command Chaining

The server can process the control elements synchronously or asynchronously, depending on whether commands are being chained or not. Normally, a server can process requests in any order (asynchronously).

Command chaining is used to force the server to process control elements synchronously. When the command-chaining bit is set, the processing of the next control element in the chain cannot begin until the server completes the processing of the current control element and returns the appropriate reply to the client. The chaining process continues until all control elements in the chain are processed or an error is detected. When an error is detected, an error type control element is returned and the remaining control elements in the chain are discarded.

Data Chaining

Data chaining is a method of managing complex data transfers. For example, a single logical block of data can be located in several noncontiguous areas in shared memory and might need to be transferred to a single destination. Or, a single block of data might need to be transferred from a single source and placed in several noncontiguous areas of shared memory.

The architecture supports two methods of data chaining: direct and indirect.

Direct Data Chaining

In direct data chaining, each noncontiguous data area is described by an entry in a chaining list that is included in the request control element. The entry contains a byte count and an address pointer to a data area. After each data area described by an entry is transferred, the next entry in the list is processed. This action continues until all the entries in the list have been processed or until no more data exits. Direct data chaining is invoked by using the read list and write list function codes (see "Read List" on page 1-43 and "Write List" on page 1-49).

Indirect Data Chaining

Indirect data chaining is similar to direct data chaining, except for the location of the chaining list. The list, however, is not part of the request control element. Instead, the list is kept separate and is referenced by means of an indirect list pointer in the Value field of the request control element. Indirect data chaining is invoked by setting the indirect bit in a read or write list request control element. The following figure illustrates how the request control element and indirect list appear for a read list request.

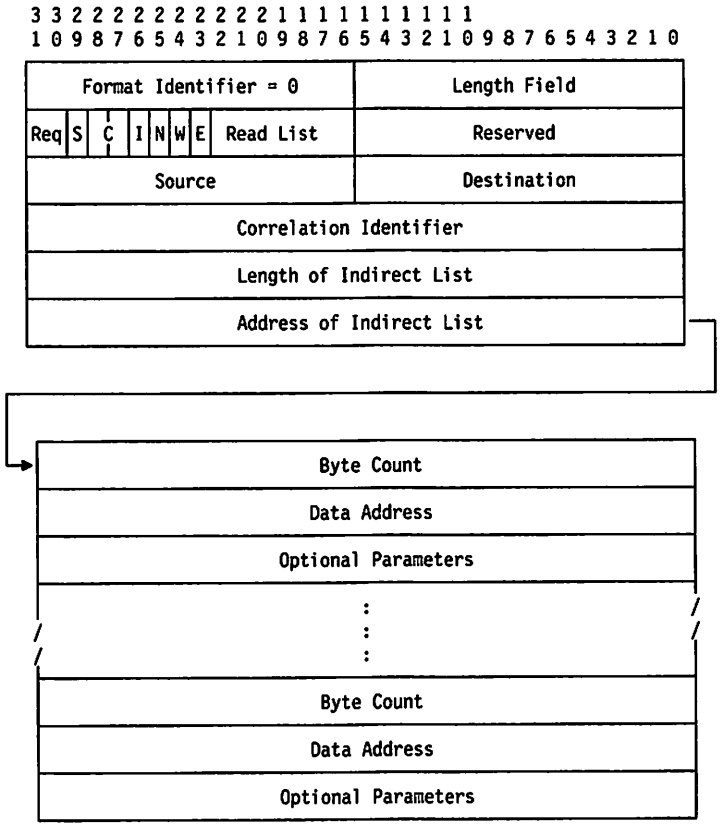


Figure 4-3. Indirect Data Chaining

Notification and Wait

A client can request notification when a server is about to process a specific control element by setting the notification bit (N) in that control element. The server then returns an event control element with the Source, Destination, and Correlation Identifier fields for the appropriate control element to the client.

A client can suspend the processing of a chain of control elements at a particular point by setting the wait bit (W) in the appropriate control element prior to submitting it to the delivery service. This causes the server to:

- Suspend further processing of request control elements

- Return an event notification control element that contains the Correlation Identifier field of the control element with the suspend bit in its Value field
- Wait for a resume event from the originating client before continuing.

The original request control element that contains the wait bit is not processed until the resume event is received from the client.

The following table shows the actions taken for different combinations of the wait and notification bits.

Notify	Wait	Action
0	0	Continue processing request control elements.
0	1	Suspend processing of the current control element and all subsequent request control elements from this source until a resume event control element is received from this source.
1	0	Return a notification to this source and process the request control element and all subsequent control elements from this source.
1	1	Return a notification to this source and suspend processing of the current control element and all subsequent control elements from this source until a resume event control element is received from this source.

Figure 4-4. Notify and Wait Actions

Chapter 5. Design Considerations

This section discusses the following areas, which are part of the Move mode architecture definition.

- Configuration
- Initialization
- Exception handling
- System management.

Configuration

Proper configuration of the system and the subsequent initialization of the individual units (system units and adapters), as well as the delivery service itself, are prerequisites for using the services. As such, it is considered to be a part of the Move mode architecture.

The process of initializing the delivery services requires that the following configuration information be defined and available at the time the system is loaded:

- Unit level
- System level
- Peer level.

Unit Level

Information from the unit-level configuration is specific to a single unit (system unit or adapter). The type of information provided is similar to that in an adapter description file (adapter description file is defined by the Setup Architecture). This includes, but is not limited to, the following information:

- Adapter ID
- Adapter base I/O address
- Adapter base I/O size
- Adapter base shared-memory address
- Adapter base shared-memory size
- Interrupt level
- DMA arbitration level.

© Copyright IBM Corp. 1991.

System Level

Information from the system-level configuration allows a pair of cooperating units (system or adapter) to communicate with each other. This information is used to set up and coordinate the delivery service. This information includes, but is not limited to the following:

- Unit ID

  The delivery support uses a Unit ID to associate a unit with the set of control areas in memory and I/O address space.

- Peer Unit ID

  At the system level, each unit requires an ID and IDs of the other units it sends control elements to or receives control elements from.

- Unit Type

  The delivery service supports the following operations:

  - System unit to adapter
  - Adapter to adapter
  - System unit to system unit.

  Both units must identify the configuration information for the unit type (system unit or adapter).

- Resource Definitions

  The delivery pipes require:

  - Configuration information describing the size and location of the delivery pipe
  - Operational information defining the use of the pipe
  - Information indicating the current state of the pipe.

  The following is a summary of the specific information required to allocate and use these resources:

  - Placement

    The pipe can be located anywhere in shared memory.

  - Size and location

    Both units need to know the size and location of the delivery pipes and the associated control areas.

- Signalling Control area locations

  The method of signalling between units is determined by the unit type and unit pairing (system unit to adapter, adapter to system unit, or adapter to adapter.). Both units must have the location of the Signalling Control area before signalling the other unit.

- Signalling conditions

  The delivery level supports a number of signalling options. Both units must be configured to support the options used. The following are some optional signalling conditions:

  - At specific time intervals.
  - A delivery pipe goes from empty to not empty.
  - A delivery pipe goes from not full to full.
  - A delivery pipe goes from not empty to empty.
  - A delivery pipe goes from full to not full.
  - A control element is placed in a delivery pipe.
  - A control element is removed from a delivery pipe.

- Surrogate offset information

  The offsets in the Surrogate Control areas (surrogate start-of-free and surrogate start-of-element) are used by both the enqueue and dequeue logics.

- Surrogate status information

  Information regarding the current state of the delivery pipes (full, empty, wrap, etc.) is maintained for both pipes.

- Timer information

  A watchdog or idle timer can be used to ensure reliable operation of the delivery level in case interrupts are masked or lost. If the timer is used, both units must support it and be aware of the frequency used and the source.

Peer Level

Information from the peer-level configuration is used by the system management to identify adapter-to-adapter pairings. This information is needed during peer-level initialization to provide each of the adapters with the parameters it needs to carry out unit-level initialization with its peers.

Initialization

Unit Level Initialization

Units are initialized by the system unit at power-on. The POST (power-on self-test) performs the initialization using the setup information maintained in non-volatile RAM.

System Level Initialization

System-level initialization is carried out by a system unit at IPL time and by an adapter at IML time. The system-level initialization is a multistage process. For system unit to adapter pairs, the first stage is carried out by the system unit, the second stage by the adapter, and the third stage by the system unit. In the third stage, the system unit verifies that initialization is complete and notifies the adapter of the status.

For adapter-to-adapter pairs, the initialization is done under the control of system management as a third party operation, using the initialization control element.

The first stage of the initialization is done by the system unit support when it is loaded. As part of its initialization process, this support constructs a configuration record for each bus unit it supports. The configuration record is constructed in shared memory and its address passed to each adapter during unit-level initialization. The format and content of the configuration record is the same for all adapters and is defined in Figure 5-1 on page 5-5.

During system-level initialization, the term *out-bound* refers to the direction of flow and identifies the delivery pipe used to pass control elements from the system unit to the adapter. The term *in-bound* refers to the direction of flow and identifies the delivery pipe used to pass control elements from the adapter to the system unit.

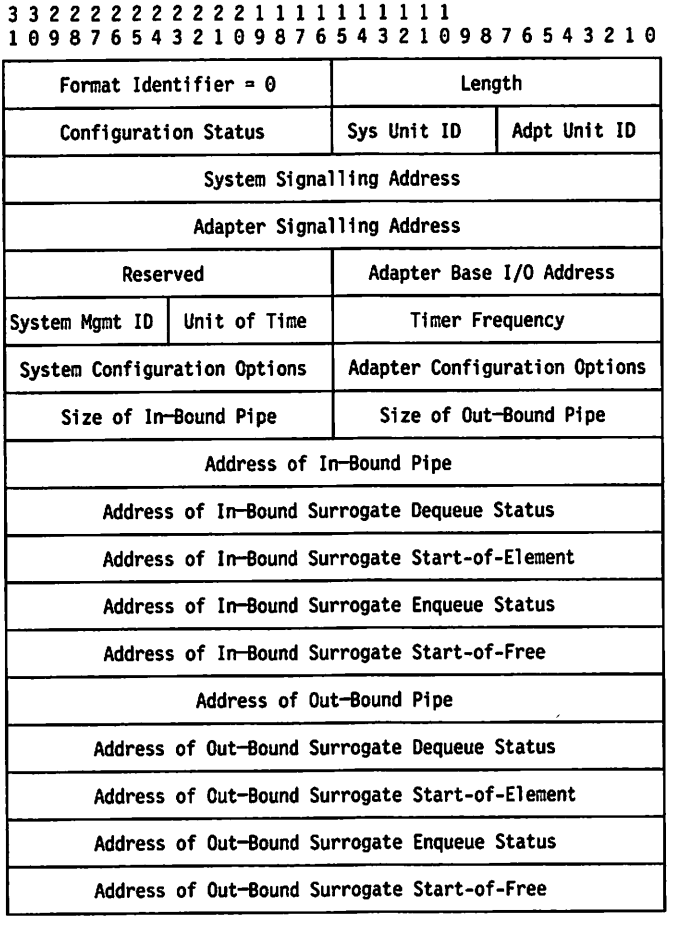


Figure 5-1. System Configuration Record Format

Format Identifier Field

This field identifies the format of the configuration record. Configuration records with a Format Identifier field value of 0 indicate a Move mode configuration record. All other values are reserved.

Length Field

This field contains the length of the configuration record in bytes, including the two bytes required for the Length field.

Configuration Status Field

This field holds status information that shows the current state of the initialization process. It is set to all 1s by the system unit after the configuration record is built.

System and Adapter Unit Identification Fields

These fields are set by the system unit and indicate the unit assignments (logical to physical mapping) as well as the unit pairing.

System Signalling Address Field

This field identifies the shared memory location of the Signalling Control area where the adapter is to place signalling information prior to signalling the system unit. The format and content of the Signalling Control area can be found in Figure 1-20 on page 1-30.

Adapter Signalling Address Field

This field identifies the shared memory location of the Signalling Control area where the system unit should place signalling information prior to signalling the adapter. The format and content of the Signalling Control area can be found in Figure 1-20 on page 1-30.

Adapter Base I/O Address Field

This field contains the base I/O address assigned to the adapter during Setup.

System Management ID Field

This field contains the entity identifier of the system-management entity in the system unit.

Unit of Time Field

This field indicates the granularity of the value in the Timer Value field. It is interpreted as:

01h - milliseconds
02h - 10s of milliseconds
04h - 100s of milliseconds
08h - seconds
10h - minutes

All other values are reserved.

Timer Frequency Field

The value in this field, when used with the Unit of Time field, indicates the frequency the system unit will present a timer-expiration signal of the paired unit. This field is valid only if the signal-on-timer-expiration bit is also set in the Configuration Options field of the appropriate bus unit.

System Configuration Options Field

This field contains information describing the system unit configuration options. The format and content of the field are found in the following figure.

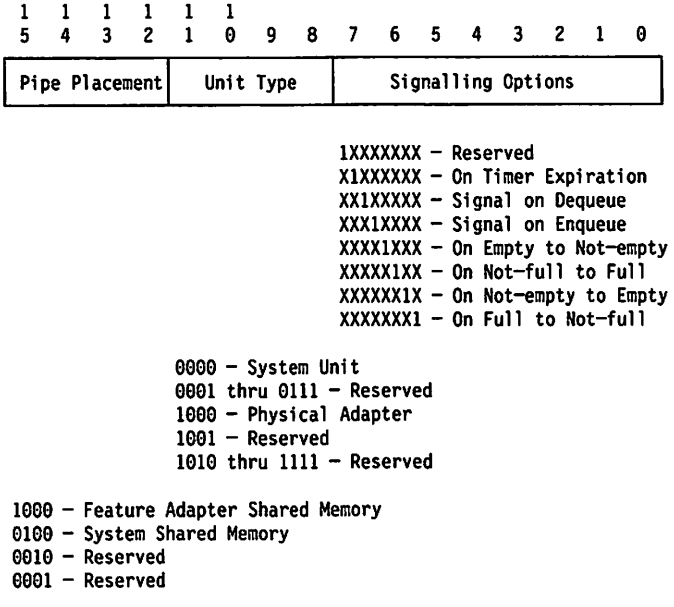


Figure 5-2. Configuration Field Format

Adapter Configuration Options Field

This field contains information describing the adapter configuration options. The format and content of the field are found in Figure 5-2 on page 5-7.

Size of In-Bound and Size of Out-Bound Pipe Fields

These fields define the amount of storage in bytes that is allocated for each pipe.

Note: For details on the roles of the remaining fields in the configuration record, refer to Figure 3-4 on page 3-9. The definitions for the pipe related control areas are found in "Delivery Pipe" on page 1-19.

Address of In-Bound Pipe Field

This field contains the address, in shared memory, for the pipe from the adapter to the system unit.

Address of In-Bound Surrogate Dequeue Status

This address contains the address, in shared memory, of the Surrogate Dequeue Status Control area that is written by the dequeue logic in the adapter and read by the enqueue logic in the system unit.

Address of In-Bound Surrogate Start-of-Element

This field contains the address, in shared memory, of the Surrogate Start of Element Control area which is written by the dequeue logic in the adapter and read by the enqueue logic in the system unit.

Address of In-Bound Surrogate Enqueue Status

This field contains the address, in shared memory, of the Surrogate Enqueue Status Control area that is written by the enqueue logic and read by the dequeue logic.

Address of In-Bound Surrogate Start-of-Free

This field contains the address, in shared memory, of the Surrogate Start of Free Control area that is written by the enqueue logic and read by the dequeue logic.

Address of Out-Bound Pipe Field

This field contains the base address, in shared memory, of the delivery pipe used to pass control elements from the system unit to the adapter.

Address of Out-Bound Surrogate Dequeue Status

This field contains the address, in shared memory, of the Surrogate Dequeue Status Control area that is written by the dequeue logic and read by the enqueue logic.

Address of Out-Bound Surrogate Start-of-Element

This field contains the address, in shared memory, of the Surrogate Start of Element Control area that is written by the dequeue logic and read by the enqueue logic.

Address of Out-Bound Surrogate Enqueue Status

This field contains the address, in shared memory, of the Surrogate Enqueue Status Control area that is written by the enqueue logic and read by the dequeue logic.

Address of Out-Bound Surrogate Start-of-Free

This field contains the address, in shared memory, of the Surrogate Start of Free Control area that is written by the enqueue logic and read by the dequeue logic.

The information required by system-level initialization can be obtained from any or all of the following places:

- Adapter description files and programs (ADF/ADP)
- Parameters provided at device-driver installation
- A separate configuration file created and maintained specifically for initialization
- Information compiled or assembled directly into the device driver itself.

In general, system-level initialization is concerned with the information required and not with how the information is obtained.

Peer Level Initialization

Peer-level, or adapter-to-adapter initialization, is performed following system-level initialization under the control of the system manager.

The configuration record used in peer-to-peer initialization is passed in an initialization request control element between the system-management entity in the system unit and the unit-management entity in the adapter (Figure 5-3 on page 5-11).

In the following figure, the term *Your* represents the unit that is the target of the initialization. The term *Peer* represents the unit that is the source of the initialization.

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1			
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0			
Format Identifier = 0		Length	
Configuration Status		Peer Unit ID	Your Unit ID
Peer Signalling Address			
Your Signalling Address			
Peer Base I/O Address		Your Base I/O Address	
System Mgmt ID	Unit of Time	Timer Frequency	
Peer Configuration Options		Your Configuration Options	
Size of In-Bound Pipe		Size of Out-Bound Pipe	
Address of In-Bound Pipe			
Address of In-Bound Surrogate Dequeue Status			
Address of In-Bound Surrogate Start-of-Element			
Address of In-Bound Surrogate Enqueue Status			
Address of In-Bound Surrogate Start-of-Free			
Address of Out-Bound Pipe			
Address of Out-Bound Surrogate Dequeue Status			
Address of Out-Bound Surrogate Start-of-Element			
Address of Out-Bound Surrogate Enqueue Status			
Address of Out-Bound Surrogate Start-of-Free			

Figure 5-3. Peer Configuration Record Format

Peer-level initialization is similar to system-level initialization, except that it is performed only for the adapter-to-adapter pairs, and is performed using the management entities in the affected units. System-level initialization must be complete prior to peer-level initialization. Basically, the system-management entity acts as the control point for peer-level initialization. Its function is:

1. Building an initial peer-configuration record for each of the peer adapters.
2. Sending the configuration record to the management entity (entity 0) in the first adapter of the pair for processing. This is

accomplished by placing the configuration record into an initialization request control element.

3. Waiting for a response to the initialization request from the management entity in the first adapter of the pair.

   Upon receiving the initialization request, the management entity in the adapter examines and optionally indicates changes to the configuration record. Then, it returns the configuration record to the system-management entity in the system unit using either a reply or error control element.

4. After receiving a reply, the system-management entity sends the configuration record with the first unit's information to the management entity in the second adapter for processing. This is also accomplished by placing the configuration record into an initialization request control element.

   The second unit follows the same procedure as the first unit and returns either a reply or error control element to the system-management entity after indicating any needed changes to the configuration record.

5. After receiving a reply control element with changes to the configuration record, the system manager will reissue the initialization request to the unit-management entity in the first adapter, completing the initialization and confirming the establishment of its delivery pipes.

6. The system-management entity notifies the unit-management entity in the second adapter that the delivery pipes are established and that the flow of information between the peer adapters can now begin.

Any failure during the peer-level-initialization process is reported by an error control element and is processed by the system-management entity.

The manner in which adapters or system units are initially loaded (IPLed) is not defined by the architecture.

Exception Handling

At the delivery level, certain delivery-related exception conditions must be handled by the enqueue logic and dequeue logic. If the following exceptions are detected during the specified operation, the following action will be taken.

- Enqueue of a control element
  - Delivery pipe is full - return an indication to the caller.
  - Destination unit does not exist - return an indication to the caller.
- Dequeue of a control element
  - Delivery pipe is empty - return an indication to the caller.
  - Destination entity does not exist - the action taken is dependent on the send or receive interface policy. The control element containing the error can be discarded or returned to the originator with the appropriate exception indication noted in the Status field of the error control element.
  - Destination entity not active - the action taken is dependent on the send or receive interface policy. The control element can be discarded, returned to the originator, or be queued for retrieval when the entity becomes active.

At the physical level, the following exception conditions must be detected and reported so that recovery actions can be taken by the affected units (system unit and adapter):

- Channel data and address parity exceptions
- Channel non-parity exceptions
- Master-dependent and slave-dependent exceptions
- Castrophic exceptions
- Channel-timeout exceptions.

The recovery action taken is based on the operation in progress at the time of the exception. Exceptions during the following operations will result in the indicated recovery actions:

- Memory read or write of a control element

- Retry the read or write of the control element until a specified threshold value has been reached.
- Notify the unit-management entity for possible higher-level recovery by the system manager.

- Memory read or write of the information in the Enqueue or Dequeue Control areas
  - Notify the unit-management entity for possible higher-level recovery by the system manager.
- Memory read or write of the information in the Signalling Control areas
  - Retry the read or write operation until the a specified threshold value has been reached.
  - Notify the unit-management entity for possible higher-level recovery by the system manager.
- I/O read or write of a port in I/O address space
  - Retry the read or write operation until the a specified threshold value has been reached.
  - Notify the unit-management entity for possible higher-level recovery by the system manager.
- Memory read or write of data
  - Retry the read or write operation until the a specified threshold value has been reached.
  - Retry the command in the control element which initiated the data read or write operation.
  - Notify the unit management entity for possible higher-level recovery by the system manager.

Management Relationships

The Move mode architecture also defines the relationship of the various parts of the delivery services to the unit-management services.

The unit-management services can be viewed as the platform on

which future delivery-management support and RAS-type services for bus masters are built.

The overall relationships for unit-level management are shown in Figure 5-4

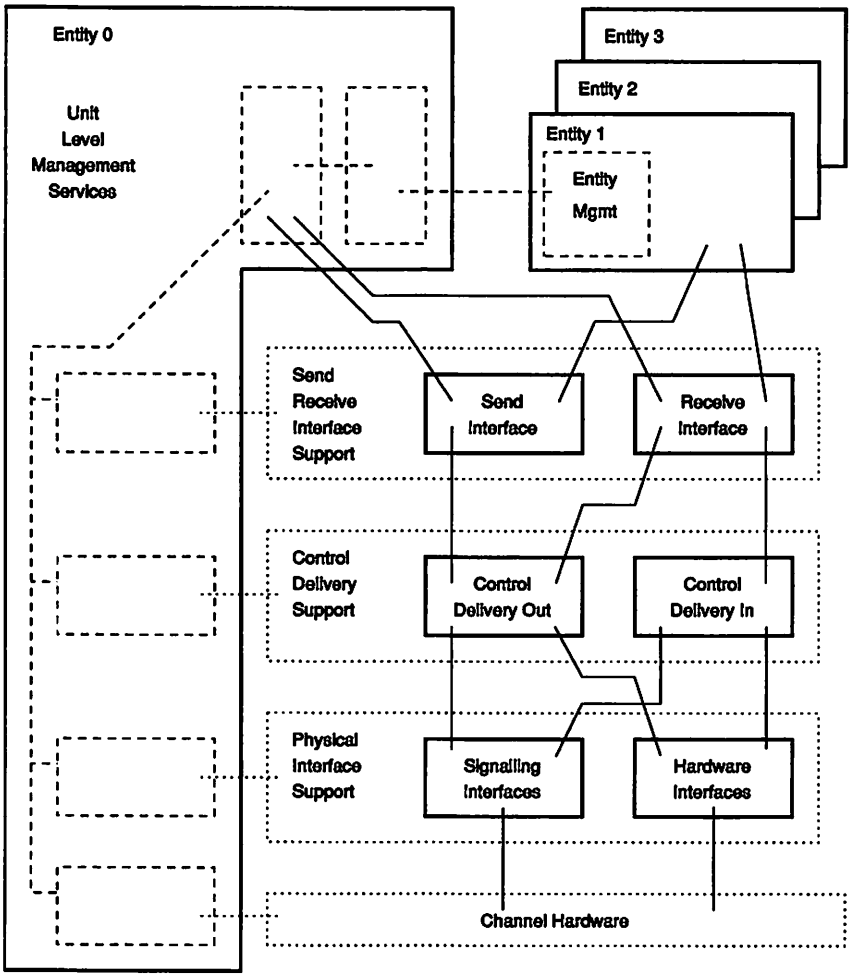


Figure 5-4. Unit Management Relationships

There are different forms of relationships between the various parts of the Move mode support and the unit-level management services.

- The management services are provided by a management entity that is defined as entity 0. There is an entity 0 in each unit in the configuration. This allows the overall management services to send and receive management-type control elements to and from the various units in the configuration after the delivery support is initialized.

  There can be control elements:

  - To reset the delivery pipes
  - To notify the overall-management entity of events in the unit
  - To report detail-error information for entities within the unit
  - To run tests on the unit.

- Unit management services need independent operations of entity-to-entity delivery and are required to have an independent unit of execution (thread).

- There is a portion of unit-management services that can be used as logic that actually handles the turn-around for requests with delivery-type errors. It also provides the coupling between the receive-interface logic and the enqueue logic for the pipe in the opposite direction.

- There can be other portions of management support to handle entity-level and physical-level error conditions as well as other management-related operations.

- Initialization can also be considered part of unit-management services.

The architecture requires that entity 0 be used for management-type services and not for client-type or server-type entities.

Chapter 6. Architectural Compliance

The architecture defines the Move mode protocols used to deliver control elements between pairs of functional entities located in different system units or adapters. Several functional entities can exist within a system unit or adapter. Different entities within a specific system unit can be located in different system units or adapters.

By complying with the architecture, two compliant system units or adapters, that are interconnected by a physical media and share addressable memory, can support the same base set of delivery protocols between entity pairs.

The architecture logically structures the delivery service into three functional layers:

- The physical level
- The delivery level
- The processing level.

The statements of compliance for each level define the requirements and the portion which can be configured. In addition to the protocols defined as base requirements, there are other optional architected protocols for a given implementation.

© Copyright IBM Corp. 1991.

Physical Level

The architecture defines several protocols for the physical-level primitives.

Base Protocols

To comply with the architecture, a product must:

- Implement the architected physical-level protocols.
- Maintain compatibility with the existing products by using an adapter that can use the physical level to support private protocols.

Additional Architected Protocols

Because the initialization protocol is product defined, the implementation of a physical-level protocol might not implement any port which is not used as part of the physical-level delivery logic.

Delivery Level

The delivery-level protocols include the signalling protocol, the enqueue protocol, the dequeue protocol, and the receive-interface protocol.

Base Protocols

To comply with the architecture, a product must:

- Implement the signalling protocol for the six state-change signals defined by the signalling logic in "Enqueue Protocol" on page 3-11, and "Dequeue Protocol" on page 3-19.
- Implement Signalling Control areas for each unit as defined in "Signalling Control Area" on page 1-29.
- Implement the delivery pipes for each pair of units supported as defined by the above references and Chapter 3, "Delivery Level" on page 3-1.

- Implement the delivery-pipe structure defined in "Delivery Pipe" on page 1-19.
- Implement the function defined for control element enqueue in "Enqueue Protocol" on page 3-11 and the Enqueue Surrogate Control areas defined in "Surrogate Enqueue Control Area" on page 1-24.
- Implement the function defined for control element dequeue in "Dequeue Protocol" on page 3-19 and the Dequeue Surrogate Control areas defined in "Surrogate Dequeue Control Area" on page 1-28.

Additional Architected Protocols

The architecture defines the signalling of state-changes based on device-dependent intervals for product implementation.

The architecture defines a management signal to indicate that initialization is complete. The support for this during initialization is optional for a product implementation.

The architecture defines receive-interface error handling which is optional for a product implementation.

Processing Level

In general, the entity-to-entity protocols are not defined by the architecture. However, to comply with the architecture, certain portions of the control element are supported by the delivery level.

Base Protocols

To comply with the architecture, a product must:

- Implement the control element structure, defined in "Control Elements" on page 1-32, on all control elements used.

  This header contains the following fields:

  - Format Identifier
  - Length
  - Common Indicators

- Source Unit
- Source Entity
- Destination Unit
- Destination Entity
- Correlation ID.

- Implement the bits of the Common Indicators field if these functions are used.
- Implement the architected form of the control element when the same functions are used.

Additional Architected Protocols

The architecture defines a number of common-use control elements which are optional.

Management

The architecture defines protocols for configuring the delivery services and for initializing each unit (system unit and adapter) that will use the delivery service.

Base Protocols

To comply with the architecture, a product must :

- Implement the configuration record as defined in "Configuration" on page 5-1.
- Implement entity 0 as a management entity.

Additional Architected Protocols

The protocols for using the configuration record are device-dependent.

Appendix A. C Language

```
/*
 +-------------------------------------------------------------------+
 | Header File: MOVMODE.H                                            |
 |                                                                   |
 | Description: Contains the include files, constants, data structures,|
 |              macro definitions, and function declarations common to |
 |              all Move mode Delivery Services components.          |
 |                                                                   |
 | Status Information:                                               |
 |                                                                   |
 |     Created: 04/01/89                                             |
 |     Revised: 01/10/90                                             |
 |     Revised: 07/17/90                                             |
 +-------------------------------------------------------------------+
*/
#ifndef _MOVMODE_H
#define _MOVMODE-H
/*
                +---------------------------------------+
                |        USEFUL TYPE DEFINITIONS        |
                +---------------------------------------+
*/
typedef int                 BOOLEAN;
typedef unsigned char       BYTE;       /* full 8 bit char           */
typedef unsigned char      *pBYTE;      /* pointer to 8 bit char     */
typedef unsigned char  far *fpBYTE;     /* far pointer to 8 bit char */
typedef unsigned short      WORD;       /* full 16 bit word          */
typedef unsigned short     *pWORD;      /* pointer to 16 bit word    */
typedef unsigned short far *fpWORD;     /* far pointer to 16 bit word */
typedef unsigned long       DWORD;      /* full 32 bit word          */
typedef unsigned long      *pDWORD;     /* pointer to 32 bit word    */
typedef unsigned long  far *fpDWORD;    /* far pointer to 32 bit word */

/*
                +---------------------------------------+
                |             INCLUDE FILES             |
                +---------------------------------------+
*/
#include <stdio.h>              /* standard stream I/O               */

#ifdef __TURBOC__
   #include <alloc.h>           /* Turbo C memory allocation         */
   #include <mem.h>             /* memory manipulation               */
#else
   #include <malloc.h>          /* MS/IBM C memory allocation        */
   #include <memory.h>          /* memory manipulation               */
#endif

#include <stdlib.h>             /* standard library                  */
#include <ctype.h>              /* char classification/conversion    */

#include "element.h"            /* Control elements                  */
#include "pipe.h"               /* Pipe (delivery queues)            */
```

© Copyright IBM Corp. 1991.

```
/*
                +--------------------------------------+
                |          DEFINED CONSTANTS           |
                +--------------------------------------+
*/
#define FALSE   0
#define TRUE    1

#define OFF     0
#define ON      1

#define SUCCESS 1
#define FAILURE 0

#endif
```

```
/*
 +-------------------------------------------------------------------+
 | Header File: pipe.h                                               |
 |                                                                   |
 | Description: Contains the data structure and definitions for the  |
 |              following control areas in shared memory address     |
 |              space:                                               |
 |               - configuration record                              |
 |               - local enqueue control area                        |
 |               - surrogate enqueue control areas                   |
 |               - local dequeue control area                        |
 |               - surrogate dequeue control areas                   |
 |               - signalling areas                                  |
 |                                                                   |
 | Status Information:                                               |
 |                                                                   |
 |     Created: 05/02/89                                             |
 |     Revised: 01/10/90                                             |
 |     Revised: 07/17/90                                             |
 +-------------------------------------------------------------------+
*/
#ifndef _PIPE_H
#define _PIPE_H
/*
               +--------------------------------------+
               |           DATA STRUCTURES            |
               +--------------------------------------+
*/

/* -----
    The flags within the Configuration Record are used to indicate
    the specific options to be used between a pair of bus units.
                                                          ----- */

#ifdef __TURBOC__

typedef struct {                /* Configuration Options             */
                                /*  Physical Placement of the Pipe   */
  unsigned adp_mem    : 1;      /*  - bit 15 - in adapter            */
                                /*    0 = outbound pipe in adapter   */
                                /*    1 = inbound pipe in adapter    */
  unsigned sys_mem    : 1;      /*  - bit 14 - in system unit        */
                                /*    0 = outbound pipe in system    */
                                /*    1 = inbound pipe in system     */
  unsigned b13        : 1;      /*  - bit 13 - reserved              */
  unsigned b12        : 1;      /*  - bit 12 - reserved              */
  unsigned unit_type  : 4;      /*  Bus Unit Type                    */
                                /*  - 0000 - system unit             */
                                /*  - 0001 thru 0111 - reserved      */
                                /*  - 1000 - adapter type 1          */
                                /*  - 1001 - adapter type 2          */
                                /*  - 1010 thru 1111 - reserved      */
                                /*  Signalling Conditions            */
  unsigned b7         : 1;      /*  - bit 7  - reserved              */
  unsigned on_timer   : 1;      /*  - bit 6  - on timer expiration   */
  unsigned on_notfull : 1;      /*  - bit 5  - on full to non-full   */
  unsigned on_deque   : 1;      /*  - bit 4  - on dequeue element    */
  unsigned on_enque   : 1;      /*  - bit 3  - on enqueue element    */
```

```
  unsigned on_empty   : 1;      /*  - bit 2  - on pipe empty          */
  unsigned on_full    : 1;      /*  - bit 1  - on pipe full           */
  unsigned on_notempty: 1;      /*  - bit 0  - on empty to non-empty  */
} CFG_OPT;

#else

typedef struct {                /* Configuration Options              */
                                /*  Signalling Conditions             */
  unsigned on_notempty: 1;      /*  - bit 0  - on empty to non-empty  */
  unsigned on_full    : 1;      /*  - bit 1  - on pipe full           */
  unsigned on_empty   : 1;      /*  - bit 2  - on pipe empty          */
  unsigned on_enque   : 1;      /*  - bit 3  - on enqueue element     */
  unsigned on_deque   : 1;      /*  - bit 4  - on dequeue element     */
  unsigned on_notfull : 1;      /*  - bit 5  - on full to non-full    */
  unsigned on_timer   : 1;      /*  - bit 6  - on timer expiration    */
  unsigned b7         : 1;      /*  - bit 7  - reserved               */
  unsigned unit_type  : 4;      /*  Bus Unit Type                     */
                                /*  - 0000 - system unit              */
                                /*  - 0001 thru 0111 - reserved       */
                                /*  - 1000 - adapter type 1           */
                                /*  - 1001 - adapter type 2           */
                                /*  - 1010 thru 1111 - reserved       */
  unsigned b12        : 1;      /*  - bit 12 - reserved               */
  unsigned b13        : 1;      /*  - bit 13 - reserved               */
                                /*  Physical Placement of the Pipe    */
  unsigned sys_mem    : 1;      /*  - bit 14 - in system unit         */
                                /*    0 = outbound pipe in system     */
                                /*    1 = inbound pipe in system      */
  unsigned adp_mem    : 1;      /*  - bit 15 - in adapter             */
                                /*    0 = outbound pipe in adapter    */
                                /*    1 = inbound pipe in adapter     */
} CFG_OPT;

#endif

/* -----
    The fields within the Configuration Record (CFG_RCD) are used
    at initialization time to identify the location of control areas
    in shared memory address space and the options to be used when
    using the delivery service between a pair of bus units.
                                                              ----- */

typedef struct {                /* Configuration Record               */
  WORD    fid;                  /*  - Format Identifier               */
  WORD    len;                  /*  - length field                    */
  WORD    status;               /*  - configuration status field      */
  BYTE    sid;                  /*  - system unit identification field*/
  BYTE    aid;                  /*  - adapter identification field    */
  fpDWORD ss_addr;              /*  - system signalling address field */
  fpDWORD as_addr;              /*  - adapter signalling address field*/
  WORD    peer_io_addr;         /*  - base I/O addr - peer (from)     */
  WORD    base_io_addr;         /*  - base I/O addr - adapter (to)    */
  BYTE    smgmt_eid;            /*  - system manager entity id field  */
  BYTE    time_units;           /*  - units of time                   */
  WORD    time_freq;            /*  - timer frequency                 */
  CFG_OPT sys_config;           /*  - system unit configuration options*/
  CFG_OPT adp_config;           /*  - adapter configuration options   */
```

```
  WORD    out_size;           /* - out-bound pipe size (sys to adp) */
  WORD    in_size;            /* - in-bound pipe size (adp to sys)  */
  fpDWORD in_pipe;            /* - in-bound pipe address            */
  fpDWORD in_sds;             /* - in-bound surrogate dequeue       */
                              /*      status area address           */
  fpDWORD in_sse;             /* - in-bound surrogate start of      */
                              /*      elements control area         */
  fpDWORD in_ses;             /* - in-bound surrogate enqueue       */
                              /*      status area address           */
  fpDWORD in_ssf;             /* - in-bound surrogate start of      */
                              /*      free control area address     */
  fpDWORD out_pipe;           /* - out-bound pipe address           */
  fpDWORD out_sds;            /* - out-bound surrogate dequeue      */
                              /*      status area address           */
  fpDWORD out_sse;            /* - out-bound surrogate start of     */
                              /*      elements control area address */
  fpDWORD out_ses;            /* - out-bound surrogate enqueue      */
                              /*      status area address           */
  fpDWORD out_ssf;            /* - out-bound surrogate start of     */
                              /*      free control area address     */
} CFG_RCD;

/* -----
    The flags within the Enqueue Status Word are used by the Enqueue
    Logic to maintain local state information at the enqueue end of
    the pipe.
                                                              ----- */
#ifdef __TURBOC__

typedef struct {              /* Enqueue Status Word Flags          */
  unsigned b15      : 1;      /*  - bit 15 - reserved               */
  unsigned b14      : 1;      /*  - bit 14 - reserved               */
  unsigned b13      : 1;      /*  - bit 13 - reserved               */
  unsigned b12      : 1;      /*  - bit 12 - reserved               */
  unsigned queued   : 1;      /*  - bit 11 - element stored         */
  unsigned b10      : 1;      /*  - bit 10 - reserved               */
  unsigned b9       : 1;      /*  - bit 9  - reserved               */
  unsigned wrap     : 1;      /*  - bit 8  - enqueue wrap           */
  unsigned b7       : 1;      /*  - bit 7  - reserved               */
  unsigned b6       : 1;      /*  - bit 6  - reserved               */
  unsigned b5       : 1;      /*  - bit 5  - reserved               */
  unsigned b4       : 1;      /*  - bit 4  - reserved               */
  unsigned b3       : 1;      /*  - bit 3  - reserved               */
  unsigned b2       : 1;      /*  - bit 2  - reserved               */
  unsigned empty    : 1;      /*  - bit 1  - empty                  */
  unsigned full     : 1;      /*  - bit 0  - queue full             */
} LES_FLAGS;

#else

typedef struct {              /* Enqueue Status Word Flags          */
  unsigned full     : 1;      /*  - bit 0  - queue full             */
  unsigned empty    : 1;      /*  - bit 1  - empty                  */
  unsigned b2       : 1;      /*  - bit 2  - reserved               */
  unsigned b3       : 1;      /*  - bit 3  - reserved               */
  unsigned b4       : 1;      /*  - bit 4  - reserved               */
  unsigned b5       : 1;      /*  - bit 5  - reserved               */
  unsigned b6       : 1;      /*  - bit 6  - reserved               */
```

```
  unsigned b7        : 1;      /*  - bit 7  - reserved              */
  unsigned wrap      : 1;      /*  - bit 8  - enqueue wrap          */
  unsigned b9        : 1;      /*  - bit 9  - reserved              */
  unsigned b10       : 1;      /*  - bit 10 - reserved              */
  unsigned queued    : 1;      /*  - bit 11 - element stored        */
  unsigned b12       : 1;      /*  - bit 12 - reserved              */
  unsigned b13       : 1;      /*  - bit 13 - reserved              */
  unsigned b14       : 1;      /*  - bit 14 - reserved              */
  unsigned b15       : 1;      /*  - bit 15 - reserved              */
} LES_FLAGS;

#endif

/* -----
    The fields within the Local Enqueue Control Area (LECA) identify
    the location of the pipe, indicate the status of the pipe, and
    provide state information identifying the starting and ending
    offsets of control elements within the pipe.
                                                                ----- */

typedef struct {                /* Local Enqueue Control Area       */
  fpDWORD base;                 /* - Pipe Address                   */
  WORD   we;                    /* - offset to wrap element         */
  union {                       /* - enqueue status                 */
     WORD      fld;             /*   as a word                      */
     LES_FLAGS flg;             /*   as individual flags            */
  } es;
  WORD   sf;                    /* - start of free space in pipe    */
  WORD   ef;                    /* - end of free space in pipe      */
  WORD   end;                   /* - offset to end (of queue space) */
  WORD   top;                   /* - offset to top (wrap point)     */
} LECA;

/* -----
    The flags within the Surrogate Enqueue Status Word are used by
    the Enqueue Logic to convey the state of the pipe to the Dequeue
    Logic at the other end of the pipe.
                                                                ----- */
#ifdef __TURBOC__

typedef struct {                /* Surrogate Enqueue Status Word Flag */
  unsigned b15       : 1;      /*  - bit 15 - reserved             */
  unsigned b14       : 1;      /*  - bit 14 - reserved             */
  unsigned b13       : 1;      /*  - bit 13 - reserved             */
  unsigned b12       : 1;      /*  - bit 12 - reserved             */
  unsigned b11       : 1;      /*  - bit 11 - reserved             */
  unsigned b10       : 1;      /*  - bit 10 - reserved             */
  unsigned b9        : 1;      /*  - bit 9  - reserved             */
  unsigned wrap      : 1;      /*  - bit 8  - wrap of pipe occurred */
  unsigned b7        : 1;      /*  - bit 7  - reserved             */
  unsigned b6        : 1;      /*  - bit 6  - reserved             */
  unsigned b5        : 1;      /*  - bit 5  - reserved             */
  unsigned b4        : 1;      /*  - bit 4  - reserved             */
  unsigned b3        : 1;      /*  - bit 3  - reserved             */
  unsigned b2        : 1;      /*  - bit 2  - reserved             */
  unsigned b1        : 1;      /*  - bit 1  - reserved             */
  unsigned full      : 1;      /*  - bit 0  - pipe is full         */
} SES_FLAGS;
```

```
#else

typedef struct {                    /* Surrogate Enqueue Status Word Flag */
  unsigned full       : 1;          /*  - bit 0  - pipe is full           */
  unsigned b1         : 1;          /*  - bit 1  - reserved               */
  unsigned b2         : 1;          /*  - bit 2  - reserved               */
  unsigned b3         : 1;          /*  - bit 3  - reserved               */
  unsigned b4         : 1;          /*  - bit 4  - reserved               */
  unsigned b5         : 1;          /*  - bit 5  - reserved               */
  unsigned b6         : 1;          /*  - bit 6  - reserved               */
  unsigned b7         : 1;          /*  - bit 7  - reserved               */
  unsigned wrap       : 1;          /*  - bit 8  - wrap of pipe occurred  */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
  unsigned b10        : 1;          /*  - bit 10 - reserved               */
  unsigned b11        : 1;          /*  - bit 11 - reserved               */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
} SES_FLAGS;

#endif

typedef struct {                    /* Surrogate Enqueue Control Area     */
  union {                           /* - surrogate enqueue status         */
     WORD      fld;                 /*   as a word                        */
     SES_FLAGS flg;                 /*   as individual flags              */
  } ses;
  WORD  ssf;                        /* - offset to start of free space    */
  } SECA;

/* -----
    The flags within the Dequeue Status Word are used by the Dequeue
    Logic to maintain local state information at the dequeue end of
    pipe.
                                                                  ----- */
#ifdef __TURBOC__

typedef struct {                    /* Dequeue Status Word Flags          */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned dequeued   : 1;          /*  - bit 11 - element removed        */
  unsigned prempt     : 1;          /*  - bit 10 - stop dequeue operation */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
  unsigned wrap       : 1;          /*  - bit 8  - wrap element toggle    */
  unsigned b7         : 1;          /*  - bit 7  - reserved               */
  unsigned b6         : 1;          /*  - bit 6  - reserved               */
  unsigned b5         : 1;          /*  - bit 5  - reserved               */
  unsigned b4         : 1;          /*  - bit 4  - reserved               */
  unsigned b3         : 1;          /*  - bit 3  - reserved               */
  unsigned b2         : 1;          /*  - bit 2  - reserved               */
  unsigned empty      : 1;          /*  - bit 1  - pipe is empty          */
  unsigned full       : 1;          /*  - bit 0  - full                   */
} LDS_FLAGS;
```

```
#else

typedef struct {                    /* Dequeue Status Word Flags          */
  unsigned full       : 1;          /*  - bit 0  - full                   */
  unsigned empty      : 1;          /*  - bit 1  - pipe is empty          */
  unsigned b2         : 1;          /*  - bit 2  - reserved               */
  unsigned b3         : 1;          /*  - bit 3  - reserved               */
  unsigned b4         : 1;          /*  - bit 4  - reserved               */
  unsigned b5         : 1;          /*  - bit 5  - reserved               */
  unsigned b6         : 1;          /*  - bit 6  - reserved               */
  unsigned b7         : 1;          /*  - bit 7  - reserved               */
  unsigned wrap       : 1;          /*  - bit 8  - wrap element toggle    */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
  unsigned prempt     : 1;          /*  - bit 10 - stop dequeue operation */
  unsigned dequeued   : 1;          /*  - bit 11 - element removed        */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
} LDS_FLAGS;

#endif

/* -----
    The fields within the Local Dequeue Control Area (LDCA) identify
    the location of the pipe, indicate the status of the pipe, and
    provide state information identifying the starting and ending
    offsets of control elements within the pipe.
                                                                  ----- */

typedef struct {                    /* Local Dequeue Control Area         */
  fpDWORD base;                     /* - Pipe Address                     */
  WORD   we;                        /* - offset to wrap element           */
  union {                           /* - Dequeue status                   */
     WORD      fld;                 /*   as a word                        */
     LDS_FLAGS flg;                 /*   as individual flags              */
  } ds;
  WORD   se;                        /* - start of elements in pipe        */
  WORD   ee;                        /* - end of elements in pipe          */
  WORD   end;                       /* - offset to end (of queue space)   */
  WORD   top;                       /* - offset to top (wrap point)       */
} LDCA;

/* -----
    The flags within the Surrogate Dequeue Status Word are used by
    the Dequeue Logic to convey the state of the pipe to the Enqueue
    Logic at the other end of the pipe.
                                                                  ----- */
#ifdef __TURBOC__

typedef struct {                    /* Surrogate Dequeue Status Flags     */
  unsigned b15        : 1;          /*  - bit 15 - reserved               */
  unsigned b14        : 1;          /*  - bit 14 - reserved               */
  unsigned b13        : 1;          /*  - bit 13 - reserved               */
  unsigned b12        : 1;          /*  - bit 12 - reserved               */
  unsigned b11        : 1;          /*  - bit 11 - reserved               */
  unsigned b10        : 1;          /*  - bit 10 - reserved               */
  unsigned b9         : 1;          /*  - bit 9  - reserved               */
```

```
  unsigned wrap       : 1;      /*  - bit 8  - wrap of pipe occurred  */
  unsigned b7         : 1;      /*  - bit 7  - reserved               */
  unsigned b6         : 1;      /*  - bit 6  - reserved               */
  unsigned b5         : 1;      /*  - bit 5  - reserved               */
  unsigned b4         : 1;      /*  - bit 4  - reserved               */
  unsigned b3         : 1;      /*  - bit 3  - reserved               */
  unsigned b2         : 1;      /*  - bit 2  - reserved               */
  unsigned empty      : 1;      /*  - bit 1  - pipe is empty          */
  unsigned b0         : 1;      /*  - bit 0  - reserved               */
} SDS_FLAGS;

#else

typedef struct {                /* Surrogate Dequeue Status Flags     */
  unsigned b0         : 1;      /*  - bit 0  - reserved               */
  unsigned empty      : 1;      /*  - bit 1  - pipe is empty          */
  unsigned b2         : 1;      /*  - bit 2  - reserved               */
  unsigned b3         : 1;      /*  - bit 3  - reserved               */
  unsigned b4         : 1;      /*  - bit 4  - reserved               */
  unsigned b5         : 1;      /*  - bit 5  - reserved               */
  unsigned b6         : 1;      /*  - bit 6  - reserved               */
  unsigned b7         : 1;      /*  - bit 7  - reserved               */
  unsigned wrap       : 1;      /*  - bit 8  - wrap of pipe occurred  */
  unsigned b9         : 1;      /*  - bit 9  - reserved               */
  unsigned b10        : 1;      /*  - bit 10 - reserved               */
  unsigned b11        : 1;      /*  - bit 11 - reserved               */
  unsigned b12        : 1;      /*  - bit 12 - reserved               */
  unsigned b13        : 1;      /*  - bit 13 - reserved               */
  unsigned b14        : 1;      /*  - bit 14 - reserved               */
  unsigned b15        : 1;      /*  - bit 15 - reserved               */
} SDS_FLAGS;

#endif

typedef struct {                /* Surrogate Dequeue Control Area     */
  union {                       /* - surrogate dequeue status         */
     WORD      fld;             /*   as a word                        */
     SDS_FLAGS flg;             /*   as individual flags              */
  } sds;
  WORD   sse;                   /* - offset to start of elements      */
} SDCA;

/* -----
    The fields within the Signalling Area (SIGNAL) are used to identify
    the source as well as the reason for signalling from one bus unit
    to another.
                                                                ----- */
typedef struct {                /* Signalling Control Area            */
  BYTE   nque;                  /*   - enqueue state change           */
  BYTE   mgmt;                  /*   - management state change        */
  BYTE   byte2;                 /*   - reserved                       */
  BYTE   dque;                  /*   - dequeue state change           */
} SCA;

#endif
```

```
/*
 +-------------------------------------------------------------------+
 | Header File: ELEMENT.H                                            |
 |                                                                   |
 | Description: Contains the data structure definitions for the basic |
 |              control elements.                                    |
 |                                                                   |
 | Status Information:                                               |
 |                                                                   |
 |     Created: 05/02/89                                             |
 |     Revised: 01/10/90                                             |
 |     Revised: 07/17/90                                             |
 +-------------------------------------------------------------------+
*/
#ifndef _ELEMENT_H
#define _ELEMENT_H
/*

              +--------------------------------------+
              |          DEFINED CONSTANTS           |
              +--------------------------------------+
*/
          /* ----- Format Identifier field values ----- */

#define FID_MM     0            /* SCB Move Mode Format */

      /* ----- Control Element Type field Element IDs ----- */

#define REQ_EL     0            /* request */
#define REP_EL     1            /* reply   */
#define EVT_EL     2            /* event   */
#define ERR_EL     3            /* error   */

#define ERR_EL_LEN 20           /* length of error element          */
#define DEL_EL_LEN 6            /* length of delivery wrap element  */

      /* ----- Control Element Type field Function Codes ----- */

                                /* 0 and 1 are reserved             */
#define FC_INIT      2          /* initialize                       */
                                /* 3 is reserved                    */
#define FC_READ      4          /* read                             */
#define FC_READL     5          /* read list                        */
#define FC_READI     6          /* read immediate                   */
#define FC_WRITE     7          /* write                            */
#define FC_WRITEL    8          /* write list                       */
#define FC_WRITEI    9          /* write immediate                  */
#define FC_EXL      10          /* execute list                     */
#define FC_MARK     11          /* mark                             */
#define FC_CANCEL   12          /* cancel                           */
#define FC_PARAM    13          /* parameters                       */
#define FC_RDCFG    14          /* read configuration               */
#define FC_DIAG     15          /* run diagnostics                  */
                                /* 16 is reserved                   */
#define FC_RESUME   17          /* resume (event)                   */
#define FC_NOTIFY   18          /* notification (event)             */
#define FC_INFORM   19          /* user status information (event)  */
```

```
                                /* 20 thru 62 are reserved              */
#define FC_WRAP     63          /* wrap of queue (internal)             */

      /* ----- Control Element Value field Error Codes ----- */

#define NO_RECV_PENDING 1

/*
                +--------------------------------------+
                |           DATA STRUCTURES            |
                +--------------------------------------+
*/
#ifdef __TURBOC__
typedef struct {                /* Common Indicators Field              */
  unsigned id       : 2;        /* - bit 15 - 14 identifier (ID)        */
  unsigned suppress : 1;        /* - bit 13 suppress reply flag (S)     */
  unsigned chain    : 2;        /* - bit 12 - 11 chaining flag (C)      */
  unsigned indirect : 1;        /* - bit 10 indirect flag  (I)          */
  unsigned notify   : 1;        /* - bit 9 notification flag  (N)       */
  unsigned wait     : 1;        /* - bit 8 wait flag (W)                */
  unsigned expedite : 1;        /* - bit 7 expedited flag (E)           */
  unsigned function : 7;        /* - bit 6 - 0 function code (FC)       */
} CIND_FLD;

#else

typedef struct {                /* Common Indicators Field              */
  unsigned function : 7;        /* - bit 6 - 0 function code (FC)       */
  unsigned expedite : 1;        /* - bit 7 expedited flag (E)           */
  unsigned wait     : 1;        /* - bit 8 wait flag (W)                */
  unsigned notify   : 1;        /* - bit 9 notification flag (N)        */
  unsigned indirect : 1;        /* - bit 10 indirect flag (I)           */
  unsigned chain    : 2;        /* - bit 12 - 11 chaining flag (C)      */
  unsigned suppress : 1;        /* - bit 13 suppress reply flag (S)     */
  unsigned id       : 2;        /* - bit 15 - 14 identifier (ID)        */
} CIND_FLD;

#endif

typedef struct {                /* Element Source/Destination Field     */
  BYTE eid;                     /* - entity identifier                  */
  BYTE uid;                     /* - unit identifier                    */
} ADDR_FLD;

typedef struct {                /* Element descriptor                   */
  WORD     fid;                 /* - Format Identifier                  */
  WORD     len;                 /* - length field                       */
  CIND_FLD cind;                /* - common indicators field            */
  WORD     res1;                /* - reserved for future use            */
  ADDR_FLD dest;                /* - destination field                  */
  ADDR_FLD srce;                /* - source field                       */
  DWORD    cor_id;              /* - correlation field                  */
  DWORD    val_fld;             /* - value field                        */
} ELEMENT;
#endif
```

Notes:

Appendix B. Assembler Language

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                       *
;* INCLUDE FILE: MOVMODE.INC                                             *
;*                                                                       *
;*    COPYRIGHT: (C) Copyright IBM Corporation 1989, 1990.  All          *
;*               rights reserved.                                        *
;*                                                                       *
;*    REV LEVEL: 1.1                                                     *
;*                                                                       *
;*  DESCRIPTION: Contains constants, data structures, and macro          *
;*               definitions common to all Move Mode Delivery            *
;*               Services components.                                    *
;*                                                                       *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
INCLUDE "ELEMENT.INC"           ;control element
INCLUDE "PIPE.INC"              ;delivery pipes
```

© Copyright IBM Corp. 1991.

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                       *
;* INCLUDE FILE: PIPE.INC                                                *
;*                                                                       *
;*    COPYRIGHT: (C) Copyright IBM Corporation 1989, 1990.  All          *
;*               rights reserved.                                        *
;*                                                                       *
;*    REV LEVEL: 1.1                                                     *
;*                                                                       *
;*  DESCRIPTION: Contains data structure and definitions for the         *
;*               following control areas in shared memory address        *
;*               space:                                                  *
;*                                                                       *
;*               - configuration record                                  *
;*               - local enqueue control area                            *
;*               - surrogate enqueue control areas                       *
;*               - local dequeue control area                            *
;*               - surrogate dequeue control areas                       *
;*               - signalling areas                                      *
;*                                                                       *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;*             ****************************************
;*             *          DATA STRUCTURES             *
;*             ****************************************

;*  Flags within the Configuration Word (CFG_WRD) are used to indicate
;*  specific options to be used between a pair of bus units.

CFG_OPT RECORD                  ;Physical Placement of the Pipe
        adp_mem:1,              ; - bit 15 - in feature adapter
                                ;   0 = outbound pipe in adapter
                                ;   1 = inbound pipe in adapter
        sys_mem:1,              ; - bit 14 - in system unit
                                ;   0 = outbound pipe in system
                                ;   1 = inbound pipe in system
        cfg_b13:1,              ; - bit 13 - reserved
        cfg_b12:1,              ; - bit 12 - reserved
        unit_type:4,            ;Bus Unit Type (bits 11-8)
                                ; - 0000 - system unit
                                ; - 0001 thru 0111 - reserved
                                ; - 1000 - adapter type 1
                                ; - 1001 - adapter type 2
                                ; - 1010 thru 1111 - reserved
                                ;Signalling Conditions
        cfg_b7:1,               ; - bit 7  - reserved
        on_timer:1,             ; - bit 6  - on timer expiration
        on_notfull:1,           ; - bit 5  - on full to non-full
        on_deque:1,             ; - bit 4  - on dequeue element
        on_enque:1,             ; - bit 3  - on enqueue element
        on_empty:1,             ; - bit 2  - on pipe empty
        on_full:1,              ; - bit 1  - on pipe full
        on_notempty:1           ; - bit 0  - on empty to non-empty
CFG_OPT ends

;*  Fields within the Configuration Record (CFG_RCD) are used
;*  at initialization time to identify the location of control areas
```

```
;*  in shared memory address space and the options to be used when
;*  using the delivery service between a pair of bus units.

CFG_RCD struc                   ;Configuration Record
   fid          dw  ?           ;- format identifier
   len          dw  ?           ;- length field
   status       dw  ?           ;- configuration status field
   sid          db  ?           ;- system unit id field
   aid          db  ?           ;- adapter id field
   ss_addr      dd  ?           ;- system signalling address field
   as_addr      dd  ?           ;- adapter signalling address field
   peer_io_addr dw  ?           ;- base I/O address - peer (from)
   base_io_addr dw  ?           ;- base I/O address - adapter (to)
   smgmt_eid    db  ?           ;- system manager entity id field
   time_units   db  ?           ;- units of time
   time_freq    db  ?           ;- timer frequency
   sys_config   CFG_OPT <>      ;- system unit configuration options
   adp_config   CFG_OPT <>      ;- adapter configuration options
   out_size     dw  ?           ;- out-bound pipe size
   in_size      dw  ?           ;- in-bound pipe size
   in_pipe      dd  ?           ;- in-bound pipe address
   in_sds       dd  ?           ;- in-bound surrogate dequeue
                                ;    status area
   in_sse       dd  ?           ;- in-bound surrogate start of
                                ;    elements control area
   in_ses       dd  ?           ;- in-bound surrogate enqueue
                                ;    status area
   in_ssf       dd  ?           ;- in-bound surrogate start of
                                ;    free control area
   out_pipe     dd  ?           ;- out-bound pipe address
   out_sds      dd  ?           ;- out-bound surrogate dequeue
                                ;    status area
   out_sse      dd  ?           ;- out-bound surrogate start of
                                ;    elements control area
   out_ses      dd  ?           ;- out-bound surrogate enqueue
                                ;    status area
   out_ssf      dd  ?           ;- out-bound surrogate start of
                                ;    free control area
CFG_RCD ends

;*
;*  Flags within Local Enqueue Status Flag (LES_FLAGS) are used by
;*  Enqueue Logic to maintain local state information at enqueue
;*  end of the pipe.

LES_FLAGS RECORD                ;Local Enqueue Status Word Flags
        lesb15:1,               ;- bit 15 - reserved
        lesb14:1,               ;- bit 14 - reserved
        lesb13:1,               ;- bit 13 - reserved
        lesd12:1,               ;- bit 12 - reserved
        lesqueued:1,            ;- bit 11 - element stored
        lesb10:1,               ;- bit 10 - reserved
        lesb9:1,                ;- bit 9  - reserved
        leswrap:1,              ;- bit 8  - enqueue wrap
        lesb7:1,                ;- bit 7  - reserved
        lesb6:1,                ;- bit 6  - reserved
        lesb5:1,                ;- bit 5  - reserved
        lesb4:1,                ;- bit 4  - reserved
```

```
        lesb3:1,                ;- bit 3  - reserved
        lesb2:1,                ;- bit 2  - reserved
        lesempty:1,             ;- bit 1  - empty
        lesfull:1               ;- bit 0  - queue full
LES_FLAGS ENDS

;*  Fields within the Local Enqueue Control Area (LECA) identify
;*  the location of the pipe, indicate the status of the pipe, and
;*  provide state information identifying the starting and ending
;*  offsets of control elements within the pipe.

LECA    STRUC                   ;Local Enqueue Control Area
        base    DD  ?           ;- Pipe Address
        we      DW  ?           ;- offset to wrap element
        status  LES_FLAGS <>    ;- enqueue status
        sf      DW  ?           ;- start of free space in pipe
        ef      DW  ?           ;- end of free space in pipe
        end     DW  ?           ;- offset to end (of queue space)
        top     DW  ?           ;- offset to top (wrap point)
LECA    ENDS

;*  Flags within Surrogate Enqueue Status Word (SES_WRD) are used
;*  by Enqueue Logic to convey state of the pipe to Dequeue Logic
;*  at the other end of the pipe.

SES_FLAGS RECORD                ;Surrogate Enqueue Status Word Flags
        sesb15:1,               ;- bit 15 - reserved
        sesb14:1,               ;- bit 14 - reserved
        sesb13:1,               ;- bit 13 - reserved
        sesb12:1,               ;- bit 12 - reserved
        sesb11:1,               ;- bit 11 - reserved
        sesb10:1,               ;- bit 10 - reserved
        sesb9:1,                ;- bit 9  - reserved
        seswrap:1,              ;- bit 8  - wrap of pipe occurred
        sesb7:1,                ;- bit 7  - reserved
        sesb6:1,                ;- bit 6  - reserved
        sesb5:1,                ;- bit 5  - reserved
        sesb4:1,                ;- bit 4  - reserved
        sesb3:1,                ;- bit 3  - reserved
        sesb2:1,                ;- bit 2  - reserved
        sesb1:1,                ;- bit 1  - reserved
        sesfull:1               ;- bit 0  - pipe is full
SES_FLAGS ENDS

;*  Fields within the Surrogate Enqueue Control Area (SECA)
;*  are used by the Enqueue Logic to provide status and state
;*  information to the Dequeue Logic at the other the pipe.

SECA    STRUCT                  ;Surrogate Enqueue Control Area
        status  SES_FLAGS <>    ;- surrogate enqueue status
        ssf     DW  ?           ;- offset to start of free space
SECA    ENDS

;*  Flags within Local Dequeue Status Word (LDS_WRD) are used by
;*  the Dequeue Logic to maintain local state information at the
;*  dequeue end of pipe.

LDS_FLAGS RECORD                ;Local Dequeue Status Flags
```

```
        ldsb15:1,               ;- bit 15 - reserved
        ldsb14:1,               ;- bit 14 - reserved
        ldsb13:1,               ;- bit 13 - reserved
        ldsb12:1,               ;- bit 12 - reserved
        ldsdequeued:1,          ;- bit 11 - element removed
        ldsprempt:1,            ;- bit 10 - stop dequeue operation
        ldsb9:1,                ;- bit 9  - reserved
        ldswrap:1,              ;- bit 8  - wrap element toggle
        ldsb7:1,                ;- bit 7  - reserved
        ldsb6:1,                ;- bit 6  - reserved
        ldsb5:1,                ;- bit 5  - reserved
        ldsb4:1,                ;- bit 4  - reserved
        ldsb3:1,                ;- bit 3  - reserved
        ldsb2:1,                ;- bit 2  - reserved
        ldsempty:1,             ;- bit 1  - pipe is empty
        ldsfull:1               ;- bit 0  - full
LDS_FLAGS ENDS

;*  Fields within Local Dequeue Control Area (LDCA) identify
;*  the location of the pipe, indicate the status of the pipe, and
;*  provide state information identifying the starting and ending
;*  offsets of control elements within the pipe.

LDCA    STRUC                   ;Local Dequeue Control Area
        base    DD  ?           ;- Pipe Address
        we      DW  ?           ;- offset to wrap element
        ds      LDS_FLAGS <>    ;- Dequeue status
        se      DW  ?           ;- start of elements in pipe
        ee      DW  ?           ;- end of elements in pipe
        end     DW  ?           ;- offset to end (of queue space)
        top     DW  ?           ;- offset to top (wrap point)
LDCA    ends

;*  Flags within Surrogate Dequeue Status Word (SDS_WRD) are used
;*  by the Dequeue Logic to convey the state of the pipe to the
;*  Enqueue Logic at the other end of the pipe.

SDS_FLAGS RECORD                ;Surrogate Dequeue Status Flags
        sdsb15:1,               ;- bit 15 - reserved
        sdsb14:1,               ;- bit 14 - reserved
        sdsb13:1,               ;- bit 13 - reserved
        sdsb12:1,               ;- bit 12 - reserved
        sdsb11:1,               ;- bit 11 - reserved
        sdsb10:1,               ;- bit 10 - reserved
        sdsb9:1,                ;- bit 9  - reserved
        sdswrap:1,              ;- bit 8  - wrap of pipe occurred
        sdsb7:1,                ;- bit 7  - reserved
        sdsb6:1,                ;- bit 6  - reserved
        sdsb5:1,                ;- bit 5  - reserved
        sdsb4:1,                ;- bit 4  - reserved
        sdsb3:1,                ;- bit 3  - reserved
        sdsb2:1,                ;- bit 2  - reserved
        sdsempty:1,             ;- bit 1  - pipe is empty
        sdsb0:1                 ;- bit 0  - reserved
SDS_FLAGS ENDS

SDCA    STRUC                   ;Surrogate Dequeue Control Area
        sds     SDS_FLAGS <>    ;- surrogate dequeue status
```

```
        sse     DW  ?           ;- offset to start of elements
SDCA    ENDS

;*  Fields within the Signalling Area (SIGNAL) are used to identify
;*  the source as well as the reason for signalling from one bus unit
;*  to another.

SCA     STRUC                   ;Signalling Control Area
        nque    DB  ?           ;- enqueue state change
        mgmt    DB  ?           ;- management state change
        byte2   DB  ?           ;- reserved
        dque    DB  ?           ;- dequeue state change
SCA     ENDS
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                           *
;* INCLUDE FILE: ELEMENT.INC                                                 *
;*                                                                           *
;*    COPYRIGHT: (C) Copyright IBM Corporation 1989, 1990.  All              *
;*               rights reserved.                                            *
;*                                                                           *
;*    REV LEVEL: 1.1                                                         *
;*                                                                           *
;*  Description: Contains the data structure definitions for the             *
;*               Move Mode control elements.                                 *
;*                                                                           *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;*        ----- Format Identifier field values -----

FID_MM  equ     0               ;SCB Move mode format

;*        ----- Control Element Type field Element IDs -----

REQ_EL  equ     0               ;request
REP_EL  equ     1               ;reply
EVT_EL  equ     2               ;event
ERR_EL  equ     3               ;error

ERR_EL_LEN equ  20              ;length of error element
DEL_EL_LEN equ  6               ;length of delivery wrap element

;*       ----- Control Element Type field Function Codes -----

                                ;0 and 1 are reserved
FC_INIT    equ  2               ;initialize
                                ;3 is reserved
FC_READ    equ  4               ;read
FC_READL   equ  5               ;read list
FC_READI   equ  6               ;read immediate
FC_WRITE   equ  7               ;write
FC_WRITEL  equ  8               ;write list
FC_WRITEI  equ  9               ;write immediate
FC_EXL     equ 10               ;execute list
FC_MARK    equ 11               ;mark
FC_CANCEL  equ 12               ;cancel
FC_PARAM   equ 13               ;parameters
FC_RDCFG   equ 14               ;read configuration
FC_DIAG    equ 15               ;run diagnostics
                                ;16 is reserved
FC_RESUME  equ 17               ;resume (event)
FC_NOTIFY  equ 18               ;notification (event)
FC_INFORM  equ 19               ;user status information (event)
                                ;20 thru 62 are reserved
FC_WRAP    equ 63               ;wrap of queue (internal)

;        ----- Control Element Value field Error Codes -----

NO_RECV_PENDING equ 1


/*
```

```
                +--------------------------------------+
                |           DATA STRUCTURES            |
                +--------------------------------------+
*/
CIND_FLD RECORD                 ;common indicators field
        id:2,                   ;element id (bits 15-14)
                                ;- (00) request
                                ;- (01) reply
                                ;- (10) event
                                ;- (11) error
        suppress:1,             ;suppress reply flag (bit 13)
        chain:2,                ;chaining flags (bits 12-11)
                                ;- (00) no chaining
                                ;- (01) start of chain
                                ;- (11) middle of chain
                                ;- (10) end of chain
        indirect:1,             ;- bit 10 indirect flag
        notify:1,               ;- bit 9 notification flag
        wait:1,                 ;- bit 8 wait flag
        expedite:1,             ;- bit 7 expedited flag
        fc:7                    ;function codes (bits 6-0)
CIND_FLD ENDS

ADDR_FLD STRUC                  ;Source/Destination Field
        eid     DB  ?           ;- entity identifier
        uid     DB  ?           ;- unit identifier
ADDR_FLD ENDS

ELEMENT STRUC                   ;Element descriptor
        fid     DW  ?           ;- format identifier
        len     DW  ?           ;- length field
        type    CIND_FLD <>     ;- common indicators field
        res1    DW  ?           ;- reserved for future use
        dest    ADDR_FLD <>     ;- destination field
        srce    ADDR_FLD <>     ;- source field
        cor_id  DD  ?           ;- correlation field
        val_fld DD  ?           ;- value field
ELEMENT ENDS
```

Index

A

address space 2-8, 2-9, 2-11
area 2-12
attention port 1-16

B

busy port 1-18

C

cancel 1-54
chaining 4-9
chaining bit 1-36
command busy/status port 1-18
configuration 5-1
configuration record 5-4
control area 1-21, 1-24, 1-25, 1-28, 1-29, 2-13
control element
  basic structure 1-32
  cancel 1-54
  common indicators 1-34
  correlation field 1-38
  destination field 1-37
  diagnose 1-60
  entity-to-entity field 1-38
  error 1-35
    cancel 1-56
    diagnose 1-61
    execute list 1-54
    initialize 1-41
    read 1-43
    read configuration 1-60
    read immediate 1-46
    read list 1-44
    reset 1-58
    write 1-48
    write immediate 1-52
    write list 1-50
control element *(continued)*
  event 1-35, 1-62
    inform 1-63
    notification 1-63
    resume 1-62
    wrap 1-64
  execute list 1-52
  function codes 1-39
  general structure 1-33
  inform 1-63
  initialize 1-40
  mark 1-54
  notification 1-63
  read 1-42
  read configuration 1-58
  read immediate 1-45
  read list 1-43
  reply 1-35, 1-40
    cancel 1-55
    diagnose 1-61
    execute list 1-53
    initialize 1-41
    read 1-42
    read configuration 1-59
    read immediate 1-46
    read list 1-44
    reset 1-58
    write 1-48
    write immediate 1-51
    write list 1-49
  request 1-35, 1-40
    cancel 1-54
    diagnose 1-60
    execute list 1-52
    initialize 1-40
    mark 1-54
    read 1-42
    read configuration 1-58
    read immediate 1-45
    read list 1-43
    reset 1-57

© Copyright IBM Corp. 1991.

control element *(continued)*
request *(continued)*
write 1-47
write immediate 1-50
write list 1-49
reset 1-57
resume 1-62
source field 1-37
wrap 1-64
write 1-47
write immediate 1-50
write list 1-49
control element chaining 4-9
control element delivery 1-11
control element enqueue 3-6
control elements 1-40
correlation field 1-38

D

data chaining 4-9
data delivery 2-13
delivery 1-1
delivery flow 1-11
delivery level 3-1
delivery level protocols 3-8
delivery level services 3-6
delivery pipe 1-19
delivery structure 1-1
dequeue control area 1-25, 1-28
dequeue protocol 3-19
dequeue service 3-7
dequeue status field 1-26
dequeued flag 1-26
design considerations 5-1
destination field 1-37
diagnose 1-60
direct data chaining 4-10

E

element identifier 1-35
empty bit 1-23, 1-27
enable bus master bit 1-17
enable interrupt bit 1-18
end of elements field 1-27
end of free space field 1-23
enqueue 3-14
enqueue control area 1-21, 1-24
enqueue field 1-31
enqueue initialization 3-14
enqueue protocol 3-11
enqueue pseudo code 3-14
enqueue status field 1-22
entity to entity field 1-38
error 1-35
event 1-35
exception handling 5-13
execute list 1-52
expedite bit 1-37

F

feature adapter 2-3
from address 2-11, 2-12
full bit 1-27
full flag 1-23
function code subfield 1-37
function codes 1-39

H

hardware support 2-3

I

indirect bit 1-36
indirect data chaining 4-10
inform 1-63
initialization 3-14, 5-3
initialize 1-40
interface 2-14
interrupt valid bit 1-18
I/O address space 2-9

L

length 2-11, 2-12
length field 1-33
local dequeue control area 1-25
local enqueue control area 1-21

M

management 5-14
mark 1-54
memory address space 1-19, 2-8
move mode 1-1
  chaining 4-9
  characteristics 1-4
  configuration 5-1
  configuration record 5-4
  considerations 5-1
  control areas 2-13
  control element chaining 4-9
  control element delivery 1-11
  control element enqueue 3-6
  data chaining 4-9
  data delivery 2-13
  delivery flow 1-11
  delivery level 3-1
    control element enqueue 3-6
    dequeue protocol 3-19
    dequeue service 3-7
    enqueue 3-14
    enqueue initialization 3-14
    enqueue protocol 3-11
    enqueue pseudo code 3-14
    protocols 3-8
    receive interface 3-5
    receive signal 3-14, 3-21
    send interface 3-5
    signalling 3-5
    single control element enqueue 3-11
  delivery level protocols 3-8
    dequeue 3-19
    enqueue 3-11, 3-14
    enqueue initialization 3-14
    enqueue pseudo code 3-14
move mode *(continued)*
  delivery level protocols *(continued)*
    receive signal 3-14, 3-21
    single control element enqueue 3-11
  delivery level services 3-6
    control element enqueue 3-6
    dequeue 3-7
  dequeue protocol 3-19
  dequeue service 3-7
  direct data chaining 4-10
  enqueue 3-14
  enqueue initialization 3-14
  enqueue protocol 3-11
  enqueue pseudo code 3-14
  entity level
    chaining 4-9
    control element chaining 4-9
    data chaining 4-9
    direct data chaining 4-10
    indirect data chaining 4-10
    notification and wait 4-11
    protocols 4-9
    services 4-8
  entity level protocols
    chaining 4-9
    control element chaining 4-9
    data chaining 4-9
    direct data chaining 4-10
    indirect data chaining 4-10
    notification and wait 4-11
  entity level services
  exception handling 5-13
  feature adapter 2-3
  feature adapter to feature adapter 2-16
  feature adapter to system unit 2-14
  hardware 2-3
  indirect data chaining 4-10
  initialization 5-3
  I/O address space 2-9
  management 5-14

move mode *(continued)*
memory address space 2-8
multiple control element
enqueue 3-7
notification and wait 4-11
peer level 5-3
peer level initialization 5-10
peer unit ID 5-2
physical interface 2-14
physical level 2-1
control areas 2-13
data delivery 2-13
feature adapter 2-3
feature adapter to feature
adapter 2-16
hardware 2-3
interface 2-14
I/O address space 2-9
memory address space 2-8
protocols 2-13
pull 2-4, 2-14
push 2-4, 2-14
services 2-10
signalling 2-5, 2-15
structure 2-1
support logic 2-3
system 2-3
physical level protocols 2-13
feature adapter to system
unit 2-14
physical interface 2-14
pull 2-14
push 2-14
signalling 2-15
physical level services 2-10
data delivery 2-13
pull 2-10
push 2-10
signal 2-12
processing level 4-1
processing level operation 4-2
processing level protocols 4-9
processing level services 4-8
pull 2-4, 2-10, 2-14
move mode *(continued)*
push 2-4, 2-10, 2-14
receive interface 3-5
receive signal 3-14, 3-21
resource definitions 5-2
send interface 3-5
signal 2-12
signalling 2-5, 2-15, 3-5, 5-3
single control element
enqueue 3-11
structure 2-1, 3-1
support logic 2-3
system 2-3
system level configuration 5-2
system level initialization 5-4
unit ID 5-2
unit level configuration 5-1
unit level initialization 5-4
unit type 5-2
multiple control element
enqueue 3-7

N

notification 1-63
notification and wait 4-11
notification bit 1-36

O

offset to end field 1-23, 1-28
offset to top field 1-23, 1-27
operation pending bit 1-18
operational characteristics 1-4
overview, move mode 1-1

P

peer level 5-3
peer level initialization 5-10
peer to peer 1-7
peer unit ID 5-2
physical interface 1-15

physical level 2-1
physical level protocols 2-13
physical level services 2-10
pipe 1-19
pipe address field 1-22, 1-26
pre-empt flag 1-26
processing level 4-1
processing level operation 4-2
processing level protocols 4-9
processing level services 4-8
protocol 2-14, 2-16, 3-11
protocols 2-13, 4-9
pull 2-4, 2-10, 2-14
push 2-4, 2-10, 2-14

Q

queued flag 1-22
queueing 1-9

R

read 1-42
read configuration 1-58
read immediate 1-45
read list 1-43
receive interface 3-5
receive signal 3-14, 3-21
reply 1-35
request 1-35
reset 1-57
reset bit 1-17
resource definitions 5-2
resume 1-62
return code 2-11, 2-12

S

send interface 3-5
signal 2-12
signal code 1-17
signalling 1-29, 2-5, 2-15, 3-5, 5-3
signalling control area 1-29
single control element enqueue 3-11
source field 1-37
start of elements field 1-27
start of free space field 1-23
status port 1-18
structure 2-1, 3-1
subsystem control port 1-17
support logic 2-3
suppress reply bit 1-35
surrogate dequeue control area 1-28
surrogate dequeue status 1-29
surrogate enqueue control area 1-24
surrogate enqueue status 1-25
surrogate start of elements 1-28
surrogate start of free 1-24
system 2-3
system level configuration 5-2
system level initialization 5-4

T

to address 2-11, 2-12
type field 1-33

U

unit ID 2-10, 2-11, 2-12, 5-2
unit level configuration 5-1
unit level initialization 5-4
unit type 5-2

V

value 2-12
value field 1-33

W

wait bit 1-36
wrap 1-64

wrap bit 1-22, 1-27
wrap element offset 1-23
wrap element offset field 1-27
write 1-47
write immediate 1-50
write list 1-49