



®

# Macintosh Technology in the Common Hardware Reference Platform

Apple Computer, Inc.

## **ONLINE VERSION**

You can obtain paper copies of this book from computer bookstores or by writing Morgan Kaufmann Publishers, Inc., 340 Pine Street, Sixth Floor, San Francisco, CA 94104. Reference ISBN 1-55860-393-X.

## NOTICES

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law. In such countries, the minimum country warranties will apply.

APPLE COMPUTER, INC. PROVIDES THIS DOCUMENT (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) “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. THE DISCLAIMER OF WARRANTY APPLIES NOT ONLY TO THE DOCUMENT (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) BUT ALSO TO ANY COMBINATIONS, INCORPORATIONS, OR OTHER USES OF THE DOCUMENT (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) UPON WHICH A CLAIM COULD BE BASED.

Some states do not allow disclaimers of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

These materials 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 document. Apple Computer, Inc. may make improvements and/or changes in the product(s) and/or the program(s) described in, or accompanying, this document at any time.

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

Requests for technical information about products described herein should be directed to Apple Computer, Inc. Refer to “Developer Services and Programs” on page 205 for a description of information available and telephone numbers.

© Copyright 1995 Apple Computer, Inc. All rights reserved.

## LICENSE INFORMATION

To the extent that Apple Computer, Inc. owns licensable copyrights in *Macintosh Technology in the Common Hardware Reference Platform* (including accompanying source code samples), Apple Computer, Inc. grants you a copyright license to copy and distribute this document (including accompanying source code samples) in any form, without payment to Apple Computer, Inc., for the purpose of developing code or equipment that conform to this document (including accompanying source code samples) and for the purpose of using, reproducing, marketing, and distributing such code or equipment. This authorization applies to the content of this specification only and not to any materials referenced herein.

In consideration you agree to include for each reproduction of any portion of these documents (including accompanying source code samples) the copyright notice as displayed below or to include for any derivative works based thereon that are marketed or distributed to others a copyright notice as follows:

“© Copyright (year) (your company name). All rights reserved.”

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization. If you fail to comply with the above terms, your authorization terminates.

Apple Computer, Inc. and others may have patents or pending patent applications or other intellectual property rights covering the subject matter described herein. This document neither grants nor implies a license or immunity under any Apple Computer, Inc. or third party patents, patent applications or other intellectual property rights other than as expressly provided in the foregoing copyright license. Apple Computer, Inc. assume no responsibility for any infringement of third party rights resulting from your use of the subject matter disclosed in, or from the manufacturing, use, lease or sale of products described in, this document.

Licenses under utility patents of Apple Computer, Inc. that are necessary to implement the specifications set forth in this document are available on reasonable and non-discriminatory terms. Apple Computer, Inc. does not grant licenses to its appearance design patents. Direct your licensing inquiries in writing to:

Mac OS Licensing Department  
Apple Computer, Inc.  
1 Infinite Loop, MS 305-1DS  
Cupertino, CA 95014

## **TRADEMARKS**

Apple, the Apple logo, APDA, AppleLink, AppleTalk, GeoPort, LaserWriter, LocalTalk, Mac, Macintosh, Power Macintosh, PowerTalk, and QuickTime are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Apple Desktop Bus, eWorld, and QuickDraw are trademarks of Apple Computer, Inc.

CHRP™ is a trademark of Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc.

NuBus™ is a trademark of Texas Instruments.

PowerPC™ is a trademark of International Business Machines Corporation, used under license therefrom.



# Contents

Figures .....	xvii
Tables .....	xix
About This Book .....	xxiii
Who Should Use This Book .....	xxiii
Organization of Material .....	xxiv
Suggested Reading .....	xxv
Conventions Used in This Book .....	xxv
Acronyms and Abbreviations .....	xxvi
Acknowledgments .....	xxix
Chapter 1 Introduction .....	1
1.1 Overview .....	1
1.2 Macintosh Technology Covered in This Book .....	1
1.3 Historical Background .....	2
1.3.1 First-Generation Power Macintosh Computers .....	2
1.3.2 Second-Generation Power Macintosh Computers .....	3
1.4 CHRP Specification Power Macintosh Computers .....	3
Chapter 2 The Mac I/O Chip .....	7
2.1 Overview .....	7
2.2 Internal Architecture .....	7
2.3 Features .....	9
2.4 Functional Units .....	9
2.4.1 PCI Interface .....	9
2.4.2 DBDMA Controller .....	10
2.4.3 SCSI Controller .....	10
2.4.4 SCC Controller .....	10

2.4.5	VIA Functions . . . . .	11
2.4.6	ADB Controller . . . . .	11
2.4.7	Open PIC Interrupt Controller . . . . .	11
2.5	Software Interface . . . . .	12
2.5.1	Register Accesses . . . . .	12
2.5.2	DBDMA Operation . . . . .	12
2.5.3	Interrupts. . . . .	12
2.6	Memory Map and Register Addressing . . . . .	13
2.6.1	DBDMA Channel Number Assignments . . . . .	15
2.6.2	Device Number Assignments . . . . .	15
2.6.3	Register Map . . . . .	16
2.7	General Functions . . . . .	25
2.7.1	Cache Indicator. . . . .	25
2.7.2	CPU IDs . . . . .	26
2.7.3	Feature Controls. . . . .	26
2.7.4	Sleep Mode. . . . .	27
2.8	PCI Bus Interface . . . . .	28
2.8.1	General PCI Bus Transactions . . . . .	28
2.8.1.1	Address Transactions . . . . .	28
2.8.1.2	Data Transactions. . . . .	30
2.8.1.3	Slave Address Decoding. . . . .	30
2.8.1.4	Bus Control and Turnaround. . . . .	31
2.8.1.5	Byte Enables. . . . .	32
2.8.1.6	Bus Parity . . . . .	32
2.8.1.7	Transaction Termination . . . . .	33
2.8.2	Read Transactions . . . . .	34
2.8.3	Write Transactions . . . . .	36
2.8.4	Configuration Transactions. . . . .	36
2.8.5	Configuration Registers . . . . .	37
2.8.5.1	Device Identification . . . . .	38
2.8.5.2	Device Control . . . . .	38
2.8.5.3	Device Status . . . . .	39
2.8.5.4	Miscellaneous Functions. . . . .	40
2.8.6	Base Registers . . . . .	40
2.8.7	Interface States. . . . .	41
2.8.8	PCI Arbitration . . . . .	41
2.9	DBDMA Operation . . . . .	42
2.9.1	DBDMA Channel Registers . . . . .	42
2.9.1.1	ChannelControl. . . . .	43
2.9.1.2	ChannelStatus . . . . .	43
2.9.1.3	CommandPtr. . . . .	45
2.9.1.4	InterruptSelect Register . . . . .	46
2.9.1.5	BranchSelect Register . . . . .	46
2.9.1.6	WaitSelect Register . . . . .	47
2.9.1.7	Summary of Channel Registers . . . . .	47

---

2.9.2	Commands	48
2.9.2.1	Command Formats	48
2.9.2.2	INPUT and OUTPUT Commands	53
2.9.2.3	STORE_QUAD Command	54
2.9.2.4	LOAD_QUAD Command	55
2.9.2.5	NOP Command	55
2.9.2.6	STOP Command	56
2.10	Interrupt Processing	57
2.10.1	Interrupt Assignments	59
2.10.2	Disabling the Open PIC Cell	60
2.11	I/O Modules	61
2.11.1	VIA Support	61
2.11.2	ADB Support	62
2.11.3	SCC Support	63
2.11.3.1	SCC Registers	63
2.11.3.2	LocalTalk Support	65
2.11.3.3	Channel Status Register Usage	67
2.11.3.4	Conditional Interrupt, Branch, and Wait Generation	67
2.11.3.5	Channel Program Example	68
2.11.4	SCSI Support	69
2.11.4.1	Channel Status Register Usage	69
2.11.4.2	Conditional Interrupt, Branch, and Wait Generation	70
2.11.4.3	SCSI Channel Commands	70
2.12	Testing	71
2.13	Physical Package	72
2.13.1	Signal Lines	72
2.13.2	Pin Assignments	77
<b>Chapter 3 Apple Desktop Bus Controller</b>		<b>81</b>
3.1	Overview	81
3.2	Electrical Interface	81
3.3	Architecture and Signals	82
3.4	Communication With the System	83
3.4.1	Command/Data Register File	83
3.4.2	CDRF Arbitration	84
3.4.3	Data Transfers From the System to the ADB	85
3.4.4	Data Transfers From the ADB to the System	86
3.5	Error Handling	86
3.5.1	Data Lost Error	87
3.5.2	No Response Error	87
3.5.3	SRQ Autopolling Error	87
3.5.4	Inactive Device Response Error	87
3.6	Autopolling, SRQ Autopolling, and Error Recovery	88



3.7	ADB Controller Special Functions	89
3.7.1	Keyboard-Invoked Reset	90
3.7.2	Keyboard-Invoked NMI	90
3.7.3	Power Message Handling	90
3.8	Bit Cell Determination and Timing	90
3.8.1	Bit Cell Sampling Algorithm	91
3.8.1.1	Bit Determination Algorithm	92
3.8.1.2	Bit Determination Timeout	92
3.8.2	Miscellaneous ADB Timing Cautions	92
3.9	Controller Hard Reset	93
3.10	System Interface to the ADB	93
3.10.1	Status and Control Registers	94
3.10.1.1	Command/Data Register File (CDRF)	97
3.10.1.2	Transfer Access Request (TAR)	97
3.10.1.3	Transfer Access Grant (TAG)	98
3.10.1.4	TAG Interrupt Source Enable (TAG_IS_EN)	98
3.10.1.5	Data From Bus (DFB)	98
3.10.1.6	DFB Interrupt Source Enable (DFB_IS_EN)	98
3.10.1.7	Data to Bus (DTB)	99
3.10.1.8	Data Transfer Process	99
3.10.1.9	Interrupt Request (IRQ)	100
3.10.1.10	Data Lost Error (DLE)	100
3.10.1.11	No Response Error (NRE)	101
3.10.1.12	Autopoll Data (APD)	101
3.10.1.13	How Many Bytes (HMB[3:0])	101
3.10.1.14	Command Response Expected (CRE)	102
3.10.1.15	Autopoll Enable (APE)	102
3.10.1.16	Active Device Address Registers	102
3.10.1.17	ADB Reset (ADB_RST)	103
3.10.1.18	Monostable Reset Enable (MR_EN)	103
3.10.1.19	ADB Controller Test Register (TST[7:0])	103
3.10.2	Timing Value Registers	103
Chapter 4	SCSI Support	105
4.1	Overview	105
4.2	Architecture	106
4.3	Electrical Interface	108
4.4	Registers	109
4.4.1	Transfer Count Registers	109
4.4.2	Bus FIFO Register	110
4.4.3	Sequence Register	111
4.4.3.1	DMA Bit	111
4.4.3.2	TMode Bit	111
4.4.3.3	Atn Bit	111
4.4.3.4	ActNeg Bit	111
4.4.3.5	Seq Bits	112

4.4.4	Bus Status Registers	115
4.4.5	Bus FIFO Count Register	115
4.4.6	Exception Register	115
4.4.6.1	SelWAtn Bit	116
4.4.6.2	Selected Bit	116
4.4.6.3	Reselected Bit	116
4.4.6.4	ArbLost Bit	116
4.4.6.5	PhaseMM Bit	116
4.4.6.6	SelTO Bit	116
4.4.7	Error Register	117
4.4.7.1	UnExpDisc Bit	117
4.4.7.2	SCSIRst Bit	117
4.4.7.3	SeqErr Bit	117
4.4.7.4	ParityErr Bits	117
4.4.8	Interrupt Mask Register	118
4.4.9	Interrupt Register	118
4.4.9.1	Error Bit	118
4.4.9.2	Exception Bit	118
4.4.9.3	CmdDone Bit	119
4.4.10	SourceID Register	119
4.4.11	DestinationID Register	119
4.4.12	SyncParms Register	119
4.4.12.1	SyncOffset Field	119
4.4.12.2	SyncPeriod Field	120
4.4.12.3	Synchronous Data Transfers	120
4.4.13	MESH ID Register	121
4.4.14	Selection TimeOut Register	121
4.5	Signal Interface	121
4.5.1	Control and Status Signals	121
4.5.2	SCSI Bus Signals	123
4.5.3	Operating Phases	124
4.5.3.1	Phase Types	125
4.5.3.2	Phase Expecting	125
4.6	Timing	126
4.6.1	Clock Input	126
4.6.2	Register Writes	127
4.6.3	Register Reads	128
4.6.4	DMA Writes	129
4.6.5	DMA Reads	130
4.6.6	Interrupts	131
4.6.7	Resets	131
4.6.8	Host Asynchronous Transmission	132
4.6.9	Host Asynchronous Reception	132
4.6.10	Target Asynchronous Transmission	133
4.6.11	Target Asynchronous Reception	133

4.7	Sample Phase Timing	134
4.7.1	Arbitration Phase	134
4.7.2	Selection Phase	134
4.8	Implementation Notes	135
4.8.1	SCSI Bus Connector	135
4.8.1.1	Pin Assignments	135
4.8.1.2	Bus Termination	136
4.8.2	Request and Acknowledge Signal Circuitry	137
4.8.3	Comparison of MESH and 53C9x	138
Chapter 5 Descriptor-Based DMA		139
5.1	Overview	139
5.1.1	Design Objectives	140
5.1.2	Design Strategies	140
5.1.3	Contents of This Chapter	141
5.2	Conventions	142
5.2.1	Terminology	142
5.2.2	Bit and Byte Ordering	142
5.2.2.1	Big-Endian Variant	143
5.2.2.2	Little-Endian Variant	143
5.2.3	Register, Field, and Constant Names	144
5.2.4	Reserved Values	144
5.3	Summary of DBDMA Operations	145
5.3.1	Multiplexed Channels	146
5.3.2	Command List Structure	147
5.4	Design Model	148
5.4.1	Controller Components	148
5.4.2	System Bus Errors	150
5.4.3	Device Status	150
5.4.4	Conditional Actions	151
5.5	Controller Registers	151
5.5.1	Register Organization	151
5.5.2	ChannelControl Register	153
5.5.2.1	ChannelControl.mask Field	154
5.5.2.2	ChannelControl.data Field	154
5.5.3	ChannelStatus Register	154
5.5.3.1	ChannelStatus.run Bit	154
5.5.3.2	ChannelStatus.pause Bit	155
5.5.3.3	ChannelStatus.flush Bit	155
5.5.3.4	ChannelStatus.wake Bit	155
5.5.3.5	ChannelStatus.dead Bit	155
5.5.3.6	ChannelStatus.active Bit	156
5.5.3.7	ChannelStatus.bt Bit	156
5.5.3.8	ChannelStatus.s7..s0 Bits	156
5.5.4	CommandPtr Registers	157
5.5.5	InterruptSelect Register	157

---

5.5.6	BranchSelect Register	158
5.5.7	WaitSelect Register	158
5.5.8	Data2Ptr Registers	159
5.5.9	TransferMode Register	159
5.5.9.1	TransferMode.pam and TransferMode.sam Fields	160
5.5.9.2	TransferMode.pbs and TransferMode.sbs Fields	160
5.5.9.3	TransferMode.pc and TransferMode.sc Fields	161
5.5.10	AddressHi Register	161
5.5.11	BranchAddressHi Register	162
5.6	Commands	162
5.6.1	Command Formats	162
5.6.1.1	Command.cmd Field	163
5.6.1.2	Command.key Field	163
5.6.1.3	Command.i Field	165
5.6.1.4	Command.b Field	166
5.6.1.5	Command.w Field	167
5.6.1.6	Command.reqCount Field	167
5.6.1.7	Command.address Field	167
5.6.1.8	Command.cmdDep Field	168
5.6.1.9	Command.xferStatus Field	168
5.6.1.10	Command.resCount Field	169
5.6.2	INPUT and OUTPUT Commands	169
5.6.3	STORE_QUAD Command	170
5.6.4	LOAD_QUAD Command	171
5.6.5	NOP Command	171
5.6.6	STOP Command	172
5.7	Asynchronous Event Packet Formats	173
5.8	Summary of DBDMA Options	173
5.8.1	Hardwired Interrupts	174
5.8.2	Asynchronous Event Channel	174
5.8.3	Optional Registers	174
5.9	Examples	174
5.9.1	Command Queuing	174
5.9.2	Ethernet Reception	176
5.9.3	Full Handshake Flow Control	178
Chapter 6 Macintosh Serial Port		181
6.1	Overview	181
6.2	Connector	182
6.3	Pin Assignments	182
6.4	Serial Communications Controller	183

---

Chapter 7	Macintosh Toolbox ROM	185
7.1	Overview	185
7.2	Physical Mounting	185
7.3	Electrical Connection	185
7.4	Timing	187
Chapter 8	Open Firmware Requirements	189
8.1	Overview	189
8.2	Name Registry	189
8.2.1	Description	189
8.2.2	Constructing the Name Registry	190
8.2.3	Name Registry Properties	191
8.2.3.1	Name Property	192
8.2.3.2	Driver Property	193
8.2.3.3	Assigned Addresses	193
8.2.3.4	ID Properties	193
8.2.3.5	ROM Offset	193
8.2.3.6	Driver-Ref Pointer	193
8.2.3.7	Driver Description	194
8.2.3.8	Apple Address Property	194
8.2.3.9	Apple Interrupts Property	194
8.2.3.10	Apple Slot Name Property	194
8.2.3.11	Graphic Display Properties	195
8.2.3.12	Driver Code Access	195
8.2.4	Translation From the DCT to the Name Registry	195
Chapter 9	Mac OS NVRAM Requirements	197
9.1	Overview	197
9.2	NVRAM Allocation	197
9.3	Storing Name Registry Data in NVRAM	198
Chapter 10	Macintosh Power Controls	199
10.1	Overview	199
10.2	Determining Device Power Levels	199
10.3	Setting Device Power Levels	200
Appendix A	List of Requirements	201
A.1	Apple Desktop Bus Controller	201
A.2	SCSI Support	202
A.3	Descriptor-Based DMA	202
A.4	Macintosh Serial Port	202
A.5	Macintosh Toolbox ROM	203
A.6	Open Firmware	203
A.7	NVRAM	203
A.8	Power Controls	203

---

Appendix B Developer Services and Programs .....	205
B.1 Overview .....	205
B.2 How to Get Information .....	205
B.3 Developer Resources .....	206
B.3.1 Orientation Kit .....	206
B.3.2 Developer University .....	206
B.3.3 Worldwide Developers Conference .....	206
B.3.4 Hardware Purchasing Privileges .....	207
B.3.5 Technology Seeding .....	207
B.3.6 Market Research .....	207
B.3.7 Technical Journal .....	207
B.3.8 Developer CD .....	207
B.3.9 Software Development Kit .....	207
B.3.10 Compatibility Testing Lab .....	208
Bibliography .....	209
Apple / IBM / Motorola .....	209
Apple Computer, Inc. ....	209
IBM. ....	211
American National Standards Institute. ....	212
FirmWorks .....	212
Institute of Electrical and Electronic Engineers .....	212
PCI Special Interest Group .....	213
SunSoft Press .....	213
Glossary .....	215
Index .....	221



# Figures

1. MIO block diagram . . . . .	8
2. MIO memory map . . . . .	14
3. PCI bus parity generation . . . . .	32
4. PCI bus target-initiated termination . . . . .	33
5. PCI bus read transactions . . . . .	35
6. PCI bus slave write transactions . . . . .	36
7. Configuration transaction . . . . .	37
8. Command entry format . . . . .	48
9. Format of INPUT and OUTPUT commands . . . . .	53
10. STORE_QUAD command fields . . . . .	54
11. LOAD_QUAD command fields . . . . .	55
12. NOP command fields . . . . .	56
13. STOP command fields . . . . .	56
14. 8259 master interrupt configuration . . . . .	57
15. Open PIC master interrupt configuration . . . . .	58
16. Initial Recovery Count register . . . . .	63
17. StartA and StartB registers . . . . .	65
18. DetectAB register . . . . .	66
19. ADB connector . . . . .	82
20. ADB controller block diagram . . . . .	82
21. Bit cell timing diagram . . . . .	91
22. Status/control register bits . . . . .	95
23. MESH controller architecture . . . . .	106
24. Sequence register format . . . . .	111
25. Bus Status0 register format . . . . .	115
26. Bus Status1 register format . . . . .	115
27. Exception register format . . . . .	116
28. Error register format . . . . .	117
29. Interrupt Mask register format . . . . .	118



---

30. Interrupt register format . . . . .	118
31. SyncParms register format . . . . .	119
32. Clock input timing . . . . .	126
33. Register write timing . . . . .	127
34. Register read timing . . . . .	128
35. DMA write timing . . . . .	129
36. DMA read timing . . . . .	130
37. Interrupt timing . . . . .	131
38. Reset timing . . . . .	131
39. Host transmit timing . . . . .	132
40. Host receive timing . . . . .	132
41. Target transmit timing . . . . .	133
42. Target receive timing . . . . .	134
43. Arbitration phase timing . . . . .	134
44. Selection phase timing . . . . .	135
45. Req signal circuitry . . . . .	137
46. Ack signal circuitry . . . . .	137
47. Big-endian byte ordering . . . . .	143
48. Little-endian byte ordering . . . . .	143
49. Reserved field illustration . . . . .	145
50. DBDMA operation . . . . .	145
51. Multiplexed channel types . . . . .	146
52. Command list structure . . . . .	147
53. DBDMA controller model . . . . .	149
54. ChannelControl register format . . . . .	153
55. ChannelStatus register format . . . . .	154
56. InterruptSelect register . . . . .	157
57. TransferMode register . . . . .	159
58. Command entry format . . . . .	162
59. INPUT and OUTPUT command format . . . . .	169
60. STORE_QUAD command format . . . . .	170
61. LOAD_QUAD command format . . . . .	171
62. NOP command format . . . . .	172
63. STOP command format . . . . .	172
64. Event packet format . . . . .	173
65. Flow-controlled device model . . . . .	178
66. Synchronizing flow-controlled channel programs . . . . .	179
67. Macintosh serial port connector . . . . .	182

# Tables

1. Macintosh technology in this book . . . . .	1
2. CHRP-Macintosh feature comparison . . . . .	4
3. Register space definitions. . . . .	13
4. DBDMA channel number assignments. . . . .	15
5. Device number assignments . . . . .	15
6. MIO register map . . . . .	16
7. CachePD register . . . . .	25
8. CPUID register . . . . .	26
9. Feature Control register . . . . .	26
10. PCI bus commands supported by the MIO chip . . . . .	29
11. Device response times . . . . .	29
12. PciHeader registers . . . . .	37
13. PciHeader register values. . . . .	38
14. Device identification bit-field assignments . . . . .	38
15. Device control bit-field assignments . . . . .	39
16. Device status bit-field assignments . . . . .	40
17. PciBase register . . . . .	41
18. ChannelControl register format. . . . .	43
19. ChannelControl register description . . . . .	43
20. ChannelStatus register format . . . . .	44
21. ChannelStatus register description. . . . .	44
22. CommandPtr register format . . . . .	45
23. InterruptSelect register format . . . . .	46
24. Channel register summary . . . . .	47
25. Command.cmd field values . . . . .	48
26. Command.key field effects . . . . .	49
27. Key field usage . . . . .	50
28. Command.i field effects . . . . .	51
29. Command.b field effects . . . . .	51

30. Command.w field effects . . . . .	52
31. Effective count values . . . . .	55
32. MIO Open PIC modes . . . . .	58
33. Interrupt assignments . . . . .	59
34. Interrupt modes . . . . .	60
35. Interrupt assignments with Open PIC disabled . . . . .	60
36. VIA port A usage . . . . .	62
37. Sample IRC setting . . . . .	64
38. SCC transmit channel status bits . . . . .	67
39. SCC receive channel status bits . . . . .	67
40. SCC channel program trace . . . . .	68
41. SCSI channel status bits . . . . .	69
42. Field definitions for SCSI DBDMA commands . . . . .	70
43. Field definitions for SCSI register commands . . . . .	71
44. MIO test pins . . . . .	72
45. MIO chip signal lines . . . . .	72
46. MIO chip pin assignments . . . . .	77
47. ADB controller signal lines . . . . .	83
48. ADB error conditions . . . . .	86
49. Bit cell timing limits . . . . .	91
50. ADB status and control registers . . . . .	94
51. Timing register parameters . . . . .	104
52. SCSI controller interface . . . . .	107
53. SCSI and TTL signal levels . . . . .	108
54. SCSI controller registers . . . . .	109
55. Seq bit encoding . . . . .	112
56. Control and status signals . . . . .	122
57. SCSI bus signals . . . . .	123
58. SCSI bus signal definitions . . . . .	124
59. Information transfer phases . . . . .	125
60. Clock timing parameters . . . . .	126
61. Register write timing parameters . . . . .	127
62. Register read timing parameters . . . . .	128
63. DMA write timing parameters . . . . .	129
64. DMA read timing parameters . . . . .	130
65. Interrupt timing parameters . . . . .	131
66. Reset timing parameters . . . . .	131
67. Host transmit timing parameters . . . . .	132
68. Host receive timing parameters . . . . .	133
69. Target transmit timing parameters . . . . .	133
70. Target receive timing parameters . . . . .	134
71. SCSI connector pin assignments . . . . .	135
72. Name notation examples . . . . .	144
73. Channel registers . . . . .	152
74. DBDMA register summary . . . . .	153
75. TransferMode register bits . . . . .	159

---

76. Address mode encoding . . . . .	160
77. Block size encodings . . . . .	160
78. Command.cmd field values . . . . .	163
79. Effects of Command.key field . . . . .	163
80. Data transfer operations . . . . .	164
81. Effects of Command.i field . . . . .	166
82. Effects of Command.b field . . . . .	166
83. Effects of Command.w field . . . . .	167
84. STORE_QUAD reqCount values . . . . .	170
85. Queued transmission commands . . . . .	175
86. Ethernet receive channel commands . . . . .	176
87. Ethernet receive command sequence . . . . .	177
88. Serial port pin assignments . . . . .	182
89. ROM pin assignments . . . . .	186
90. ROM timing . . . . .	188
91. Name Registry properties . . . . .	191
92. Typical NVRAM space allocations . . . . .	197
93. Driver Gestalt selectors for power information . . . . .	199
94. Power setting controls . . . . .	200



# About This Book

This book presents specifications and standards for the Apple Macintosh technology cited in the PowerPC Microprocessor Common Hardware Reference Platform (CHRP) specification.

The CHRP specification is a joint publication of Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc. It defines a common hardware architecture for computers that use the PowerPC microprocessor. For information about obtaining the CHRP specification, see the bibliography, page 209.

All of the Macintosh technology incorporated in the CHRP specification is currently used in Power Macintosh computers. This book defines that technology so that third-party vendors can design and build computers containing the required Macintosh features. By following this specification, third-party computers running the Macintosh Operating System (Mac OS) can deliver many of the same operating benefits and user experiences as computers manufactured by Apple. For information about Apple Computer's copyright, patent, and intellectual property rights in the technology disclosed here, see the licensing information in the front of this book.

## Who Should Use This Book

This book is written for professional hardware and software engineers who are familiar with computer design. It assumes that the reader already understands the CHRP specification and is seeking detailed technical information about the Macintosh technology cited in it.

Besides publishing this book, Apple offers a range of services and partnerships to assist developers of products compatible with Macintosh computers. This assistance is described in Appendix B.

## Organization of Material

Following is a brief description of the major sections of this book:

- Chapter 1, “Introduction,” beginning on page 1, lists the technology described in this book and surveys the Macintosh environment in which the technology operates.
- Chapter 2, “The Mac I/O Chip,” beginning on page 7, describes an ASIC chip designed by Apple and manufactured by several vendors that implements much of the Macintosh technology that has been incorporated in the CHRP architecture.
- Chapter 3, “Apple Desktop Bus Controller,” beginning on page 81, describes the controller required to implement the Apple Desktop Bus, a low-speed bus for input devices such as keyboards, mouse devices, and tablets.
- Chapter 4, “SCSI Support,” beginning on page 105, describes the Apple-designed controller for the Small Computer System Interface (SCSI) bus used in designs compliant with the CHRP specification.
- Chapter 5, “Descriptor-Based DMA,” beginning on page 139, describes the hardware and software operation of Apple’s descriptor-based direct memory access (DBDMA) technology for I/O data transfers.
- Chapter 6, “Macintosh Serial Port,” beginning on page 181, describes the 85C30-compatible serial port that supports Macintosh peripheral devices such as printers, modems, LocalTalk local area networks, and Apple GeoPort networking hardware.
- Chapter 7, “Macintosh Toolbox ROM,” beginning on page 185, gives hardware details on the Apple read-only memory (ROM) chip that contains system code required to run the Mac OS.
- Chapter 8, “Open Firmware Requirements,” beginning on page 189, describes the Open Firmware code that must be present in a computer that complies with the CHRP specification and operates Mac OS-compatible drivers and expansion cards.
- Chapter 9, “Mac OS NVRAM Requirements,” beginning on page 197, specifies the NVRAM support required to run the Mac OS on a computer that complies with the CHRP specification.
- Chapter 10, “Macintosh Power Controls,” beginning on page 199, describes the power controls that the Mac OS requires in a computer that complies with the CHRP specification.

The following appendixes provide supplementary information:

- Appendix A, “List of Requirements,” beginning on page 201, summarizes in one place all the conformance requirements stated in this book.
- Appendix B, “Developer Services and Programs,” beginning on page 205, describes resources that provide technical and marketing assistance to third-party developers of Macintosh-compatible hardware and software, including manufacturers of computers that comply with the CHRP specification.

At the end of this book are a bibliography, a glossary, and an index.

## Suggested Reading

Familiarity with the following documents is suggested as a valuable foundation for understanding the technology presented in this book. For information about obtaining these documents, see the bibliography.

- *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*, Version 1.0, by Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc. Morgan Kaufmann Publishers, San Francisco, CA, 1995
- *1275-1994 Standard for Boot (Initialization, Configuration) Firmware*, by the Institute for Electrical and Electronic Engineers
- *PCI Local Bus Specification*, Revision 2.0, by the PCI Special Interest Group
- *ANSI X3T9.2 SCSI-3 SCSI Parallel Interface (SPI) Specification*, by the American National Standards Institute

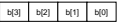
## Conventions Used in This Book

Requirements are clearly defined in the body of this book with notes headed “**Requirement**” in boldface type. Following each heading is text that may point to other text, figures, tables, or referenced documents that specify the requirement.

The referenced material becomes part of the requirements in this book. Users of this book must comply with these requirements to implement Macintosh technology in the Common Hardware Reference Platform.



The following typographical, numbering, and graphic conventions are used in this book:

Element	Description of Use
0xnnnn	0x is a prefix to denote hexadecimal numbers.
0bnnnn	0b is a prefix to denote binary numbers.
nnnnn	Numbers without a prefix are decimal numbers.
0:9	The colon is used to define a range of bits or bytes, including the end values and all those between the limits given.
0xm-0xn	A range of addresses within the book is always inclusive, from m up to and including n.
/Xxx Xxx_L	A slash before a signal name indicates that the signal is active-low; that is, the signal is asserted when low and deasserted when high. Active-low signals may also be indicated by _L after their names.
	When a register or command format is illustrated as a horizontal block, the high-order bit or byte is shown at the left end; the low-order bit or byte is shown at the right end.

## Acronyms and Abbreviations

The following acronyms and abbreviations are used in this book:

Term	Meaning
ADB	Apple Desktop Bus
ANSI	American National Standards Institute
APDA	Apple Programmers and Developers Association
ARBus	Apple RISC Bus
ASCII	American National Standards Code for Information Interchange
ASIC	application-specific integrated circuit
BE	big-endian
BIST	built-in self test
bpp	bits per pixel

Term	Meaning
CDB	command descriptor block
CD-ROM	compact disk read-only memory
CDRF	command/data register file
CHRP	Common Hardware Reference Platform
CISC	complex instruction set computer
CPU	central processing unit
CSR	control and status register (architecture)
CTR	count register
DAC	dual address cycle
DAR	data address register
DBDMA	descriptor-based direct memory access
DCT	device configuration tree
DRAM	dynamic random access memory
DMA	direct memory access
EIA	Electronics Industry Association
EISA	extended industry standard architecture
EPROM	erasable programmable read-only memory
FCode	Forth code
FIFO	first in, first out
GB	gigabytes
GUI	graphical user interface
HB	host bridge
Hz	hertz
IEEE	Institute of Electrical and Electronics Engineers
IRC	initial recovery count
IRQ	interrupt request

Term	Meaning
KB	kilobytes
LE	little-endian
LSB	least significant byte
lsb	least significant bit
MB	megabytes
MHz	megahertz
MESH	Macintosh Enhanced SCSI Hardware
MIO	Mac I/O
MRU	most-recently used
MSB	most significant byte
msb	most significant bit
MSR	machine state register
n.a.	not applicable
NMI	nonmaskable interrupt
NVRAM	nonvolatile random access memory
OEM	original equipment manufacturer
OS	operating system
PC	personal computer
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
PCR	PCI configuration register
PHB	PCI host bridge
PIC	programmable interrupt controller
POST	power-on self-test
RAM	random access memory
RGB	red/green/blue

Term	Meaning
RISC	reduced instruction set computing
ROM	read-only memory
RTAS	run-time abstraction services
SCC	Serial Communications Controller
SCSI	Small Computer System Interface
SDTR	synchronous data transfer request
SIMM	single inline memory module
SMP	symmetric multiprocessor
TB	time base
TBD	to be determined
VESA	Video Electronics Standards Association
VIA	versatile interface adapter

## Acknowledgments

This book was written, edited, and produced by the staff of Apple Developer Press in Cupertino, CA. The following people were directly responsible for its creation: Writer, George Towner; Copyeditor, Wendy Krafft; Illustrator, Sandee Karr; Project Manager, Rolly Reed. Apple would also like to thank Jennifer Mann and the staff of Morgan Kaufmann Publishers for their help with this book's publication.

The information in this book was developed primarily by engineers at Apple Computer, with suggestions, comments, and assistance by engineers at IBM and Motorola. Apple would particularly like to thank the following advisors and contributors: Art Adkins, Stanford Au, Bob Bailey, Mike Bell, Steve Bunch, Kevin Christiansen, Scott Comer, Keith Cox, Garey De Angelis, Mike Eneboe, Jim Gable, Ron Hochsprung, David James, Steve MacKenzie, Paul Resch, Al Scalise, Mark Stubbs, and Dave Tjon.



# Introduction

# 1

## 1.1 Overview

This chapter surveys the Macintosh environment in which the technology described in this book operates. The next section lists the areas of Macintosh technology that are described in later chapters.

**Note:** This chapter is descriptive. It contains no specification requirements.

## 1.2 Macintosh Technology Covered in This Book

Table 1 lists the general areas of Macintosh technology that are relevant to the CHRP specification and cites the sections of this book where that technology is discussed.

Table 1. Macintosh technology in this book

Technology	Page references
Apple Desktop Bus (ADB)	11, 62, 81
SCSI controller (MESH)	10, 69, 105
Descriptor-based direct memory access (DBDMA)	10, 42, 139
Interrupt handling (VIA and Open PIC)	11, 57
Macintosh serial port (SCC)	10, 63, 181

Table 1. Macintosh technology in this book (*continued*)

Technology	Page references
Mac OS Toolbox ROM	185
Open Firmware	189
Nonvolatile RAM (NVRAM) usage	197
Power management	199

Besides discussing the specific areas of technology listed in Table 1, this book describes an ASIC chip designed by Apple that implements much of the Macintosh technology used in the Common Hardware Reference Platform. It is described in Chapter 2, “The Mac I/O Chip,” starting on page 7.

## 1.3 Historical Background

Before 1994, Apple designed computers around Motorola’s 68000 processor architecture. In March 1994, Apple started using the PowerPC microprocessor in its Macintosh family of desktop computers, replacing the Motorola 68000 series. Built-in emulation software let programs compiled to the 68LC040 instruction set continue to run in the PowerPC environment.

Apple has introduced a number of desktop computer configurations with PowerPC 601, 603, and 604 processors running at speeds up to 132 MHz. These computers run a PowerPC version of the Mac OS, with provisions for both software emulation and hardware coprocessing to run software compiled to the instruction set of the Intel x86 family of processors.

### 1.3.1 First-Generation Power Macintosh Computers

The first generation of Power Macintosh desktop computer products used a version of the PowerPC bus (called the Apple RISC bus, or ARBus) to connect built-in I/O devices to memory and to the PowerPC processor, and used NuBus to support plug-in expansion cards.

### 1.3.2 Second-Generation Power Macintosh Computers

The second generation of Power Macintosh desktop products, introduced in the summer of 1995, use the Peripheral Component Interconnect (PCI) bus to connect both internal I/O ASICs and expansion cards. They include built-in firmware, called Open Firmware, that implements IEEE *Standard 1275 for Boot (Initialization, Configuration) Firmware*. They use PowerPC 601, 603, and 604 processors and continue to support both the Mac OS and software compiled to the Intel x86 environment. Second-generation Power Macintosh computers also include these general features:

- PowerPC microprocessors for main system processing. The instruction set of the Motorola 68LC040 is supported through a built-in emulation system. Specific configurations also include x86-based processors to support DOS and Windows applications.
- PCI bus connections to all I/O and system expansion. Other buses (such as NuBus, SCSI, and IDE) are implemented by means of bridge ASICs connected to the PCI bus.
- System startup through Open Firmware. While the Mac OS continues to be the principal operating system for Power Macintosh computers, Open Firmware lets other operating systems that are ported to the PowerPC instruction set take control of the computer. Open Firmware also allows PCI expansion cards made for traditional PCs to function in Power Macintosh computers.
- Processor bus coherency. Memory systems connected directly to the PowerPC bus, including main RAM and all levels of cache, belong to a single coherency domain.
- Support for big-endian and little-endian addressing. Besides the support for both modes built into the PowerPC processor, storage subsystems such as frame buffers are accessible to software through both big-endian and little-endian apertures.
- Support for PowerPC-native interrupts and native device drivers.

## 1.4 CHRP Specification Power Macintosh Computers

Apple is currently designing computers that will comply with the CHRP specification. Table 2 lists the general characteristics of first- and second-generation



Power Macintosh computers and the currently projected characteristics of Power Macintosh computers that comply with the CHRP specification.

Table 2. CHRP-Macintosh feature comparison

Feature	First Generation Power Macintosh	Second Generation Power Macintosh	CHRP Specification Power Macintosh
Processor type	PowerPC 601	PowerPC 601, 603, or 604	All PowerPC models
Processor upgrade	None	By replacing processor subsystem card	By replacing processor subsystem card
OS support	Mac OS <sup>a</sup>	Mac OS <sup>a</sup>	Mac OS, OS/2, AIX, NetWare, Solaris, and Windows NT <sup>a</sup>
External cache	Up to 1 MB	Up to 4 MB	Up to 4 MB
RAM expansion	32-bit SIMM	64-bit DIMM	64-bit DIMM
Maximum RAM	72–264 MB	1 GB–1.5 GB	To be determined
NVRAM	Macintosh PRAM	8 KB	8 KB
Support for 21-inch monitors	None, 16 bpp, or 24 bpp	16 bpp or 24 bpp	16 bpp or 24 bpp
Sound	Macintosh 16-bit, 44.1 MHz, stereo input and output	Macintosh 16-bit, 44.1 MHz, stereo input and output	Both Macintosh and Sound Blaster-compatible stereo input and output
Internal hard disk	160 MB to 1 GB	250 MB to 2 GB	To be determined
Additional internal drives	One 5.25-inch, one or two 3.5-inch	One 5.25-inch, one or two 3.5-inch	To be determined
SCSI buses	1 fast internal, 1 external	1 fast internal, 1 external	To be determined
IDE bus	No	No	Yes
Floppy disk format	MFM or GCR	MFM or GCR	MFM only
Ethernet	AAUI	AAUI and 10baseT	10baseT
GeoPort serial ports	2	2	1–2
x86-compatible RS-232 serial ports	None	None	0–2
IEEE P1284 parallel port	None	None	In some models

Table 2. CHRP-Macintosh feature comparison (*continued*)

Feature	First Generation Power Macintosh	Second Generation Power Macintosh	CHRP Specification Power Macintosh
NuBus slots	1-3	None	None
PCI slots	None	3-6	3-6
ISA slot	None	None	None

a. DOS, Windows, and Windows 95 supported by DOS compatibility card



# The Mac I/O Chip

# 2

## 2.1 Overview

The Mac I/O (MIO) ASIC chip is an integrated I/O and DBDMA controller designed by Apple for use in CHRP specification Power Macintosh and third-party computers. It implements much of the Macintosh technology specified in later chapters of this book.

**Note:** This chapter is descriptive. It contains no specification requirements.

The MIO chip is a refinement of the O'Hare I/O controller used in the second generation of Power Macintosh computers. It implements a subset of the O'Hare functions with the addition of an ADB interface and a multiprocessor interrupt controller (Open PIC). The MIO chip complies with the CHRP specification; its PCI bus interface complies with the PCI specification, version 2.0.

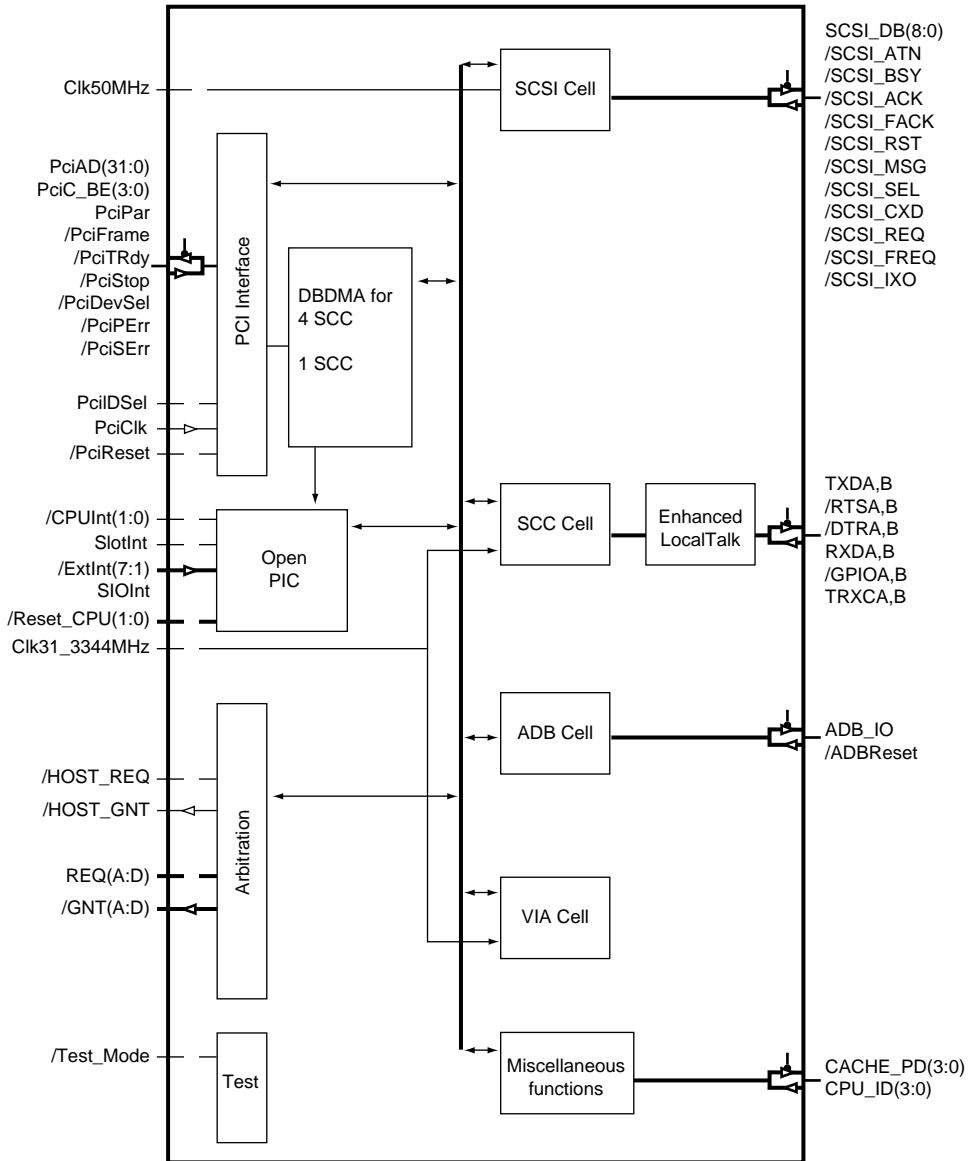
Third-party manufacturers of CHRP specification computers can obtain MIO chips from several vendors. Hardware designers who wish to create their own proprietary chips to perform MIO functions must follow the specifications in the rest of this book and may want to read this chapter as a sample implementation guide.

## 2.2 Internal Architecture

Figure 1 presents a block diagram of the of the internal architecture of the MIO ASIC designed by Apple.

**Note:** The signal lines shown in Figure 1 are listed and described in Table 45, beginning on page 72.

Figure 1. MIO block diagram



## 2.3 Features

Apple's MIO chip implements the following features:

- 33 MHz PCI bus interface that supports master and slave transactions
- integrated SCSI controller implemented through Macintosh Enhanced SCSI Hardware (MESH)
- integrated 85C30 SCC cell that supports Apple GeoPort and LocalTalk
- integrated versatile interface adapter (VIA) cell
- built-in 5-channel DMA controller that performs scatter/gather transfers based on a buffer list in main memory
- ADB hardware interface
- Open PIC interrupt controller that can support two processors
- sleep mode for reduced power consumption

## 2.4 Functional Units

The following functional units of the MIO chip, shown in Figure 1, are discussed in the next sections:

- PCI interface
- DBDMA controller
- SCSI controller
- SCC controller
- VIA interface
- ADB controller
- Open PIC multiprocessor interrupt controller

### 2.4.1 PCI Interface

Apple Computer's MIO chip complies with the PCI specification, Version 2.0, and acts as both a master and a slave on the PCI bus. As a master, it acts on behalf of the DBDMA controller to generate transactions on the PCI bus. As a slave, it decodes and responds to accesses to registers within the I/O controllers

as well as registers within the DBDMA controller. PCI bus transactions move data between system memory and the devices that the MIO chip controls, and read and write DBDMA command lists.

## 2.4.2 DBDMA Controller

The descriptor-based direct memory access (DBDMA) controller module is responsible for supplying the addresses associated with all the data transfers initiated by the MIO chip. The DBDMA controller is capable of sequencing through buffer descriptor lists stored in main memory to find the next buffer address after a channel exhausts the previous buffer. This frees the system from stringent interrupt response requirements after buffer completions. The buffer descriptor list is called a *command list*.

The DBDMA controller contains a register file that stores the current channel program pointers for each of five DBDMA channels. Another register file stores the current context (address, count, and flags) for each of the channels. One 32-bit incrementer for all five channels updates both the channel program pointers and the current buffer pointers. One 16-bit decremter for all five channels adjusts their count values. The DBDMA controller arbitrates data transfer requests from each of the I/O modules and generates PCI transactions that read and write the data buffers as well as the channel program entries.

For further information about DBDMA operation, see Chapter 5.

## 2.4.3 SCSI Controller

The MIO chip incorporates the Macintosh Enhanced SCSI Hardware (MESH) SCSI controller. A single DBDMA channel is used to support data transfers.

The SCSI controller's register set is mapped into the host processor's address space.

For further information about MESH SCSI control, see Chapter 4.

## 2.4.4 SCC Controller

The MIO chip incorporates a Macintosh Serial Communications Controller (SCC) cell that is Z85C30-compatible. The SCC is also used to drive the GeoPort telecom interface. External LocalTalk cables and GeoPort-compatible pods are used to transmit and receive on the serial lines that the SCC supports. The external serial port interface is described in Chapter 6.

Four DBDMA channels are used to support simultaneous transmission and reception on up to two SCC serial ports. The SCC's register set is mapped into

the host processor's address space. When performing register accesses, the MIO hardware ensures that the recovery time of the SCC is satisfied.

### 2.4.5 VIA Functions

To improve system compatibility, the MIO chip implements the full Macintosh Versatile Interface Adapter (VIA) functionality. This interface provides compatibility with earlier Macintosh products; no DBDMA support is provided.

For further information about VIA operation, see "VIA Support" beginning on page 61 and *Technical Introduction to the Macintosh Family*, second edition. This book is listed the bibliography.

### 2.4.6 ADB Controller

The MIO chip implements the Apple Desktop Bus (ADB), an asynchronous bus used for relatively slow Macintosh user-input devices such as the keyboard and the mouse. CHRP specification Power Macintosh computers have one or more ADB ports on the back panel.

For technical details about implementing the ADB, see Chapter 3.

### 2.4.7 Open PIC Interrupt Controller

The MIO chip incorporates the Open PIC multiprocessor interrupt control technology. The Open PIC register set is mapped into the host processor's address space. The Open PIC interrupt controller supports

- up to 20 I/O interrupt sources
- up to 2 microprocessors
- programmable interrupt vectors and task priority settings
- global task timers

For further information about the Open PIC technology, see *Open PIC Multiprocessor Interrupt Controller Register Interface Specification*. This book is listed in the bibliography (page 211).

**Note:** The Open PIC implementation inside the MIO chip can be disabled if an external interrupt controller is used. In this case, all internal interrupt sources are multiplexed out through the CpuInt[1:0]\_L, ExtInt[7:1]\_L, and SIOInt lines. For details, see Section 2.10.2.



## 2.5 Software Interface

Software communicates with the MIO ASIC in three ways: through register accesses, DBDMA transactions, and interrupts. These forms of software/hardware communication are described in the next sections.

### 2.5.1 Register Accesses

The PCI host bridge (PHB) is responsible for mapping the MIO chip's registers into a portion of the processor's address space. To do this, it must decode an address range corresponding to the MIO devices. When the host detects a processor access within that range, it generates a PCI transaction for the MIO chip. This mechanism is transparent to software. The only impact is some additional latency in the transactions. "Memory Map and Register Addressing" beginning on page 13, defines the MIO chip's register address space.

### 2.5.2 DBDMA Operation

DBDMA transfers are accomplished through the DBDMA architecture described in Chapter 5, "Descriptor-Based DMA," beginning on page 139.

Memory-resident data structures are used to describe lists of data buffers. The MIO chip automatically sequences through each buffer descriptor list, which is called a command list. The data structures also contain status information about the transfers. Upon completion of each data transfer, the DBDMA controller updates the corresponding channel command and (optionally) interrupts the processor so it can monitor the status of the transaction.

A set of registers within the MIO chip is used to initialize each DBDMA channel and perform control actions such as terminating the transfer. These registers are described in "Commands" beginning on page 48.

### 2.5.3 Interrupts

When a DMA transfer finishes, the DBDMA controller may send an interrupt to the host system. Other interrupts may originate in device interfaces—for example, in response to the asserting edge of the SCSI controller's interrupt signal. Certain devices (such as the SCC) may contain more than one interrupt source. For further information about interrupt processing see Section 2.10, beginning on page 57.

The MIO chip implements Open PIC, which supports up to two microprocessors simultaneously. It processes single 8259-type interrupts in two modes:

- The 8259 can act as a master, receiving a single interrupt from the Open PIC cell. This mode is used mainly in existing single-processor environments.
- The Open PIC cell can act as a master, receiving a single interrupt from the 8259. This mode is used mainly by existing multiprocessor designs and may be used by future single-processor environments.

## 2.6 Memory Map and Register Addressing

The MIO chip's registers occupy a 512 KB region of PCI memory space. The base address configuration register (0) is programmed by software to determine which 512 KB region the MIO chip decodes.

MIO memory space is allocated to device registers, central DBDMA controller registers, and individual DBDMA channel registers. Address lines AD(31:19) are programmable in the base address register. Lines AD(3:0) are used for SCC compatibility.

Table 3 shows how the PCI address bits [18:0] are decoded into controller register space, DBDMA channel register space, and device register space.

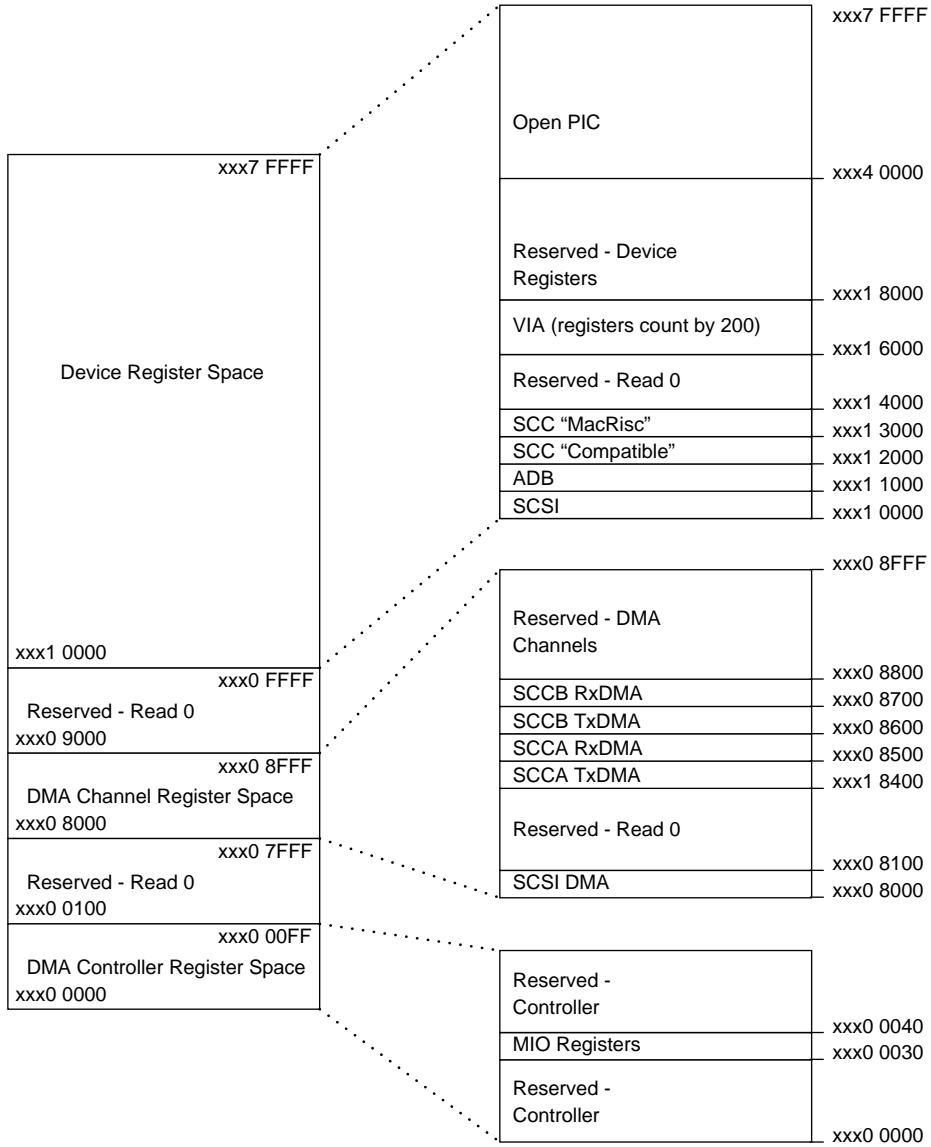
Table 3. Register space definitions

Offset (19-bit binary)	Space
0b000 0000 0000 RRRR RRRR	DBDMA Controller Register Space 8 bits RRRR RRRR select controller register
0b000 1000 cccc RRRR RRRR	DBDMA Channel Register Space 4 bits cccc select DBDMA channel 8 bits RRRR RRRR select channel register
0b0dd dddd RRRR RRRR rrrr	Device Register Space 6 bits dd dddd select device 8 bits RRRR RRRR select device register 4 bits rrrr select register for SCC compatibility port; otherwise bits rrrr are 0

Write actions to unmapped portions of the 512 KB space have no effect. Reads of unmapped locations return zeros.

The complete MIO memory map is shown in Figure 2.

Figure 2. MIO memory map



## 2.6.1 DBDMA Channel Number Assignments

The MIO chip supports 5 DBDMA channels. The channel number assignments, which correspond to the cccc bit field in Table 3, are shown in Table 4.

Table 4. DBDMA channel number assignments

Channel Number	Channel Name
0x00	SCSI
0x01	Not used (reads 0)
0x02	Not used (reads 0)
0x03	Not used (reads 0)
0x04	SCC channel A transmit
0x05	SCC channel A receive
0x06	SCC channel B transmit
0x07	SCC channel B receive

The channel number assignments are not contiguous in the MIO chip because some channels are not used. Unused channels return zeros in response to read accesses.

## 2.6.2 Device Number Assignments

The MIO chip provides access to several sets of device registers. Device numbers, corresponding to the dd dddd bit field in Table 3, are shown in Table 5.

Table 5. Device number assignments

Device Number	Device Name
0x10	SCSI
0x11	Not used (reads 0)
0x12	SCC (compatibility port)
0x13	SCC (MacRISC port)

Table 5. Device number assignments (*continued*)

Device Number	Device Name
0x14–0x15	Not used (reads 0)
0x16–0x17	VIA
0x18–0x3F	Not used (reads 0)
0x40–0x7F	Open PIC

The device number assignments were contiguous in previous Macintosh I/O ASICs. They are not contiguous in the MIO chip because some devices are not supported and new devices have been added. The MIO chip does not support Apple Media Access for Ethernet (MACE), the Ethernet PROM, the second SCSI controller, or IOBus devices 5 and 6; therefore, device spaces 0x01 (MACE), 0x08 (SCSI1), 0x09 (Ethernet PROM), 0x0E (IOBus device 5), and 0x0F (IOBus device 6) return zeros in response to read accesses.

### 2.6.3 Register Map

Table 6 presents a detailed register map for the MIO chip. It shows which address each device's or channel's register set occupies. Table 6 lists all of the MIO registers; specific details about the contents of particular registers are given later in this chapter.

Table 6. MIO register map

Offset	Device	Read Register	Write Register	Register Type
0x00030		CachePD	CachePD	DMA control
0x00034		IDs	IDs	DMA control
0x00038		Feature Control	Feature Control	DMA control
0x08000	SCSI	n.a.	Channel Control	DMA channel
0x08004	SCSI	ChannelStatus	n.a.	DMA channel
0x0800C	SCSI	CommandPtr	CommandPtr	DMA channel
0x08010	SCSI	InterruptSelect	InterruptSelect	DMA channel
0x08014	SCSI	BranchSelect	BranchSelect	DMA channel

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x08018	SCSI	WaitSelect	WaitSelect	DMA channel
0x08400	SCC A Tx	n.a.	Channel Control	DMA channel
0x08404	SCC A Tx	ChannelStatus	n.a.	DMA channel
0x0840C	SCC A Tx	CommandPtr	CommandPtr	DMA channel
0x08410	SCC A Tx	InterruptSelect	InterruptSelect	DMA channel
0x08414	SCC A Tx	BranchSelect	BranchSelect	DMA channel
0x08418	SCC A Tx	WaitSelect	WaitSelect	DMA channel
0x08500	SCC A Rx	n.a.	Channel Control	DMA channel
0x08504	SCC A Rx	ChannelStatus	n.a.	DMA channel
0x0850C	SCC A Rx	CommandPtr	CommandPtr	DMA channel
0x08510	SCC A Rx	InterruptSelect	InterruptSelect	DMA channel
0x08514	SCC A Rx	BranchSelect	BranchSelect	DMA channel
0x08518	SCC A Rx	WaitSelect	WaitSelect	DMA channel
0x08600	SCC B Tx	n.a.	Channel Control	DMA channel
0x08604	SCC B Tx	ChannelStatus	n.a.	DMA channel
0x0860C	SCC B Tx	CommandPtr	CommandPtr	DMA channel
0x08610	SCC B Tx	InterruptSelect	InterruptSelect	DMA channel
0x08614	SCC B Tx	BranchSelect	BranchSelect	DMA channel
0x08618	SCC B Tx	WaitSelect	WaitSelect	DMA channel
0x08700	SCC B Rx	n.a.	Channel Control	DMA channel
0x08704	SCC B Rx	ChannelStatus	n.a.	DMA channel
0x0870C	SCC B Rx	CommandPtr	CommandPtr	DMA channel
0x08710	SCC B Rx	InterruptSelect	InterruptSelect	DMA channel
0x08714	SCC B Rx	BranchSelect	BranchSelect	DMA channel
0x08718	SCC B Rx	WaitSelect	WaitSelect	DMA channel

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x10000	SCSI	Transfer Count0 LSB	Transfer Count0 LSB	Device
0x10010	SCSI	Transfer Count1 MSB	Transfer Count1 MSB	Device
0x10020	SCSI	FIFO	FIFO	Device
0x10030	SCSI	Sequence	Sequence	Device
0x10040	SCSI	BusStatus0	BusStatus0	Device
0x10050	SCSI	BusStatus1	BusStatus1	Device
0x10060	SCSI	Bus FIFO Counter	Bus FIFO Counter	Device
0x10070	SCSI	Exception	Exception	Device
0x10080	SCSI	Error	Error	Device
0x10090	SCSI	Interrupt Mask	Interrupt Mask	Device
0x100A0	SCSI	Interrupt	Interrupt	Device
0x100B0	SCSI	Source ID	Source ID	Device
0x100C0	SCSI	Destination ID	Destination ID	Device
0x100D0	SCSI	SyncParms	SyncParms	Device
0x100E0	SCSI	MeshID	n.a.	Device
0x100F0	SCSI	Selection TimeOut	Selection TimeOut	Device
0x11000	ADB	Interrupt Status	n.a.	Device
0x11010	ADB	Command	Command	Device
0x11020	ADB	Data1	Data1	Device
0x11030	ADB	Data2	Data2	Device
0x11040	ADB	Data3	Data3	Device
0x11050	ADB	Data4	Data4	Device
0x11060	ADB	Data5	Data5	Device
0x11070	ADB	Data6	Data6	Device
0x11080	ADB	Data7	Data7	Device

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x11090	ADB	Data8	Data8	Device
0x110A0	ADB	Interrupt Source Enable	Interrupt Source Enable	Device
0x110B0	ADB	Data Type/Count	Data Type/Count	Device
0x110C0	ADB	Error Status	n.a.	Device
0x110D0	ADB	Control	Control	Device
0x110E0	ADB	Autopoll Control	Autopoll Control	Device
0x110F0	ADB	Active Dev Addr HIGH	Active Dev Addr HIGH	Device
0x11100	ADB	Active Dev Addr LOW	Active Dev Addr LOW	Device
0x11110	ADB	Test	Test	Device
0x12000	SCC compatible (See Section 2.11.3.1.2)	Port B Command	Port B Command	Device
0x12002	SCC compatible	Port A Command	Port A Command	Device
0x12004	SCC compatible	Port B Data	Port B Data	Device
0x12006	SCC compatible	Port A Data	Port A Data	Device
0x12008	SCC compatible	Enhance Reg. B/ Version	Enhance Reg. B	Device
0x1200A	SCC compatible	Enhance Reg. A/ Version	Enhance Reg. A	Device
0x12080	SCC	Rec Count	Rec Count	Device
0x12090	SCC	Start A	Start A	Device
0x120A0	SCC	Start B	Start B	Device
0x120B0	SCC	Detect AB	n.a.	Device
0x13000	SCC MacRISC	Port B Command	Port B Command	Device
0x13010	SCC MacRISC	Port B Data	Port B Data	Device
0x13020	SCC MacRISC	Port A Command	Port A Command	Device
0x13030	SCC MacRISC	Port A Data	Port A Data	Device



Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x13040	SCC MacRISC	Enhance Reg. B/ Version	Enhance Reg. B	Device
0x13050	SCC MacRISC	Enhance Reg. A/ Version	Enhance Reg. A	Device
0x13080	SCC	Rec Count	Rec Count	Device
0x13090	SCC	Start A	Start A	Device
0x130A0	SCC	Start B	Start B	Device
0x130B0	SCC	Detect AB	n.a.	Device
0x16000	VIA	Input Register B	Output Register B	Device
0x16200	VIA	Input Register A	Output Register A	Device
0x16400	VIA	Data Direction Register B	Data Direction Register B	Device
0x16600	VIA	Data Direction Register A	Data Direction Register A	Device
0x16800	VIA	T1 Low-Order Counter	T1 Low-Order Latch	Device
0x16A00	VIA	T1 High-Order Counter	T1 High-Order Counter	Device
0x16C00	VIA	T1 Low-Order Latch	T1 Low-Order Latch	Device
0x16E00	VIA	T1 High-Order Latch	T1 High-Order Latch	Device
0x17000	VIA	T2 Low-Order Counter	T2 Low-Order Latch	Device
0x17200	VIA	T2 High-Order Counter	T2 High-Order Counter	Device
0x17400	VIA	Shift Register	Shift Register	Device
0x17600	VIA	Auxiliary Control	Auxiliary Control	Device
0x17800	VIA	Peripheral Control	Peripheral Control	Device
0x17A00	VIA	Interrupt Flag Register	Interrupt Flag Register	Device
0x17C00	VIA	Interrupt Enable Register	Interrupt Enable Register	Device
0x17E00	VIA	Input Register A	Output Register A	Device

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x41000	Open PIC	Feature Reporting Register 0	n.a.	Device
0x41020	Open PIC	Global Config Reg. 0	Global Config Reg. 0	Device
0x41080	Open PIC	Vendor Identification	n.a.	Device
0x41090	Open PIC	Processor Init	Processor Init	Device
0x410A0	Open PIC	IPI 0 Vector/Priority	IPI 0 Vector/Priority	Device
0x410B0	Open PIC	IPI 1 Vector/Priority	IPI 1 Vector/Priority	Device
0x410C0	Open PIC	IPI 2 Vector/Priority	IPI 2 Vector/Priority	Device
0x410D0	Open PIC	IPI 3 Vector/Priority	IPI 3 Vector/Priority	Device
0x410E0	Open PIC	Spurious Vector	Spurious Vector	Device
0x410F0	Open PIC	Timer Frequency Reporting	Timer Frequency Reporting	Device
0x41100	Open PIC	Timer 0 Current Count	Timer 0 Current Count	Device
0x41110	Open PIC	Timer 0 Base Count	Timer 0 Base Count	Device
0x41120	Open PIC	Timer 0 Vector/Priority	Timer 0 Vector/Priority	Device
0x41130	Open PIC	Timer 0 Destination	Timer 0 Destination	Device
0x41140	Open PIC	Timer 1 Current Count	Timer 1 Current Count	Device
0x41150	Open PIC	Timer 1 Base Count	Timer 1 Base Count	Device
0x41160	Open PIC	Timer 1 Vector/Priority	Timer 1 Vector/Priority	Device
0x41170	Open PIC	Timer 1 Destination	Timer 1 Destination	Device
0x41180	Open PIC	Timer 2 Current Count	Timer 2 Current Count	Device
0x41190	Open PIC	Timer 2 Base Count	Timer 2 Base Count	Device
0x411A0	Open PIC	Timer 2 Vector/Priority	Timer 2 Vector/Priority	Device
0x411B0	Open PIC	Timer 2 Destination	Timer 2 Destination	Device
0x411C0	Open PIC	Timer 3 Current Count	Timer 3 Current Count	Device
0x411D0	Open PIC	Timer 3 Base Count	Timer 3 Base Count	Device

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x411E0	Open PIC	Timer 3 Vector/Priority	Timer 3 Vector/Priority	Device
0x411F0	Open PIC	Timer 3 Destination	Timer 3 Destination	Device
0x50000	Open PIC	Interrupt Source 0 (SIO) Vector/Priority	Interrupt Source 0 (SIO) Vector/Priority	Device
0x50010	Open PIC	Int Source 0 (SIO) Destination	Int Source 0 (SIO) Destination	Device
0x50020	Open PIC	Interrupt Source 1 Vector/Priority	Interrupt Source 1 Vector/Priority	Device
0x50030	Open PIC	Int Source 1 Destination	Int Source 1 Destination	Device
0x50040	Open PIC	Interrupt Source 2 Vector/Priority	Interrupt Source 2 Vector/Priority	Device
0x50050	Open PIC	Int Source 2 Destination	Int Source 2 Destination	Device
0x50060	Open PIC	Interrupt Source 3 Vector/Priority	Interrupt Source 3 Vector/Priority	Device
0x50070	Open PIC	Int Source 3 Destination	Int Source 3 Destination	Device
0x50080	Open PIC	Interrupt Source 4 Vector/Priority	Interrupt Source 4 Vector/Priority	Device
0x50090	Open PIC	Int Source 4 Destination	Int Source 4 Destination	Device
0x500A0	Open PIC	Interrupt Source 5 Vector/Priority	Interrupt Source 5 Vector/Priority	Device
0x500B0	Open PIC	Int Source 5 Destination	Int Source 5 Destination	Device
0x500C0	Open PIC	Interrupt Source 6 Vector/Priority	Interrupt Source 6 Vector/Priority	Device
0x500D0	Open PIC	Int Source 6 Destination	Int Source 6 Destination	Device
0x500E0	Open PIC	Interrupt Source 7 Vector/Priority	Interrupt Source 7 Vector/Priority	Device

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x500F0	Open PIC	Int Source 7 Destination	Int Source 7 Destination	Device
0x50100	Open PIC	Interrupt Source 8 Vector/Priority	Interrupt Source 8 Vector/Priority	Device
0x50110	Open PIC	Int Source 8 Destination	Int Source 8 Destination	Device
0x50120	Open PIC	Interrupt Source 9 Vector/Priority	Interrupt Source 9 Vector/Priority	Device
0x50130	Open PIC	Int Source 9 Destination	Int Source 9 Destination	Device
0x50140	Open PIC	Interrupt Source 10 Vector/Priority	Interrupt Source 10 Vector/Priority	Device
0x50150	Open PIC	Int Source 10 Destination	Int Source 10 Destination	Device
0x50160	Open PIC	Interrupt Source 11 Vector/Priority	Interrupt Source 11 Vector/Priority	Device
0x50170	Open PIC	Int Source 11 Destination	Int Source 11 Destination	Device
0x50180	Open PIC	Interrupt Source 12 Vector/Priority	Interrupt Source 12 Vector/Priority	Device
0x50190	Open PIC	Int Source 12 Destination	Int Source 12 Destination	Device
0x501A0	Open PIC	Interrupt Source 13 Vector/Priority	Interrupt Source 13 Vector/Priority	Device
0x501B0	Open PIC	Int Source 13 Destination	Int Source 13 Destination	Device
0x501C0	Open PIC	Interrupt Source 14 Vector/Priority	Interrupt Source 14 Vector/Priority	Device
0x501D0	Open PIC	Int Source 14 Destination	Int Source 14 Destination	Device
0x501E0	Open PIC	Interrupt Source 15 Vector/Priority	Interrupt Source 15 Vector/Priority	Device
0x501F0	Open PIC	Int Source 15 Destination	Int Source 15 Destination	Device

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x50200	Open PIC	Interrupt Source 16 Vector/Priority	Interrupt Source 16 Vector/Priority	Device
0x50210	Open PIC	Int Source 16 Destination	Int Source 16 Destination	Device
0x50220	Open PIC	Interrupt Source 17 Vector/Priority	Interrupt Source 17 Vector/Priority	Device
0x50230	Open PIC	Int Source 17 Destination	Int Source 17 Destination	Device
0x50240	Open PIC	Interrupt Source 18 Vector/Priority	Interrupt Source 18 Vector/Priority	Device
0x50250	Open PIC	Int Source 18 Destination	Int Source 18 Destination	Device
0x50260	Open PIC	Interrupt Source 19 Vector/Priority	Interrupt Source 19 Vector/Priority	Device
0x50270	Open PIC	Int Source 19 Destination	Int Source 19 Destination	Device
0x60040	Open PIC	P0 IPI 0 Dispatch	P0 IPI 0 Dispatch	Device
0x60050	Open PIC	P0 IPI 1 Dispatch	P0 IPI 1 Dispatch	Device
0x60060	Open PIC	P0 IPI 2 Dispatch	P0 IPI 2 Dispatch	Device
0x60070	Open PIC	P0 IPI 3 Dispatch	P0 IPI 3 Dispatch	Device
0x60080	Open PIC	P0 Current Task Priority	P0 Current Task Priority	Device
0x600A0	Open PIC	P0 Int Acknowledge	P0 Int Acknowledge	Device
0x600B0	Open PIC	P0 End-of-Interrupt	P0 End-of-Interrupt	Device
0x61040	Open PIC	P1 IPI 0 Dispatch	P1 IPI 0 Dispatch	Device
0x61050	Open PIC	P1 IPI 1 Dispatch	P1 IPI 1 Dispatch	Device
0x61060	Open PIC	P1 IPI 2 Dispatch	P1 IPI 2 Dispatch	Device
0x61070	Open PIC	P1 IPI 3 Dispatch	P1 IPI 3 Dispatch	Device
0x61080	Open PIC	P1 Current Task Priority	P1 Current Task Priority	Device

Table 6. MIO register map (*continued*)

Offset	Device	Read Register	Write Register	Register Type
0x610A0	Open PIC	P1 Int Acknowledge	P1 Int Acknowledge	Device
0x610B0	Open PIC	P1 End-of-Interrupt	P1 End-of-Interrupt	Device

## 2.7 General Functions

This section describes functions that operate generally over the various I/O controllers in the MIO chip. For specific information about individual I/O controllers, see Section 2.11.

### 2.7.1 Cache Indicator

The presence of cache in the host system is indicated by four CachePD bits at offset 0x00030 in the MIO chip's address space. In addition, a cache ID ROM (consisting of an X24C00 serial EEPROM) is located on the cache SIMM. The cache ID ROM contains cache size information, which power-on self-test (POST) routines may use to test the cache SIMM. It also contains timing information for the SIMM. The contents of the cache ID ROM are specified by the SIMM designer.

The interface to the cache ID ROM consists of the lines SCLK and SDAT, which use two of the four CachePD pins shown in Table 45, beginning on page 72. All four pins must be pulled high on the motherboard. If any of the four pins are low, it indicates that a cache is present and gives static configuration information. If all four pins are high, then software should communicate with the cache ID ROM to fetch cache information. This arrangement lets systems support caches that span the full range of available performance and features. Table 7 shows the format of the CachePD register.

Table 7. CachePD register

7	6	5	4	3	2	1	0
0	0	SDAT_OE	SCLK_OE	PD3	PD2	PD1/SDAT	PD0/SCLK

Bits PD[3:0] reflects the status of the CachePD[3:0] lines. When SCLK\_OE is low, PD0 is an input. When SCLK\_OE is high, PD0 becomes an output and the value being written into bit 0 is reflected on the external pin. The behavior is the same for SDAT\_OE and SDAT. After a reset, lines PD[3:0] become inputs.

## 2.7.2 CPU IDs

The MIO chip supports the traditional CPUID register format, found in earlier generations of Macintosh computers, that identifies the Macintosh model. To identify the CPU uniquely, software can read the CPUID register in addition to the PCI device's identification register. The CPUID register is the least significant byte of the IDs register at MIO offset 0x00034. Its format is shown in Table 8.

Table 8. CPUID register

7	6	5	4	3	2	1	0
in3	in2	in1	in0	out	oe2	oe1	oe0

The MIO chip supports extended CPUIDs in the same manner that extended monitor IDs were supported in previous Macintosh computers. When software reads 0xF for the CPUID, it tests for the extended modes. Extended combinations are limited to CPUID[2:0], to keep the number of combinations manageable. This allows 15 (0:E) resistor combinations and 37 extended combinations for a total of 52. For further information about automatic monitor recognition, see Macintosh Technical Note 326, listed in the bibliography.

**Note:** The MIO chip's PCI device ID is 0x000E.

## 2.7.3 Feature Controls

The MIO chip's 32-bit Feature Control register, at offset 0x00038, provides software access for mode selection, programmable functions, and power control. Table 9 describes its bits.

Table 9. Feature Control register

Bit	Name	Description	Default
0	SCC_CELL_EN	Stops the SCC clock when low. Software must guarantee that DMA transactions are finished.	1
1	SCSI_CELL_EN	Stops the SCSI clock when low. Software must guarantee that DMA transactions are finished.	1

Table 9. Feature Control register (*continued*)

Bit	Name	Description	Default
2	ScCAEnable	Enables the SCC outputs TXDA, RTSA_L, and DTRA_L, which are tristated when ScCAEnable is low. The following inputs are blocked when ScCAEnable is low: GPIOA_L (forced high), TRXCA (forced low), and RXDA (forced low).	1
3	ScCBEnable	Enables the SCC outputs TXDB, RTSB_L, and DTRB_L, which are tristated when ScCBEnable is low. The following inputs are blocked when ScCBEnable is low: GPIOB_L (forced high), TRXCB (forced low), and RXDB (forced low).	1
4	ARB_BYPASS	When set to one, this bit allows the internal arbiter to be bypassed. The internal REQ_OUT_L is multiplexed with GNTA_L, GNT_IN_L is multiplexed with REQA_L, and ArbCritical is multiplexed with GNTB_L. For details, see Section 2.8.8 on page 41.	1
5	ResetSCC	When active, SccWr_L and SccRd_L in the ScCIOM are asserted simultaneously. This resets the SCC cell.	0
6	MPIC_Enable	When this bit is set to 1, the Open PIC cell is enabled and external interrupt sources are sampled through the ExtInt[7:1] and SIOInt lines. When cleared to 0, the Open PIC cell is disabled and all the MIO chip's internal interrupt sources are multiplexed out through CpuInt[1:0], ExtInt[7:1], and SIOInt. The Reset_CPU[1:0] bits are forced high.	1
7	SlowScPClk	When this bit is high, ScPClk is 15.6672 MHz. When it is low, ScPClk becomes 25 MHz.	1
8	MpicIsMaster	When this bit is high, the ScPClk rate is 15.6672 MHz. When it is low, the ScPClk rate is 25 MHz.	0
9:31	Reserved	These bits always read 0.	0..0

### 2.7.4 Sleep Mode

The assertion of the Clock Fail Warning (CFW\_L) input signal (see Table 45, beginning on page 72) indicates to the MIO chip that sleep mode is approaching. The MIO arbiter removes all GNT signals and waits for the current ownership to end (PciFrame\_L and PciIRdy\_L deasserted). Once the MIO chip is master again, it drives the PciAD(31:0), PciC\_BE(3:0), and PciPar lines low. These signals remain low during sleep. The MIO chip also stops the 50 MHz and 31.3344 MHz clocks cleanly at this point. The PCI clock should be stopped



cleanly 16 ms after CFW\_L is asserted.

When sleep mode is deactivated, the CFW\_L line is deasserted (goes high). The PCI clock must already be running again; the MIO chip will resume normal operation and tristate the PCI outputs (if it is not the highest master).

## 2.8 PCI Bus Interface

This section provides a detailed description of the PCI bus interface to the MIO chip. This interface adheres to the standard PCI bus interface protocols and electrical parameters defined in the PCI specification. The MIO chip operates as a PCI bus slave to implement system accesses to DBDMA unit registers, DBDMA channel registers, and device registers. It operates as a PCI bus master to perform read and write transactions to locations in the system address space. The MIO chip operates synchronously with the PCI bus clock, supporting a clock frequency of up to 33 MHz.

### 2.8.1 General PCI Bus Transactions

The MIO chip is mapped into the PCI bus address space as a device containing a 256-byte region of configuration registers and a 512 KB region of control and status registers. All configuration registers are accessed through PCI bus configuration transactions. All control and status registers are accessed through PCI bus memory transactions. Read and write transfer cycles proceed as multiplexed address/data transfers. There may be several data cycles per transaction. All signals are synchronized to the rising edge of the PCI bus clock, PciClk.

#### 2.8.1.1 Address Transactions

During the address phase of PCI memory or I/O transactions, the MIO chip decodes the transaction address by comparing the address presented on the PCI bus to a base register stored internally. If the PCI bus address matches the address range specified in the base address register, the MIO chip responds to the transaction by asserting the PciDevSel\_L signal. The MIO chip supports a 512 KB memory space segment that is divided into regions for DBDMA unit registers, DBDMA channel registers, and device control/status registers.

Configuration transactions are identical to memory transactions except that the transaction decoding is provided externally, using the PciIDSEL signal. PciIDSEL is active only during the address phase of a PCI bus configuration cycle, after the MIO chip has asserted PciDevSel\_L. Configuration register addressing is performed by the PciAD[7:2] signals and the PciC\_BE[3:0] lines.

This lets the MIO chip decode a 256-byte region of configuration space.

The requested address space (configuration or memory) is determined during the address phase by the `PciCmd[3:0]` signals, defining 16 possible transfer modes. The MIO chip supports the subset of transfer modes summarized in Table 10 and ignores all other transfer mode encodings.

Table 10. PCI bus commands supported by the MIO chip

PciCmd[3:0]	Name	Command Type
0x6	MemRd	Memory segment read
0x7	MemWr	Memory segment write
0xA	ConfRd	Configuration segment read
0xB	ConfWr	Configuration segment write
0xC	MemRdMult	Memory read multiple, aliased to 0x6–MemRd
0xE	MemRdLine	Memory read long, aliased to 0x6–MemRd (generated by the MIO chip as a bus master)
0xF	MemWr and Inval	Postable memory write aliased to 0x7–MemWr (generated by the MIO chip as a bus master)

The `PciDevSel_L` signal is synchronized to the system clock and is asserted only when the MIO chip is ready to respond to a valid transfer. To avoid generating a response on `PciDevSel_L` for invalid commands, the MIO chip uses `PciCmd[3:0]` and `PciAD[31:0]` (or `PciIDSel`) to generate the `PciDevSel_L` signal. Devices on the PCI bus may take more than one clock cycle to generate `PciDevSel_L`, but they must respond within three clock cycles. The response time is set by the value in bits [10:9] of the PCI bus status register (see “Device Status” beginning on page 39). This field is read-only; its value indicates the maximum response time, as shown in Table 11.

Table 11. Device response times

[10:9] Value	Response Time, Cycles
0x0	1
0x1	2
0x2	3
0x3	Reserved

The MIO chip will respond in 2 cycles to all PCI bus transactions, and bits [10:9] in the PCI bus status register will always return 0x0.

### 2.8.1.2 Data Transactions

The PCI bus interface on the MIO chip is idle when `PciFrame_L` and `PciIRdy_L` are both negated. The first clock edge on which `PciFrame_L` is asserted by the master (initiator) is the address phase. The address and bus command code are transferred on that clock edge. The next clock edge begins the first of one or more data phases. During a data phase, data is transferred between the master and slave (target) on each successive clock edge while both `PciIRdy_L` and `PciTRdy_L` are asserted. Wait states may be inserted by either the initiator or the target by deasserting `PciIRdy_L` or `PciTRdy_L` respectively. However, these two signals must be driven unconditionally; one device cannot delay asserting its `PciXrdy_L` until the other device has asserted its `PciXrdy_L` signal.

When the initiator intends to complete only one more data transfer (which could be immediately after the address phase), `PciFrame_L` is negated and `PciIRdy_L` is asserted simultaneously. This indicates that the initiator is ready to perform the last data transfer. After the target acknowledges the final data transfer by asserting `PciTRdy_L`, the interface returns to idle with both `PciFrame_L` and `PciIRdy_L` negated.

### 2.8.1.3 Slave Address Decoding

The PCI specification defines three possible transaction address spaces: memory space, I/O space, and configuration space. The memory and I/O address spaces are specific to the application. The configuration space is compulsory and is defined to support the PCI bus hardware configuration process. The MIO chip uses the memory space to access its control and status registers. In all the MIO chip's PCI bus transactions, accesses are decoded to a 32-bit word address using `PciAD[31:2]`. Individual bytes are selected using `PciBe[3:0]_L`. For multiple-word PCI bus transactions, the address is assumed to increment by one 32-bit word after each data phase until the transaction is terminated. The initiator places 00 on lines `PciAD[1:0]` during the address phase for all accesses. The other three encodings are reserved. The MIO chip ignores lines `PciAD[1:0]` during the address phase.

Configuration accesses are used to program the segment base registers and to enable the MIO memory segment on the PCI bus. The MIO chip supports a 256-byte configuration segment, accesses to which are decoded using lines `PciAD[7:2]` and `PciIDSel`. During configuration segment accesses, the lower

eight address lines (PciAD[7:0]) are used to address a 256-byte configuration segment location containing device and vendor information and the segment base registers. Chip select is performed using the PciIDSel signal.

The MIO chip supports a 512 KB memory segment for its control and status registers. The MIO chip stores and decodes 13 bits of the configuration address presented on lines PciAD[31:19] to identify the 512 KB memory segment. The PCI bus configuration must guarantee that the MIO memory segment is allocated on a 512 KB boundary. During a memory read or write transaction to the MIO chip, the 17-bit memory base register is compared with lines PciAD[31:19] to generate a valid decode for the 512 KB memory segment. Lines PciAd[17:16] are used to select between the device (0b01 or 0b10) and DBDMA registers (0b00). Within the DBDMA space, line PciAD[15] is used to select between unit registers and channel registers.

The MIO chip's PCI bus memory segment contains locations that are flagged as reserved or undefined. The MIO chip will accept transactions with all 256 bytes of the configuration segment and with all 512 bytes of the memory segment. If the location corresponding to the transaction address (within the configuration and memory segments) is undefined or reserved, then reads return zeros and writes have no effect. Normally, the data phases will be completed with PciTRdy\_L asserted. In addition, any undefined bits within a valid location return zeros and the location is not affected by writes.

#### 2.8.1.4 Bus Control and Turnaround

The MIO chip shares its PciTRdy\_L, PciStop\_L, PciDevSel\_L, PciPar, and PciAD[31:0] signals with other devices on the PCI bus. A turnaround cycle is required to avoid contention when one device stops driving a signal and another device begins. A turnaround cycle must last at least 1 clock cycle long. It is indicated in timing diagrams (such as Figure 3) by two circular arrows. The requirements for turnaround cycles vary depending on the signal. For example, lines PciTRdy\_L, PciStop\_L, and PciDevSel\_L use the address phase as their turnaround cycle; lines PciAD[31:0] use the idle time between transactions or transaction phases as their turnaround time.

The control lines PciTRdy\_L, PciStop\_L, and PciDevSel\_L must be driven high for one cycle after the end of the last data phase of a transaction and before a turnaround cycle is permitted (when the MIO chip tristates the signals). The control signals may be driven low immediately after their turnaround cycle at the start of a transaction.

### 2.8.1.5 Byte Enables

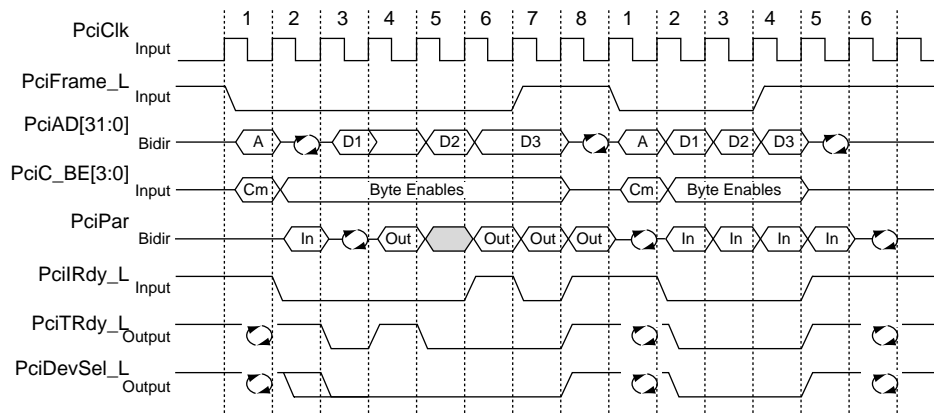
The initiator can select bytes lanes within a word by using the PciCmd[3:0] lines as PciC\_BE[3:0] byte enable lines. Even if a byte lane is selected, all the data lines must be driven to a known state by the initiator or target for every cycle during which a valid transfer occurs on the bus. The PciC\_BE lines may change for each data phase. They are valid on the edge of the clock that starts the data phase, and they remain valid during that phase. If no byte enable lines are asserted, the MIO chip completes the transaction by asserting PciTRdy\_L and providing parity if the transaction is a read request.

### 2.8.1.6 Bus Parity

All initiators and targets must generate even parity for every clock cycle in which a valid address phase or valid data phase is completed. Parity is signaled one clock cycle after the corresponding transfer. The device driving the PciAD[31:0] lines drives the PciPar signal. The number of ones on lines PciAD[31:0], PciCmd[3:0] (PciC\_BE[3:0]), and PciPar must be even.

During address and data phases, parity covers all 32 PciAD and 4 PciCmd (PciC\_BE) lines, regardless of whether or not they carry meaningful information. Even byte lanes that are not transferring valid data must be stable during the transaction. The MIO chip generates parity on reads but ignores parity on writes. Figure 3 shows an example of parity generation. Note that the PciPar signal lags the PciAD lines by 1 clock cycle, including turnaround cycles.

Figure 3. PCI bus parity generation



### 2.8.1.7 Transaction Termination

Normally, the initiator terminates a transaction when it has finished transferring data. However, the target may disconnect from the initiator by driving the PciStop\_L signal active. The MIO chip asserts the PciStop\_L signal in either of two cases:

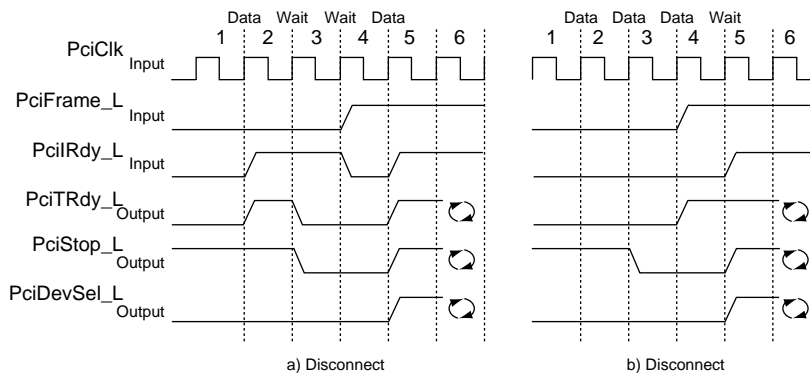
- when a transaction of more than 4 bytes is performed
- when the MIO chip is temporarily unable to complete a transaction

The target asserts PciStop\_L as a request to the master to terminate the transaction. PciStop\_L remains asserted until PciFrame\_L is negated. The relationship between PciIRdy\_L and PciTRdy\_L is independent of the relationship between PciStop\_L and PciFrame\_L. Therefore, a data transfer may or may not occur during the target's request for termination. If PciTRdy\_L is negated when PciStop\_L is asserted, it indicates that the target will not transfer any more data. The master does not wait for a final data transfer, as it would in a normal completion termination.

The type of termination where data is not transferred during the last data phase is called a *retry*. A normal completion termination, where data is transferred during the last data phase, is called a *disconnect*. The MIO chip supports both types of terminations. A disconnect is signaled when PciStop\_L is asserted and is held asserted until PciFrame\_L is negated. PciFrame\_L is negated as soon as possible after PciStop\_L is asserted. PciStop\_L is negated one cycle after PciFrame\_L is asserted.

Figure 4 shows two possible kinds of disconnect.

Figure 4. PCI bus target-initiated termination



In Figure 4a, the master could not finish the last data phase immediately so `PciStop_L` was asserted while `PciIRdy_L` was negated. Therefore, the final data transfer takes place after `PciFrame_L` is negated.

In Figure 4b, however, the master is ready to finish a data transfer (`PciIRdy_L` is asserted) when `PciStop_L` is asserted. In this case, two more data transfers could finish during cycles 3 and 4. The target must terminate the second transfer by asserting `PciTRdy_L` after the data transfer during cycle 3. As a result, a wait cycle is inserted into clock cycle 4. Note that `PciStop_L` remains active for one clock cycle after `PciFrame_L` is negated.

Only one data transfer may finish once `PciStop_L` is asserted; the target may not try to finish a second transfer once `PciStop_L` is active.

## 2.8.2 Read Transactions

A PCI bus read transaction starts with an address phase that occurs when `PciFrame_L` is asserted for the first time. During the address phase, lines `PciAD[31:2]` contain a valid word address, `PciAD[1:0]` are zeros, and `PciCmd[3:0]` contain a valid bus command. The address phase always lasts for one clock cycle.

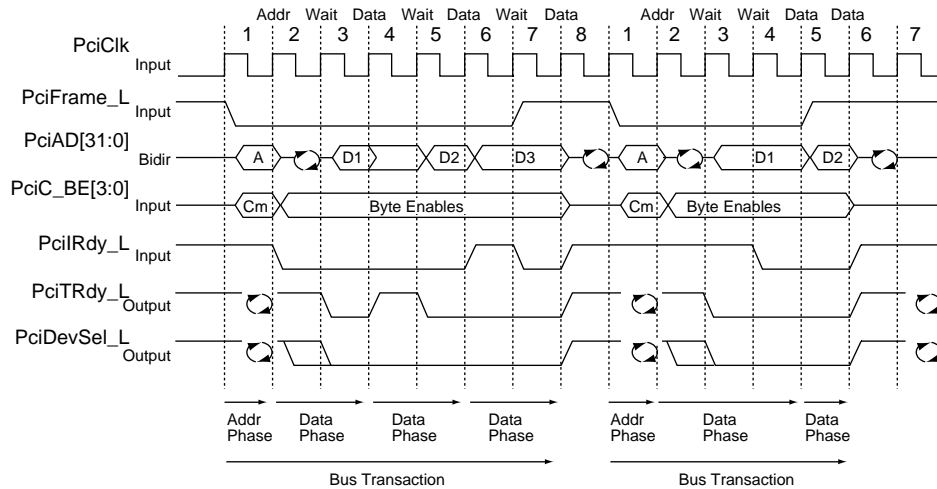
The clock cycle following the address phase is the first data phase. During data phases, lines `PciC_BE[3:0]` indicate which byte lanes are valid during the current transfer. A data phase may consist of a data transfer cycle and one or more wait cycles. A data phase starts immediately after an address phase or a preceding data phase and terminates when both `PciTRdy_L` and `PciIRdy_L` are asserted. Data is transferred while `PciTRdy_L` and `PciIRdy_L` are asserted.

The `PciC_BE` lines are valid for every clock cycle of the data phase, but may change from one data phase to the next.

The first clock cycle of the first data phase must be a turnaround cycle. This is enforced by the target asserting `PciTRdy_L`. Once the `PciAD` lines are enabled by the target, they must remain enabled throughout the transaction. `PciDevSel_L` must be driven by the target within 3 clock cycles of the address phase.

Figure 5 illustrates two PCI bus read transactions. In Figure 5, data is transferred on clock edges that are labeled Data; wait cycles are inserted at clock edges labeled Wait.

Figure 5. PCI bus read transactions



During the first transaction, the first data phase finishes in the minimum time possible for the first data phase of a read transaction. The second data phase has one wait cycle because PciTRdy\_L is negated. The third data phase has one wait cycle because PciIRdy\_L is negated during clock cycle 6.

The initiator knows during clock cycle 6 that the next data phase is the last one of the transaction, but PciFrame\_L remains asserted because the initiator is not ready to complete the last data phase. Once the initiator is ready to complete the last data phase, it asserts PciIRdy\_L and negates PciFrame\_L (as shown in cycle 7 of Figure 5). In cycle 6, the target happens to be ready to complete the last data phase. Thus PciTRdy\_L remains asserted through cycles 6 and 7, and the last data transfer finishes at the clock edge at the end of cycle 7.

The target must assert the control lines for one clock cycle, cycle 8, before the turnaround cycle, which would be cycle 9. This happens to correspond with cycle 1 of the next transaction.

The second read transaction is different from the first because the initiator was not able to complete the first data phase until clock cycle 4, but the initiator is ready to complete the last data phase immediately.

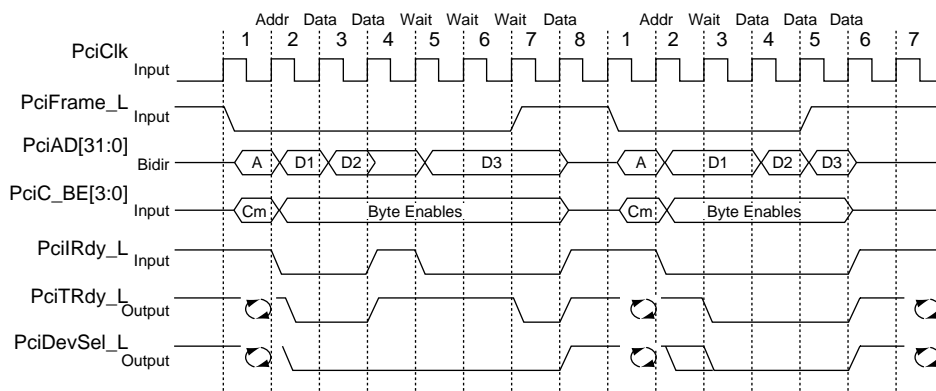
The target generates even parity one clock cycle after the transfer of each data phase. The parity must include the PciC\_BE[3:0] signals, even though they are inputs.



## 2.8.3 Write Transactions

Figure 6 shows two write transactions on the PCI bus. The transactions start when PciFrame\_L is asserted for the first time during clock cycle 1. This indicates the address phase, which lasts for only one clock cycle. A write transaction is similar to a read transaction except that no turnaround cycle is required for the PciAD lines. Therefore, the first data phase may finish at the end of clock cycle 2 (as it does for the first transaction in Figure 5). Beyond the first data phase, transfers follow the same protocol for both reads and writes.

Figure 6. PCI bus slave write transactions



In Figure 6, the first and second data phases of the first transaction finish with no wait cycles. The third data phase, however, has three wait cycles inserted by the target. A wait cycle was requested by both the target and the initiator for cycle 4. As it was for read cycles, PciIRdy\_L must be asserted when PciFrame\_L is negated by the initiator to indicate the last data phase. The last data phase will not be indicated by the master until it is ready to complete the data transfer. Even though the data may be delayed by the master until the end of the data phase, the byte enables must be valid throughout the data phase. The initiator sends a parity signal one clock cycle after the data transfer of each data phase. The MIO chip ignores the parity signal.

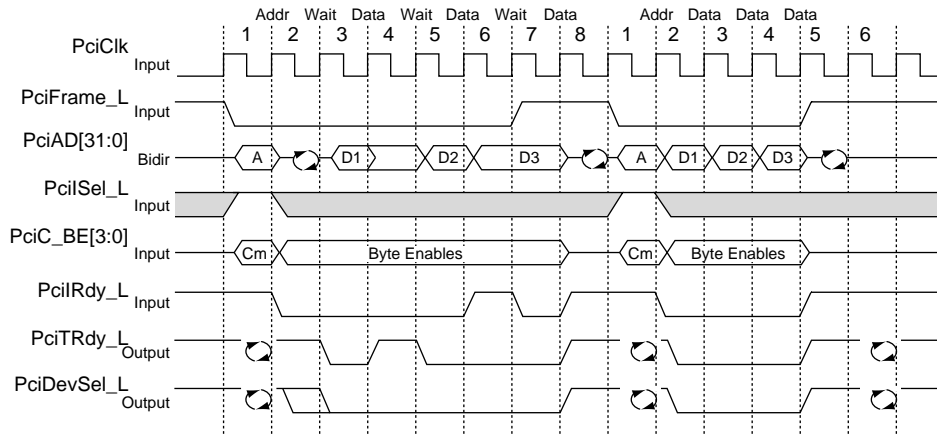
## 2.8.4 Configuration Transactions

The MIO chip accepts PCI bus configuration cycles as a PCI bus slave. Figure 7 shows an example of a PCI bus configuration cycle that the MIO chip accepts. Configuration cycles are identical to normal memory transfers, except

that the transaction decoding is provided externally during the assertion of the PciIDSel signal. PciIDSel is active only during the address phase of the PCI bus configuration cycle. The MIO chip drives PciDevSel\_L active as before.

The configuration registers are addressed using the PciAD[7:2] and PciC\_BE[3:0] signals. This provides the MIO chip with a 256-byte region of configuration space.

Figure 7. Configuration transaction



## 2.8.5 Configuration Registers

The MIO chip uses PCI configuration cycles to configure the PCI bus interface. During system initialization, the CPU polls the PCI configuration space to determine which devices are present on the PCI bus. It then initializes the PCI bus header and base registers for each PCI bus device. Table 12 shows the PCI bus configuration registers.

Table 12. PciHeader registers

PciHeader	[31:24]	[23:16]	[15:8]	[7:0]
Word[0]	DeviceID		VendorID	
Word[1]	Status		Command	
Word[2]	Class code			Revision ID
Word[3]	BIST	Reserved	Latency Timer	Cache Line Size

Table 13 shows the values that appear in the MIO PciHeader registers.

Table 13. PciHeader register values

PciHeader	[31:24]	[23:16]	[15:8]	[7:0]
Word[0]	0x000E		0x106B	
Word[1]	0bn0nn n00n 0000 0000		0b0000 0000 0n0n 0nn0	
Word[2]	0xFF0000			0x00
Word[3]	0x00	0x00	0x40	0x08

The four words in the PciHeader provide device selection information. PCI bus devices are required to implement all of these read-only fields.

### 2.8.5.1 Device Identification

Table 14 shows the three bit fields in the PciHeader that contain identification information. Word[0] (offset 0x00) has the DeviceID as the upper two bytes and the VendorID as the lower two bytes. The least significant byte of word[3] (offset 0x08) contains the RevisionID. The current values of these IDs are given in Table 14.

Table 14. Device identification bit-field assignments

Name	Purpose	Description
DeviceID	Device identifier	This field identifies the vendor's particular device and is allocated by the vendor. MIO's DeviceID field value is allocated as 0x0007.
Vendor ID	Vendor identifier	This field identifies the manufacturer of the device. Valid vendor identifiers are allocated by a central authority to ensure uniqueness. MIO's VendorID field value is allocated as 0x106B.
Revision ID	Revision identifier	This field specifies a device specific revision number. MIO's RevisionID field value is currently 0x0000.

### 2.8.5.2 Device Control

The command register (offset 0x02) provides coarse control of the device's ability to generate and respond to PCI bus cycles. Table 15 shows the command

register bit assignments. At power-up, this register contains zeros in all bit positions. The MIO chip does not support the I/O Enable, Special Enable, and Palette Snoop bits; it does support the Memory, Master, Postable, and Parity bits. Bits that are not supported always return 0.

Table 15. Device control bit-field assignments

Bit	Name	Description
0	I/O Enable	Controls device's response to I/O space accesses. A 0 disables the device response to I/O space accesses. A 1 allows the device to respond to I/O space accesses. This field always returns 0.
1	Memory Enable	Controls device's response to memory space accesses. A 0 disables the device response to memory space accesses. A 1 allows the device to respond to memory space accesses.
2	Master Enable	Controls device's ability to act as a master on the PCI bus. A 0 blocks the device from generating PCI bus transactions. A 1 allows the device to operate as a PCI bus master.
3	Special Enable	Controls the device's action on Special Cycle operations. If 0, the device ignores all Special Cycle operations. If 1, the device monitors Special Cycle operations. The MIO chip ignores Special Cycle operations, so this bit is always 0.
4	Postable Enable	Controls device's ability to issue Postable Memory Write commands. If 0, the device will use Memory Write commands. If 1, the device will use Postable Memory Write commands.
5	Palette Snoop	Controls how VGA-compatible devices handle accesses to palette registers. The MIO chip is not VGA-compatible, so this bit is always 0.
6	Parity Enable	Controls device's response to parity errors. The MIO chip ignores parity errors when this bit is 0.
15:7	Reserved	Reserved for future use.

### 2.8.5.3 Device Status

The status register is used to record status information for PCI bus-related events. Read actions return the current value of the register. Write actions allow bits to be reset but not set. A bit is reset when the register is written and the data value in the corresponding bit location is 1.

Table 16 shows the status register bit-field assignments. At power-up, this register is reset to 0. The MIO chip does not support all of the device status register bits. Bits that are not supported always return 0.

Table 16. Device status bit-field assignments

Bit	Name	Description
7:0	Reserved	Reserved for future use.
8	Data Parity Error	Set if a parity error occurs when the MIO chip is the master and the Parity Enable bit is set in the command register.
10:9	DevSel Timer	These bits encode the response timing for PciDevSel_L. They are defined as: 0x0 = 1 cycle, 0x1 = 2 cycles, 0x2 = 3 cycles, and 0x3 = reserved. The MIO chip will always respond in one cycle, so this field is read-only with a value of 0.
11	Signaled Target Abort	This bit is set by a target device whenever it terminates a transaction with a target abort. The MIO chip asserts a target abort when it receives a transaction longer than the cache line size.
12	Received Target Abort	This bit is set by a master device whenever a transaction is terminated with a target abort.
13	Received Master Abort	This bit is set by a master device whenever it terminates a transaction with master abort. The MIO chip asserts a master abort when a transaction response exceeds the time allocated in the latency timer field.
14	Signaled System Error	This bit is set whenever a device asserts PciSerr_L. The MIO chip does not support the PciSerr_L signal, so this bit is read-only with a value of 0.
15	Detected Parity Error	This bit is set whenever a device detects a parity error. Parity Enable, bit 6 of the command register, has no effect on this bit.

### 2.8.5.4 Miscellaneous Functions

The Cache Line Size register specifies the cache line size in units of 32-bit words. This register is read-only with a value of 0x08.

The Latency Timer register specifies (in units of PCI bus clocks) the value of the latency timer that the MIO chip uses when it is a PCI bus master. At power-on, this register has a value of 0x40.

The BIST register is an optional register for devices supporting Built-In-Self-Test (BIST). The MIO chip does not currently support BIST. This register is read-only and returns a value of 0x00.

### 2.8.6 Base Registers

The PCI bus base registers specify the address range and location of PCI bus devices. They are set at system initialization. The MIO chip supports one PCI

bus base register for memory space accesses, as shown in Table 17. PciBase[0] allows the MIO chip to respond to a 512 KB range of memory addresses by storing the top 13 bits of the 32-bit address value and returning 0 in the bottom 19 bits. The 0 in the least significant bit field indicates that this register is a memory-space base register, instead of an I/O-space base register.

Table 17. PciBase register

[31:19]	[18:0]
Memory-space base	0

## 2.8.7 Interface States

The PCI bus interface implements the standard slave and master state machine definitions specified in Appendix III of the PCI Specification, version 2.0. This specification is listed under “PCI Special Interest Group” on page 213.

## 2.8.8 PCI Arbitration

The MIO arbiter has the following features:

- one high-priority grant
- five round-robin grants
- deadlock prevention for broken PCI masters
- request time-out of 16 idle bus cycles
- detection of Stop without TRdy (= Retry) to back off HIGnt
- logic to disallow request parking (master holding request and repeatedly performing cycles)
- critical arbitration (when a high-priority IO controller needs the bus, the arbiter will park on HIGnt so no other master can affect the transfer; timing is critical, so high-speed serial bitstreams will not drop bits)

The round-robin HoldGnt machine is implemented as a parallel case statement for timing and other purposes. The round-robin algorithm includes a prioritized re-entry scheme. The arbiter returns to the same round-robin state that it exited when a high-priority request is detected.

The time-out counter is incremented on idle bus cycles. When a round-robin

master or high-priority master has a grant, the counter is held. The counter is cleared when the round-robin master takes the bus or when the arbiter's Parking bit is set.

The Parking bit signals that the current master is still asserting a request. It masks the master's request to allow the round-robin to progress. The Parking bit is also set when the timer expires.

The PCI arbiter in the MIO chip can be disabled by setting the ARB\_BYPASS bit in the Feature Control register (see Section 2.7.3). This lets more than one MIO chip exist in a system, for example. When ARB\_BYPASS is set, the the MIO chip's REQA\_L output is multiplexed onto the GNTA\_L line, the MIO chip's GNT\_IN\_L input is multiplexed onto the REQA\_L line, and the ArbCritical output is multiplexed onto the GNTB\_L line.

**Note:** Be careful when using the the MIO chip's arbiter as a PCI bus arbiter. To enable the MIO bus arbiter, you must clear the ARB\_BYPASS bit in the MIO Feature Control register (see Section 2.7.3). To do this, you must pull down the GNT\_L signal of the PCI host bridge and pull up the GNT\_L signals of all other PCI devices, including the MIO chip. This prevents the PCI bus lines from floating and lets the PHB turn on the MIO arbiter.

## 2.9 DBDMA Operation

The MIO chip uses the DBDMA architecture, a scatter-gather scheme based on memory-resident data structures called *descriptors*. The data structures describe the data transfers to be performed. The MIO chip supports only the base DBDMA architecture. The following optional features are not implemented:

- **64-bit addressing.** The DBDMA CommandPtrHi and DataPtrHi registers are not implemented.
- **Memory to memory transfers.** The OffsetHi and OffsetLo registers are not implemented.
- **Multiple transfer modes.** Memory addresses always increment, the block transfer size is always system-defined, and transfers are always coherent.
- **Event and management channels.** Device registers and interrupts are used to implement event and general management functions.

### 2.9.1 DBDMA Channel Registers

The MIO chip implements the registers described in this section for each of its DBDMA channels.

See “Memory Map and Register Addressing” beginning on page 13 for details about the addressing of each channel.

### 2.9.1.1 ChannelControl

The ChannelControl register contains bits that control the operational state of the channel, as well as general status bits [s0:s7]. Its format is shown in Table 18. The fields shown in Table 18 are described in Table 19.

Table 18. ChannelControl register format

31:16	15:0
mask	data

Table 19. ChannelControl register description

Field	Bits	Description
mask	16	The ChannelControl.mask field selects which of the lower 16 bits can be modified by a write action. Bits in the lower half of ChannelControl can be written only if the corresponding bit in ChannelControl.mask are set.
data	16	The value in this field is conditionally written to the ChannelStatus register, based on the value of the ChannelControl.mask field.

Bits in the ChannelControl register’s data field are read-only if their corresponding mask bits are set. Bits written to the data field are routed to the ChannelStatus register. Certain bits in the ChannelStatus register cannot be written or should only be written to 1 or 0. See the next section for full details.

### 2.9.1.2 ChannelStatus

The read-only ChannelStatus register provides access to the channel’s internal status bits. The ChannelControl register bits are also visible through the ChannelStatus register. Write actions to this register are ignored. The ChannelStatus register format is shown in Table 20. The register’s bit assignments are described in Table 21.

The Run and Pause bits are control bits that are set and cleared by software. The Flush and Wake bits are command bits that are set by software and cleared by hardware once the action has been performed. The Dead, Active, and Bt bits are hardware status bits. The bits [s7:s0] can be used for general purpose status



and control. Their meaning is channel-specific, and they can be controlled by either hardware or software. They may affect conditional interrupt, branch, or wait actions.

Table 20. ChannelStatus register format

9	8	7	6	5	4	3	2	1	0
Reserved	Bt	s7	s6	s5	s4	s3	s2	s1	s0
<hr/>									
31:16	15	14	13	12	11	10			
Reserved	Run	Pause	Flush	Wake	Dead	Active			

Table 21. ChannelStatus register description

Field	Bits	Description
Run	1	ChannelStatus.Run is set to 1 by software to start execution by the channel. This should be done after the CommandPtr registers are initialized. Software clears this bit to 0 to abort the operation of the channel. When channel operation is aborted, data transfers are terminated, status is returned, and an interrupt is generated (if appropriate). Data stored temporarily in channel buffers may be lost.
Pause	1	ChannelStatus.Pause is set to 1 by software to suspend command processing. Hardware responds by suspending data transfers and command execution and by clearing the ChannelStatus.Active bit. Software must reset ChannelStatus.Pause to 0 to allow command processing to resume.
Flush	1	ChannelStatus.Flush can be set to 1 by software to force a channel that is executing an INPUT_MORE or INPUT_LAST command to update memory with any data that has been received from the device but has not yet been written to memory. When the memory write action is complete, hardware updates the xferStatus and resCount fields in the current memory-resident channel command. Then it clears the Flush bit. A partial status update is characterized by 1 in the Flush bit of the xferStatus field; a final status update is characterized by 0.
Wake	1	ChannelStatus.Wake is set to 1 by software to cause a channel that has gone idle to wake up, refetch the command pointed to by CommandPtr, and continue processing commands. The channel becomes idle after executing a STOP command. The STOP command does not increment the CommandPtr register. ChannelStatus.Wake is reset to 0 by hardware immediately after each command fetch.

Table 21. ChannelStatus register description (*continued*)

Field	Bits	Description
Dead	1	ChannelStatus.Dead is set to 1 by hardware when the channel halts execution because of a catastrophic event, such as a bus error. When this occurs, the channel stops immediately. Status is not written to memory and further commands are not executed. If a hard-wired interrupt signal is implemented, an interrupt is generated unconditionally. When hardware sets ChannelStatus.Dead, ChannelStatus.Active is deasserted. Hardware resets ChannelStatus.Dead to 0 after the Run bit is cleared by software.
Active	1	ChannelStatus.Active is set to 1 by hardware in response to software setting the Run bit to 1. ChannelStatus.Active is reset to 0 by hardware in response to software clearing the Run bit or setting the Pause bit. It is also reset to 0 after a STOP command is executed, or when hardware sets the Dead bit to 1.
Bt	2	ChannelStatus.Bt is set by hardware at the completion of the NOP, INPUT, and OUTPUT commands. It indicates whether a branch has been taken. Branching is governed by the Command.b field in the command descriptor, the ChannelStatus.s7..s0 bits, and the values in the mask and value fields of the BranchSelect register. The presence of these bits in the Command.xferStatus field lets software follow the actual channel program flow with minimal overhead.
s7..s0	4	Each DBDMA channel has up to eight general-purpose state bits that can be read through the ChannelStatus register (ChannelControl.s7 .. ChannelControl.s0). Some of these bits reflect hardware interface signals directly, such as end-of-packet or error, while others are controlled by software and may be written through the channel control register. The MIO chip's channels typically support only a small number of these bits. See the specific I/O module descriptions in Section 2.11 for details. These bits are tested at the completion of each command to determine if certain actions should be taken. The actions may include generation of a hard-wired interrupt signal, suspension of further command processing, and branching to a new location for further command fetches.

### 2.9.1.3 CommandPtr

The CommandPtr register shown in Table 22 specifies the address of the next command entry to be fetched. Since all channel commands are 16-byte aligned, the least-significant nybble of the CommandPtr register should always be written with zeros. The CommandPtr register can be read at any time, but writes to this register are only enabled when the ChannelStatus.Active bit is 0.

Table 22. CommandPtr register format

31:4	3:0
CommandPtr	Reserved

### 2.9.1.4 InterruptSelect Register

The InterruptSelect register, shown in Table 23, is used to generate an interrupt condition bit, which is tested at the completion of each command to determine if a hard-wired interrupt signal should be asserted. The testing of this bit is controlled by a 2-bit field within each channel command called the Command.i bits. For further information, see Section 2.9.2.1.3 on page 50.

Table 23. InterruptSelect register format

31:24	23:16	15:8	7:0
Reserved	Mask	Reserved	Value

The MIO chip generates the interrupt condition signal according to the following algorithm:

```
c = ((ChannelStatus.s7..s0 & InterruptSelect.mask)
    == (InterruptSelect.value & InterruptSelect.mask))
```

The InterruptSelect register is optional. Zero to eight bits may be implemented in both the mask and value fields, depending on the needs of the specific channel. If this register is not implemented, the channel returns 0 as the value of the interrupt condition bit. Additionally, channel commands cannot specify conditional interrupts if the InterruptSelect register is not implemented. See the I/O module descriptions in Section 2.11, beginning on page 61, for details of the implementation of this register for each channel.

### 2.9.1.5 BranchSelect Register

The BranchSelect register is used to generate a branch condition bit that is tested at the completion of most commands to determine if a branch should be taken. The bit is tested using a 2-bit field within the channel command called the Command.b bit. The BranchSelect register has the same bit format as the InterruptSelect register described in Section 2.9.1.4.

The MIO chip generates the branch condition signal according to the following algorithm:

```
c = ((ChannelStatus.s7..s0 & BranchSelect.mask)
    == (BranchSelect.value & BranchSelect.mask))
```

The BranchSelect register is optional. Zero to 8 bits may be implemented in both the mask and value fields depending on the needs of the specific channel.

If this register is not implemented, the channel returns 0 as the interrupt condition bit, and channel commands cannot specify conditional branching. See the I/O module descriptions in Section 2.11 for details of the implementation of this register for each channel.

### 2.9.1.6 WaitSelect Register

The WaitSelect register is used to generate a wait condition bit that is tested at the completion of each command to determine if command execution should be suspended. The testing of this bit is controlled by a 2-bit field within each channel command called the Command.w bits. The WaitSelect register has the same format as the InterruptSelect and BranchSelect registers.

The wait condition signal is generated according to the following algorithm:

```
c = ((ChannelStatus.s7..s0 & WaitSelect.mask)
     ==(WaitSelect.value & WaitSelect.mask))
```

The WaitSelect register is optional. Zero to 8 bits may be implemented in both the mask and value fields depending on the needs of the specific channel. If this register is not implemented, the channel returns 0 as the wait condition bit, and channel commands cannot specify conditional suspension of command execution. See the I/O module descriptions in Section 2.11 for details of the implementation of this register for each channel.

### 2.9.1.7 Summary of Channel Registers

The initial values, read values, and write effects for the channel registers are summarized in Table 24.

Table 24. Channel register summary

Register Name	Initial Value	Read Value	Write Effect
ChannelControl	n.a.	0b0000 0000	Update selected bits
ChannelStatus	0b0000 0000	ChannelStatus	Ignored
CommandPtr	Undefined	CommandPtr	Store if idle
InterruptSelect	Undefined	n.a.	Update selected bits
BranchSelect	Undefined	BranchSelect	Update selected bits
WaitSelect	Undefined	ByteCount	Update selected bits

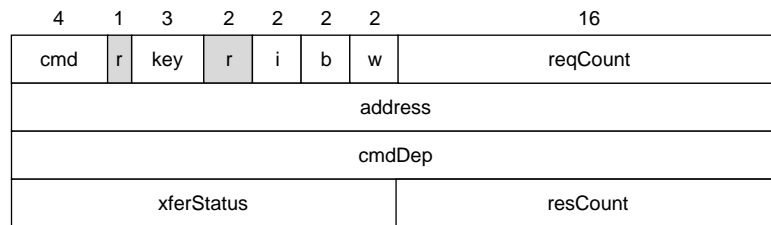
## 2.9.2 Commands

This section describes all the DBDMA channel commands that the MIO chip supports.

### 2.9.2.1 Command Formats

The general command format is comprised of 16 bits of opcode fields (including a 4-bit cmd field), a 16-bit reqCount field, a 32-bit address parameter, and a 32-bit cmdDep parameter, as shown in Figure 8.

Figure 8. Command entry format



The xferStatus and resCount fields are used by the channel to report status after a command finishes. Although the location and size of the cmdDep field is standardized for all commands, its interpretation is dependent on the Command.cmd field value.

#### 2.9.2.1.1 Command.cmd Field Values

The Command.cmd field specifies which type of data transfer will be performed. The commands are summarized and defined in Table 25 and discussed in greater detail in the next sections.

Table 25. Command.cmd field values

Value	Name	Description
0	OUTPUT_MORE	Transfer more memory to stream
1	OUTPUT_LAST	Transfer last memory to stream
2	INPUT_MORE	Transfer more stream to memory
3	INPUT_LAST	Transfer last stream to memory
4	STORE_QUAD	Store immediate 4-byte value

Table 25. Command.cmd field values (*continued*)

Value	Name	Description
5	LOAD_QUAD	Load immediate 4-byte value
6	NOP	No data transfer
7	STOP	Suspend command processing
8:15		Reserved

### 2.9.2.1.2 Command.key Value

The Command.key field is used to select which of the device access ports is used. The key values are specified in Table 26.

Table 26. Command.key field effects

Key Value	Name	Description	Comments
0	KEY_STREAM0	Default device stream	
1	KEY_STREAM1	Device-dependent stream	Not implemented
2	KEY_STREAM2	Device-dependent stream	Not implemented
3	KEY_STREAM3	Device-dependent stream	Not implemented
4		Reserved	
5	KEY_REGS	Channel register space	Not implemented (KEY_SYSTEM is used for accessing channel registers with STORE_QUAD commands)
6	KEY_SYSTEM	System memory-mapped space	Only valid in STORE_QUAD and LOAD_QUAD commands
7	KEY_DEVICE	Device memory-mapped space	Not implemented

The key field may specify alternate device streams (such as data and status) for the INPUT and OUTPUT commands. Only KEY\_STREAM0 is implemented in the MIO chip. For the LOAD\_QUAD and STORE\_QUAD commands, the key field specifies the address space where the immediate data is to be transferred. In the MIO implementation, only KEY\_SYSTEM is valid in this context.

Table 27 summarizes the operation of the key field in combination with each of the data transfer commands.

Table 27. Key field usage

Cmd	Key	Source	Destination
OUTPUT	KEY_STREAM $n$	SysMem (address)	Stream $n$
	KEY_SYSTEM	Not implemented	Not implemented
	KEY_DEVICE	Not implemented	Not implemented
	KEY_REGS	Not implemented	Not implemented
INPUT	KEY_STREAM $n$	Stream $n$	SysMem (address)
	KEY_SYSTEM	Not implemented	Not implemented
	KEY_DEVICE	Not implemented	Not implemented
	KEY_REGS	Not implemented	Not implemented
STORE_QUAD	KEY_STREAM $n$	n.a.	n.a.
	KEY_SYSTEM	data32 (from command)	SysMem (address)
	KEY_DEVICE	Not implemented	Not implemented
	KEY_REGS	Not implemented	Not implemented
LOAD_QUAD	KEY_STREAM $n$	n.a.	n.a.
	KEY_SYSTEM	SysMem(address)	data32 (in command)
	KEY_DEVICE	Not implemented	Not implemented
	KEY_REGS	Not implemented	Not implemented

### 2.9.2.1.3 Command.i Value

Upon completion of each command, a hard-wired interrupt may optionally be generated. Interrupt generation is controlled by the Command.i field in conjunction with an internal interrupt condition bit that is generated by the channel. The channel generates the interrupt condition based on the current values of the ChannelStatus.s7..s0 bits and the current values of the mask and data fields in the InterruptSelect register of each channel. The algorithm that generates the interrupt condition is the following:

```
c = ((ChannelStatus.s7..s0 & InterruptSelect.mask)
    == (InterruptSelect.value & InterruptSelect.mask))
```

Table 28 shows how the interrupt condition bit is used with the Command.i field to determine whether or not an interrupt will be generated.

Table 28. Command.i field effects

Command.i	Interrupt Condition
0	Never interrupt
1	Interrupt if the interrupt condition bit is true
2	Interrupt if the interrupt condition bit is false
3	Always interrupt

#### 2.9.2.1.4 Command.b Value

Upon completion of each data transfer command, the CommandPtr value is updated to point to either the next command or the target of a branch. Branching is controlled by the Command.b field in conjunction with an internal branch condition bit that is generated by the channel. The channel generates the branch condition based on the current values of the ChannelStatus.s7..s0 bits and the current values of the mask and data fields in the BranchSelect register of each channel. The algorithm that generates the branch condition is the following:

```
c = ((ChannelStatus.s7..s0 & BranchSelect.mask)
    == (BranchSelect.value & BranchSelect.mask))
```

Table 29 shows how the branch condition bit is used with the Command.b field to determine whether or not a branch will be taken.

Table 29. Command.b field effects

Command.b	Branch Condition
0	Never branch
1	Branch if the branch condition bit is true
2	Branch if the branch condition bit is false
3	Always branch



### 2.9.2.1.5 Command.w Value

Upon completion of each command, fetches of additional commands may be optionally suspended. This is controlled by the Command.w field in conjunction with an internal wait condition bit that is generated by the channel interface. The channel interface generates the wait condition based on the current values of the ChannelStatus.s7..s0 bits and the current mask and data values found in the WaitSelect register of each channel. The algorithm that generates the wait condition is the following:

```
c = ((ChannelStatus.s7..s0 & WaitSelect.mask)
     ==(WaitSelect.value & WaitSelect.mask))
```

Table 30 shows how the branch condition bit is used with the Command.w field to determine whether or not a branch will be taken.

Table 30. Command.w field effects

Command.w	Wait Condition
0	Never wait
1	Wait if the wait condition bit is true
2	Wait if the wait condition bit is false
3	Always wait

If the determination is that the channel should wait, the wait will occur after the data transfer finishes but before status is written back and an interrupt is generated.

### 2.9.2.1.6 Command.cmdDep Field Value

The meaning and format of the 32-bit Command.cmdDep field is dependent on the Command.cmd values. For the STORE\_QUAD and LOAD\_QUAD commands, it is the data value to be written or read. For the INPUT and OUTPUT commands, it is used to specify a conditional branch target. See the command descriptions in the next sections for more detailed information.

### 2.9.2.1.7 Command.xferStatus Value

At command completion, the current contents of the channel's ChannelStatus register are written to the Command.xferStatus field. When the Command.xferStatus value is written to memory, the bit corresponding to ChannelStatus.Active is always set to 1. This indicates that the corresponding command has been executed and that the xferStatus and resCount fields have been updated. Host

software should initialize the ChannelStatus.Active bit to 0 in the xferStatus field to use this feature.

The ChannelControl.Flush bit provides a mechanism for software to force buffered data to be written to memory and to force a status update prior to the completion of the command. Partial status reports are characterized by the Flush bit being set to 1 in the xferStatus field.

### 2.9.2.1.8 Command.resCount Value

At command completion, the 16-bit residual byte-count value is written to the Command.resCount field. This value is normally 0 but may be more if the device prematurely terminates the data transfer. The Command.resCount and Command.xferStatus fields are updated in an indivisible operation.

## 2.9.2.2 INPUT and OUTPUT Commands

The INPUT\_MORE, INPUT\_LAST, OUTPUT\_MORE, and OUTPUT\_LAST commands are used to transfer data between memory and the device stream. The OUTPUT\_MORE and OUTPUT\_LAST commands transfer data from system memory, typically into the attached device. The INPUT\_MORE and INPUT\_LAST commands transfer data, typically from the attached device, into memory. Data chaining is accomplished through the use of the MORE and LAST versions of these commands. The MORE commands indicate that the current buffer is not expected to complete a logical record (such as a network packet). The LAST commands indicate that the buffer is expected to complete a logical record. The reqCount field specifies the number of bytes to be transferred and the address field specifies a starting system memory address, as shown in Figure 9.

Figure 9. Format of INPUT and OUTPUT commands

4	1	3	2	2	2	2	16
cmd	r	key	r	i	b	w	reqCount
address							
branchAddress							
xferStatus						resCount	

The Command.key field is used to select which of the device access ports will be used.

An OUTPUT or INPUT command that performs a conventional transfer between system memory and an unaddressed device stream specifies a Com-

mand.key value of KEY\_STREAM0. Certain I/O modules may support additional stream identifiers. These identifiers provide access to other streams that may be supported by the device. Control and status information, for example, could be provided on an alternate stream. See the I/O module descriptions in Section 2.11, beginning on page 61, for details of support for multiple streams.

### 2.9.2.3 STORE\_QUAD Command

The STORE\_QUAD command stores a 32-bit immediate value into system memory space. The 32-bit data32 field specifies the data value. The 32-bit address field specifies the destination address, as shown in Figure 10.

Figure 10. STORE\_QUAD command fields



During the execution of a STORE\_QUAD command, the Command.key value specifies the destination address space for the immediate data. The MIO chip supports the KEY\_SYSTEM value, which indicates that the immediate data will be stored in the system memory space. The device and DBDMA channel registers can still be accessed via the system memory space, even though the KEY\_DEVICE and KEY\_REGS values are not supported.

The only valid entries for the reqCount field are 1, 2, and 4. Aligned transfers are the only type of transfers that are supported. If the count is 4, the two low-order address bits are assumed to be 0 and are ignored. The low-order address bit is assumed to be 0 if the count is 2. When the count is less than 4 bytes, the least significant bytes of data32 are transferred.

Illegal counts are mapped into legal counts as shown in Table 31, where *x* represents an ignored bit.

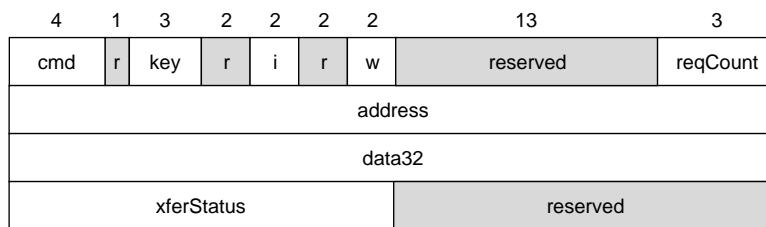
Table 31. Effective count values

3-bit reqCount	Effective count
0b00x	1
0b01x	2
0b1xx	4

### 2.9.2.4 LOAD\_QUAD Command

The LOAD\_QUAD command loads a 32-bit immediate value from memory. The 32-bit data32 field is the destination. The address field specifies the source address, as shown in Figure 11.

Figure 11. LOAD\_QUAD command fields



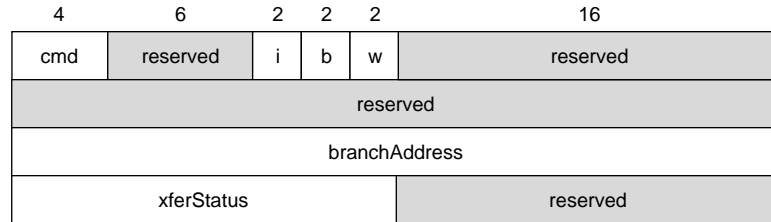
The LOAD\_QUAD and STORE\_QUAD commands handle the size and alignment of data transfers in the same manner.

During the execution of a LOAD\_QUAD command, the Command.key value specifies the source address space for the immediate data. The MIO chip supports the KEY\_SYSTEM value, which indicates that the immediate data will be loaded from the system memory space. The device and DBDMA channel registers can still be accessed via the system memory space even though the KEY\_DEVICE and KEY\_REGS values are not supported.

### 2.9.2.5 NOP Command

The NOP command uses standard mechanisms to specify interrupts, branches, or waits. It performs no data transfers. The NOP command structure is summarized in Figure 12.

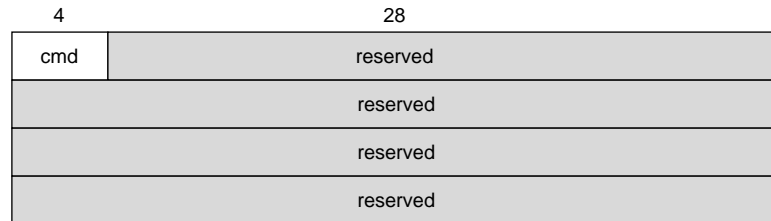
Figure 12. NOP command fields



### 2.9.2.6 STOP Command

The STOP command deactivates channel processing. It is placed at the end of a command list to indicate that there are currently no further commands to be processed. The only effects of the STOP command are to set the channel idle and clear the ChannelStatus.Active bit. The format of the STOP command is shown in Figure 13.

Figure 13. STOP command fields



Software can perform the following sequence to add additional commands to an active channel program:

1. Append the new commands to the existing channel program in memory.
2. Overwrite the STOP command with a NOP command (or the first new channel command).
3. Set the Wake bit in the ChannelControl register to 1. This lets the channel process the new commands even if it had already completed the old commands and gone idle.

## 2.10 Interrupt Processing

The MIO chip implements the Open PIC interrupt controller architecture and specification. For further information about Open PIC, see the IBM documentation cited in the bibliography.

The current version of Open PIC supports up to 20 interrupt sources and 2 processors. The interrupt architecture in the MIO chip also supports master/slave configurations by programming the Cascade Mode bit in the Open PIC Global Configuration Register 0.

Figures 14 and 15 show how the MIO chip can couple with external interrupt controllers, such as 8259 chips, in master and slave configurations. When the MIO chip is in interrupt slave mode, the CpuInt signal from the Open PIC cell is passed out through the SlotInt\_L pin to an external interrupt controller, such as an 8259 chip. In this mode, the interrupt model is fully compatible with the x86 architecture. When the MIO chip is in interrupt master mode, the Open PIC cell treats an 8259 chip as just another interrupt source.

Figure 14. 8259 master interrupt configuration

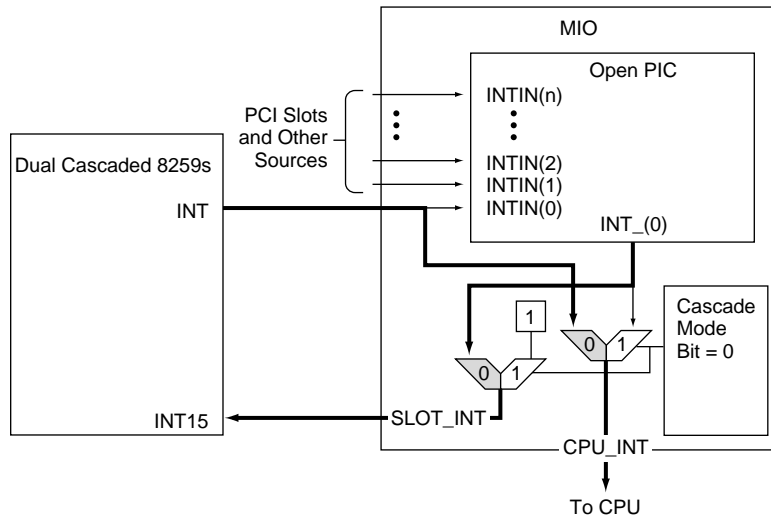
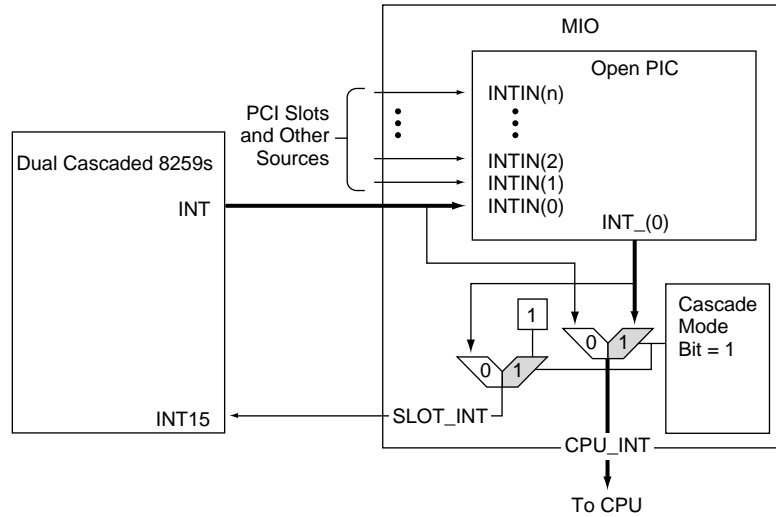


Figure 15. Open PIC master interrupt configuration



The default condition for the MIO chip's Open PIC cell is interrupt slave mode. It can be changed to interrupt master mode by first setting to 1 the Open PIC Cascade bit (bit 29 of the Open PIC Global Configuration register) and then setting to 1 the MpicIsMaster bit in the MIO Feature Control register (see Section 2.7.3). Table 32 shows the Open PIC operating modes in the MIO chip for all values of the Cascade and MpicIsMaster bits.

Table 32. MIO Open PIC modes

Mode	Cascade	MpicIsMaster	SlotInt	CpuInt0_L	SIOInt
Default	0	0	0	INT[0]_L	Normal
Not supported	0	1	0	1	Blocked into Open PIC
Interrupt slave	1	0	$\sim$ INT[0]_L	$\sim$ SIOInt	Blocked into Open PIC
Interrupt master	1	1	0	INT[0]_L	Normal

## 2.10.1 Interrupt Assignments

Table 33 shows the interrupt channel assignments in the MIO chip.

Table 33. Interrupt assignments

Interrupt	Source	Polarity	Sense
IRQ0	SIOInt	Programmable	Programmable
IRQ1	SCSI DMA	Read 0	Read 0
IRQ2	SCC Tx A DMA	Read 0	Read 0
IRQ3	SCC Rx A DMA	Read 0	Read 0
IRQ4	SCC Tx B DMA	Read 0	Read 0
IRQ5	SCC Rx B DMA	Read 0	Read 0
IRQ6	SCSI device	Read 0	Programmable
IRQ7	SCC A device	Read 0	Programmable
IRQ8	SCC B device	Read 0	Programmable
IRQ9	VIA	Read 0	Programmable
IRQ10	ADB	Read 0	Programmable
IRQ11	ADB_NMI	Read 0	Programmable
IRQ12	ExtInt1	Read 0	Programmable
IRQ13	ExtInt2	Read 0	Programmable
IRQ14	ExtInt3	Read 0	Programmable
IRQ15	ExtInt4	Read 0	Programmable
IRQ16	ExtInt5	Read 0	Programmable
IRQ17	ExtInt6	Read 0	Programmable
IRQ18	ExtInt7	Read 0	Programmable
IRQ19	Spare	Read 0	Programmable

Table 34 shows the interrupt signal modes for the channel assignments shown in Table 33.



Table 34. Interrupt modes

Interrupt	Source	Polarity	Sense	Mode	Condition
IRQ0	SIOInt	0	0	Negative edge	Default
		0	1	Negative level	
		1	0	Positive edge	
		1	0	Positive level	
IRQ1:5	Internal DMA			Leading edge	Default
IRQ6:11	Internal device			Leading edge	Default
				Active level	
IRQ12:19	External			Positive edge	Default
				Negative level	

## 2.10.2 Disabling the Open PIC Cell

The Open PIC cell in the MIO chip can be disabled by clearing to 0 the MPIC\_Enable bit in the Feature Control Register (see Section 2.7.3). When Open PIC is disabled, all internal interrupt sources in the MIO chip are multiplexed out through the CpuInt\_L[1:0], ExtInt\_L[7:1], and SIOInt lines, so the system designer can implement a different interrupt scheme using an external interrupt controller. In addition, the Reset\_CPU[1:0]\_L lines are forced high. Table 35 shows the multiplexing assignments of the MIO interrupt sources when the Open PIC cell is disabled.

Table 35. Interrupt assignments with Open PIC disabled

Pin	Interrupt Source
CpuInt0_L	Mesh DMA interrupt
CpuInt1_L	SCC Tx A DMA interrupt
SIOInt	SCC Rx A DMA interrupt
ExtInt1_L	SCC Tx B DMA interrupt
ExtInt2_L	SCC Rx B DMA interrupt

Table 35. Interrupt assignments with Open PIC disabled (*continued*)

Pin	Interrupt Source
ExtInt3_L	Mesh device interrupt
ExtInt4_L	SCC channel A device interrupt
ExtInt5_L	SCC channel B device interrupt
ExtInt6_L	VIA device interrupt
ExtInt7_L	ADB device interrupt
SlotInt	ADB_NMI device interrupt

## 2.11 I/O Modules

This section discusses the cells in the MIO chip that support traditional Macintosh I/O features. These features include:

- VIA interrupt service
- ADB bus control
- SCC serial port control
- MESH SCSI bus control

For background information about the implementation of these features in Power Macintosh computers, see Section 1.3. For information about their use in the CHRP architecture, see *PowerPC Microprocessor Common Hardware Reference Platform: I/O Reference*. This document is listed in the bibliography.

### 2.11.1 VIA Support

The existing Macintosh Versatile Interface Adapter (VIA) interrupt service is supported by a VIA I/O module in the MIO chip. For further information about VIA, see *Technical Introduction to the Macintosh Family*, second edition.

**Note:** The MIO chip supports only the VIA port A. The VIA port B lines are unused, and if port B is programmed as an input all its bits will read 0.

Table 36 shows how the MIO chip processes the signals that support VIA port A interrupts.

Table 36. VIA port A usage

Signal	Function	I/O	Description
PA0	Unused	-	(Historically, used as BURNIN pin.)
PA1	Unused	-	(Historically, used as SNDVOL1 output.)
PA2	Unused	-	(Historically, used as SNDVOL2 output.)
PA3	Sync Modem	Output	Modem clock select: 1: Select the external serial clock to drive the SCC's /RTxCA pin 0: Select the C3.864M clock to drive the SCC cell.
PA4	En WaitReqA	Output	Lets the WaitReq_L signal from port A of the SCC appear on the PA7 input pin.
PA5	Unused	-	(Historically used as Head Select output to floppy drive.)
PA6	En WaitReqB	Output	Lets the WaitReq_L signal from port B of the SCC appear on the PA7 input pin.
PA7	SCC WREQ	Input	Reflects the state of the Wait/Request pins from the SCC.
CA1	Unused	-	(Historically, used for 60.15 Hz ticks.)
CA2	Unused	-	(Historically, used for 1-second interrupts.)

The following notes apply to Table 36:

- The port A WaitReq\_L signal is called SccRxReqA\_L and the port B WaitReq\_L signal is called SccRxReqB\_L in the external pin list (Table 45, beginning on page 72).
- The logic algorithm for the PA7 signal is:  

$$PA7 = \sim(PA4 \ \& \ \sim WaitReqA\_L \ | \ PA6 \ \& \ \sim WaitReqB\_L).$$

## 2.11.2 ADB Support

The MIO chip contains a hardware controller module for the Apple Desktop Bus (ADB), a low-speed bus for input devices such as keyboards, mouse devices, and tablets. The controller is designed to relieve the microprocessor from needing to perform ADB control functions.

The MIO chip's ADB controller can replace other ADB implementations used in earlier Macintosh systems. It performs all ADB master functions, including autopolling and SRQ autopolling with error recovery and keyboard-invoked Reset and NMI actions.

The MIO chip's ADB implementation complies fully with the specifications in Chapter 3, "Apple Desktop Bus Controller," beginning on page 81.

### 2.11.3 SCC Support

The Macintosh Serial Communications Controller (SCC) serial I/O service is provided by an SCC I/O module in the MIO chip. The external interface for serial ports supported by the MIO chip's SCC module is defined in Chapter 6, "Macintosh Serial Port," beginning on page 181

The SCC I/O Module in the MIO chip controls an embedded 85C30 SCC cell. In Macintosh computers it drives a LocalTalk port and a generic serial port. The MIO chip assigns four unidirectional DBDMA channels for the SCC. This allows high-speed, full duplex operation of one or more SCC ports.

There are two basic types of SCC channel commands: buffer commands and register commands. However, all of the SCC control/status registers can also be accessed by direct memory-mapped addresses. For buffer commands, the SCC I/O module takes the memory data buffer pointed to by the channel command and either fills or empties it, depending on whether the command is a receive or a transmit. Register commands are typically used to configure the SCC.

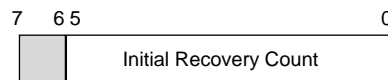
#### 2.11.3.1 SCC Registers

All SCC registers are memory mapped. Note that the following registers can be accessed by STORE\_QUAD and LOAD\_QUAD commands in addition to processor memory-mapped accesses.

##### 2.11.3.1.1 Initial Recovery Count Register

The SCC has timing processes that require a read/write Initial Recovery Count register in the SCC I/O module. This register is shown in Figure 16. The value loaded into the 6-bit Initial Recovery Count register (IRC) defines the delay between accesses to the SCC. The I/O module logic loads this count value into an internal timer when an SCC access occurs and starts counting down. The SCC I/O module holds off a second access to the SCC until the timer has reached 0.

Figure 16. Initial Recovery Count register



PCLK is the clock to the SCC cell. The recovery time for an SCC access is 3.5 PCLK cycles in EREQ mode and 4 PCLK cycles in non-EREQ mode (the default). Thus once an SCC read or write is generated, subsequent reads or writes cannot occur for 3.5 PCLK cycles. If the SCC is set in EREQ mode by software, its TxDMAReq signal will be deasserted within 70 ns after an SCC write if the transmit buffer is full. This 70 ns delay is less than the cycle time of the SCC hardware, so the TxDMAReq signal is deasserted before the SCC logic can see it as an extra (and incorrect) request. TxDMAReq is a signal internal to the MIO chip that communicates between the SCC cell and the I/O circuitry that controls it.

Upon reset, however, the SCC is not in EREQ mode. In this case the delay is 4 PCLK cycles. This could cause incorrect behavior by the SCC logic if the PCLK cycle time is long enough. Table 37 presents sample IRC settings.

Table 37. Sample IRC setting

PCLK Rate	3.5 PCLKs	4 PCLKs
15.6672 MHz	Set IRC to 5	Set IRC to 8 (this is the default value)

### 2.11.3.1.2 Registers in the SCC Cell

The system memory map for the SCC cell registers has two address mappings: a compatibility mapping that corresponds to the traditional Macintosh address scheme, and a MacRISC mapping that is used by CHRP specification Power Macintosh computers. Refer to Section 2.6, “Memory Map and Register Addressing,” on page 13, for a complete description of the SCC cell’s register address mapping.

Accesses to SCC command and status registers require a two-step process:

1. The register address must be written to the appropriate command register.
2. The data read or write must be performed to the command register.

Although transmit and receive data accesses only require one read or write, they will normally be performed via DBDMA. The following is an example of SCC register accessing:

```

SccAPortCmd    EQU    0x85003020
SccBPortCmd    EQU    0x85003000
SccAPortData   EQU    0x85003030
SccBPortData   EQU    0x85003010

```

```

// write 0xFF to port A, register 3 in SCC
write SccAPortCmd, 0x03    // write address
write SccAPortCmd, 0xFF    // write data

// read port B, register 5 in SCC
write SccBPortCmd, 0x05    // write address
write SccBPortCmd, myData  // read data

// write transmit data 0xAA to port B FIFO
write SccBPortData, 0xAA

// read receive data from port A receive FIFO
read SccAPortData, myRxData

```

### 2.11.3.2 LocalTalk Support

Apple LocalTalk is a family of hardware interconnection products for local area networking. LocalTalk support is provided by a section of logic in the MIO chip called the LTPC logic module. The access registers for this module are memory-mapped into the SCC cell's address space and explained in detail in the next sections. For information about LocalTalk, see *Technical Introduction to the Macintosh Family*, second edition, listed in the bibliography.

#### 2.11.3.2.1 StartA and StartB Registers

One 1-bit read/write Start register, of the form shown in Figure 17, serves each SCC transmit channel. These registers are cleared on reset. When software sets one of these registers to 1, the LTPC logic module begins monitoring the TxData line, looking for an abort sequence of 16 ones in a row. RTS\_ is driven high when the sequence is detected; this terminates the abort sequence. This register must be set to 0, and then 1 again, to reinitialize it for the next packet.

Figure 17. StartA and StartB registers



#### 2.11.3.2.2 DetectAB Register

The 2-bit read-only register shown in Figure 18 consists of separate DetectA and DetectB bits. They indicate whether the LTPC has detected the abort sequence for the corresponding transmit channels A and B. Each bit is cleared on reset and is set when the LTPC logic module has been armed (by setting the

Start register) and has then detected the abort sequence. It stays set until the corresponding Start register is cleared by software.

Figure 18. DetectAB register



### 2.11.3.2.3 Sending Packets Using the LTPC

The following steps list the sequence that should be followed to send packets using the LTPC module.

1. Disable interrupts through software.
2. Program the SCC to flag idle.
3. Ensure that the Start register is cleared. This will cause RTS\_ to be passed through the LTPC without any modification.
4. Turn on the RTS\_ bit in the SCC.
5. Wait for more than 1 bit time to guarantee generating an edge on the line.
6. Turn off the RTS\_ bit in the SCC.
7. Wait 1.5 bit times to generate the missing clock.
8. Enable the transmitter and turn on the RTS\_ bit in the SCC.
9. Program the SCC to mark idle.
10. Start the DBDMA transmit channel program.
11. Set the Start register. This will cause it to start monitoring the TxData line looking for the abort sequence. RTS\_ will remain in pass-through mode until the abort sequence has finished.
12. Software may now reenale interrupts.
13. Poll the DetectAB register to determine when the abort sequence has finished. When the LTPC has detected 16 ones in a row it will drive the RTS\_ line high, terminating the abort sequence.
14. Turn off the RTS\_ bit in the SCC.
15. Reset the missing clocks flag.
16. Clear the Start register. This will cause RTS\_ to be passed through without any modification.

### 2.11.3.3 Channel Status Register Usage

This section details how the channel status registers are used.

#### 2.11.3.3.1 Transmit A, B Channels

The general-purpose status bits (ChannelStatus.s7..s0) are allocated as indicated in Table 38.

Table 38. SCC transmit channel status bits

Bit	Meaning
s7..s6	Not implemented
s5	LTPC detect
s4..s1	Not implemented
s0	Wait (externally controlled)

#### 2.11.3.3.2 Receive A, B Channels

The general-purpose status bits (ChannelStatus.s7..s0) are allocated as indicated in Table 39.

Table 39. SCC receive channel status bits

Bit	Meaning
s7..s1	Not implemented
s0	Wait (externally controlled)

### 2.11.3.4 Conditional Interrupt, Branch, and Wait Generation

This section details how Interrupt, Branch, and Wait conditions are generated.

#### 2.11.3.4.1 Transmit A, B Channels

Only bits s0 and s5 may be used to generate the Interrupt, Branch, and Wait conditions. These are the only bits implemented in the InterruptSelect, BranchSelect, and WaitSelect registers for the Transmit A and B channels.



### 2.11.3.4.2 Receive A, B Channels

Only bit s0 may be used to generate the Interrupt, Branch, and Wait conditions. This is the only bit implemented in the InterruptSelect, BranchSelect, and WaitSelect registers for the Receive A and B channels.

### 2.11.3.5 Channel Program Example

The following sample program, written in pseudo-code, transmits a data buffer out of the SCC's A port. The execution of this program is traced in Table 40.

```

SccIRC      EQU    0x85003080
SccAPortCmd EQU    0x85003020
BufferA     EQU    0x10000000

STORE_QUADSccIRC      // Assume 33 MHz system clock, 3.672 MHz PCLK
STORE_QUADSccAPortCmd // Configure SCC port A, address cycle
STORE_QUADSccAPortCmd // Configure SCC port A, data cycle

// Configure remainder of SCC registers

STORE_QUADSccAPortCmd // Configure SCC port A, data cycle
OUTPUT_LASTBUFFER_B
STOP

```

Table 40. SCC channel program trace

Step	Command	Key	i	b	w	reqCount	Address	cmdDep
1	STORE_QUAD	System	Never	Never	Never	4	SccIRC	36
2	STORE_QUAD	System	Never	Never	Never	4	SccAPortCmd	reg. address
3	STORE_QUAD	System	Never	Never	Never	4	SccAPortCmd	reg. data
4								
5	STORE_QUAD	System	Never	Never	Never	4	SccAPortCmd	reg. data
6	OUTPUT_LAST	Stream0	Never	Never	Never	2K	BUFFER_B	
7	STOP							

## 2.11.4 SCSI Support

The Macintosh SCSI bus for external devices such as hard disk drives is provided by a MESH SCSI I/O module in the MIO chip. The Macintosh external interface for SCSI devices is defined in Chapter 4, “SCSI Support,” beginning on page 105.

The SCSI I/O module in the MIO chip is used to communicate between the DBDMA controller and the SCSI controller cell (MESH). There are two basic types of SCSI channel commands: DBDMA commands and register commands. All of the SCSI controller registers can also be accessed by direct memory-mapped addresses. For DBDMA commands, the SCSI cell takes the memory data buffer pointed to by the channel command and either fills or empties it, depending on whether the command is an input or an output command. The MIO chip assigns only one DBDMA channel for SCSI because the SCSI bus is half duplex.

The SCSI I/O module has three signals that provide status for the current command. These signals allow the DBDMA channel program to continue through a series of SCSI bus activities as long as there is no exception or error. This allows a normal SCSI transaction (arbitration through final BusFree, without Disconnect) to occur with only one interrupt—a significant improvement over the four interrupts required for the 53C96 chip. The MESH controller does not handle Disconnect and Reconnect automatically.

### 2.11.4.1 Channel Status Register Usage

The SCSI general-purpose status bits (ChannelStatus.s7..s0) are allocated as shown in Table 41.

Table 41. SCSI channel status bits

Bit	Meaning
s7	MESHCmdDone_L or NoConnect
s6	MESHException_L or NoConnect
s5	MESHError_L or DReq
s4..s1	Not implemented
s0	Wait (externally controlled)

The MESHCmdDone\_L bit signifies that the MESH device has completed the requested command. This bit is always set regardless of the success or failure of the command. The MESHException\_L bit indicates that the command did not finish successfully, but the reason it failed is most likely a difference in SCSI protocol (the target switched phased prematurely, arbitration failed, selection phase timed out, and so on). The MESHError\_L bit indicates more significant errors, such as synchronous overruns and underruns or parity errors. The ChannelStatus.s0 bit is set and cleared by software or STORE\_QUAD commands.

#### 2.11.4.2 Conditional Interrupt, Branch, and Wait Generation

The Interrupt, Branch, and Wait conditions can use any or all of the s7, s6, or s5 bits for conditional activation.

#### 2.11.4.3 SCSI Channel Commands

SCSI channel programs are made up of one or more channel commands. There are two types of channel commands for the SCSI module in the MIO chip: DBDMA commands and register commands. All DBDMA commands have the same functionality as described in the DBDMA architecture defined in Chapter 5. There is currently no difference between the INPUT\_MORE and INPUT\_LAST commands, nor is there any difference between the OUTPUT\_MORE and OUTPUT\_LAST commands.

##### 2.11.4.3.1 DBDMA Commands

The commands summarized in Table 42 are responsible for setting up data transfers between memory and a SCSI device.

Table 42. Field definitions for SCSI DBDMA commands

Name	Description
cmd	INPUT_MORE or INPUT_LAST for input from the SCSI bus. OUTPUT_MORE or OUTPUT_LAST for output to the SCSI bus.
key	Ignored; assumed to be KEY_STREAM0.
i	Standard usage conforming to DBDMA architecture. See Chapter 5.
b	Standard usage conforming to DBDMA architecture. See Chapter 5.

Table 42. Field definitions for SCSI DBDMA commands (*continued*)

Name	Description
w	Standard usage conforming to DBDMA architecture. See Chapter 5.
reqCount	Transfer size in bytes.
address	This address points to the start of the SCSI buffer in main memory.
xferStatus	Upon completion of the command, the ChannelStatus register is written here.
resCount	Upon completion of the command, the residual transfer count is written here.

### 2.11.4.3.2 Register Commands

The commands summarized in Table 43 are used to access the SCSI registers.

Table 43. Field definitions for SCSI register commands

Name	Description
cmd	STORE_QUAD and LOAD_QUAD commands that are used to access I/O module or SCSI controller registers.
key	Ignored; assumed to be KEY_SYSTEM.
i	Standard usage conforming to DBDMA architecture. See Chapter 5.
b	Standard usage conforming to DBDMA architecture. See Chapter 5.
w	Standard usage conforming to DBDMA architecture. See Chapter 5.
reqCount	Transfer size in bytes. Since all device registers are 1 byte wide, and they are separated by a stride of 16, this value should be set to 1.
address	System memory-mapped address of device register.
xferStatus	Upon completion of the command, the ChannelStatus register is written here.

## 2.12 Testing

The MIO chip includes an internal scan capability for testing. To initialize the MIO chip for board and device testing, the reset pulse (PciReset\_L) must go from high to low and back to high. The leading edge of the assertion of reset generates a pulse that resets the divide-by-2 function. This function generates 3.7 MHz, 15.6672 MHz, and 25 MHz clocks, which run during the remainder of the reset assertion. The multiplexed MIO test pins are shown in Table 44.

Table 44. MIO test pins

Functional Signal	Test Signal
CachePD2	ScanIn1
GntC_L	ScanOut1
CachePD3	ScanIn2
GntD_L	ScanOut2
SccGPIOA_L	ScanIn3
Reset_CPU0_L	ScanOut3
SccGPIOB_L	ScanIn4
Reset_CPU1_L	ScanOut4

## 2.13 Physical Package

The current version of the Mac I/O ASIC chip, available from several vendors, is supplied in a 160-pin PQFP package. This section describes its signal lines and pin assignments.

### 2.13.1 Signal Lines

The external signal lines for the current MIO package are listed in Table 45.

Table 45. MIO chip signal lines

Signal	Type	Pins	Description
<b>PCI Control Pins</b>			
PciClk	Input	1	Clock: 25–33 MHz. Used to synchronize transfers on the PCI bus and drive most of the internal logic.
PciAD[31:0]	Bidir	32	Multiplexed address and data. During a transaction's first cycle, these lines contain a physical address. During subsequent cycles, they contain data. Little-endian byte ordering is used; PciAD[7:0] define the LSB and PciAD[31:24] define the MSB.

Table 45. MIO chip signal lines (*continued*)

Signal	Type	Pins	Description
PciC_BE[3:0]	Bidir	4	Multiplexed bus command and byte enables. During the address phase of a transaction, PciC_BE[3:0] define the bus command. During the data phase, PciC_BE[3:0] are used as byte enables. The byte enables determine which byte lanes carry meaningful data. PciC_BE[0] applies to byte 0 (PciAD[7:0]), and PciC_BE[3] applies to byte 3.
PciPar	Bidir	1	Parity: even parity is generated across PciAD[31:0] and PciC_BE[3:0].
PciFrame_L	Bidir	1	Cycle frame: Indicates the beginning and duration of an access. PciFrame_L is asserted to indicate that a bus transaction is beginning. When PciFrame_L is deasserted the transaction is in its final data phase.
PciTRdy_L	Bidir	1	Target ready: Indicates that the target is able to complete the current data phase of the transaction. A data phase is completed on any clock cycle in which both PciTRdy_L and PciIRdy_L are asserted when sampled.
PciIRdy_L	Bidir	1	Initiator ready: Indicates that the initiator is able to complete the current data phase of the transaction. A data phase is completed on any clock cycle in which both PciIRdy_L and PciTRdy_L are asserted when sampled.
PciStop_L	Bidir	1	Stop: the current target is requesting that the master stop the current transaction
PciIDSel	Input	1	Initialization device select: used as a chip select in place of the upper 24 address lines during configuration read and write transactions.
PciDevSel_L	Bidir	1	Device select: As an output, indicates the driving device has decoded its address as the target of the current access. As an input, indicates whether any device on the bus has been selected.
PciReset_L	Input	1	Reset: initializes the PCI interface logic, DBDMA controller, and device interfaces.
PciPErr_L	Bidir	1	Parity error: reports data parity errors during all PCI transactions except Special cycles.
PciSErr_L	Bidir	1	System error: used for reporting address parity errors, data parity errors on special cycle commands, or any other system errors where the result is catastrophic.
<i>Total PCI control pins</i>		47	
<b>PCI Arbiter Pins</b>			
HOST_REQ_L	Input	1	Host request: bus request from PCI host bridge.
HOST_GNT_L	Output	1	Host grant: bus grant to PCI host bridge.
REQA_L/ GNT_IN_L	Input	1	REQA: bus request from PCI master. GNT_IN: grant to MIO if external arbiter is used.
GNTA_L/ REQ_OUT_L	Output	1	GNTA: bus grant to PCI master. REQ_OUT: request from MIO if external arbiter is used.
REQB_L	Input	1	REQB: bus request from PCI master.

Table 45. MIO chip signal lines (*continued*)

Signal	Type	Pins	Description
GNTB_L/ ArbCritical	Output	1	GNTB: bus grant to PCI master; ArbCritical: tells the PCI bus arbiter that the MIO chip is requesting bus tenure for a real-time device.
REQC_L	Input	1	REQC: bus request from PCI master.
GNTC_L/ ScanOut1	Output	1	GNTC: bus grant to PCI master or Scan chain output 1.
REQD_L	Input	1	REQD: bus request from PCI master.
GNTD_L/ ScanOut2	Output	1	GNTD: bus grant to PCI master or Scan chain output 2.
<i>Total PCI arbiter pins</i>		<i>10</i>	
<b>SCSI Signals</b>			
SCSI_DB[8:0]	Bidir	9	SCSI data bus plus parity (DB8)
SCSI_ATN_L	Bidir	1	SCSI control
SCSI_BSY_L	Bidir	1	SCSI control
SCSI_FACK_L	Output	1	SCSI control, fast ACK
SCSI_ACK_L	Input	1	SCSI control
SCSI_RST_L	Bidir	1	SCSI control
SCSI_MSG_L	Bidir	1	SCSI control
SCSI_SEL_L	Bidir	1	SCSI control
SCSI_CXD_L	Bidir	1	SCSI control
SCSI_FREQ_L	Bidir	1	SCSI control, fast REQ
SCSI_REQ_L	Input	1	SCSI control
SCSI_IXO_L	Bidir	1	SCSI control
<i>Total SCSI pins</i>		<i>20</i>	
<b>SCC Signals</b>			
TXDA	Output	1	Channel A: transmit data
TXDB	Output	1	Channel B: transmit data

Table 45. MIO chip signal lines (*continued*)

Signal	Type	Pins	Description
RTSA_L	Output	1	Channel A: RTS
RTSB_L	Output	1	Channel B: RTS
DTRA_L	Output	1	Channel A: DTR
DTRB_L	Output	1	Channel B: DTR
RXDA	Input	1	Channel A: receive data
RXDB	Input	1	Channel B: receive data
GPIOA_L/ ScanIn3	Input	1	Channel A: GPI or Scan chain input 3
GPIOB_L/ ScanIn4	Input	1	Channel B: GPI or Scan chain input 4
TRXCA	Input	1	Channel A: TRXC
TRXCB	Input	1	Channel B: TRXC
<i>Total SCC pins</i>		12	
<b>ADB Interfaces</b>			
ADB_IO	Bidir	1	Data pin for ADB
ADBReset_L	Output	1	Reset out generated by ADB
<i>Total ADB pins</i>		2	
<b>Interrupts</b>			
CpuInt0_L/ MeshDMA_L	Output	1	Interrupt output to CPU0 (when Open PIC is enabled) Internal MESH DMA interrupt (when Open PIC is disabled)
CpuInt1_L/ SCCTXA_DMA_L	Output	1	Interrupt output to CPU1 (when Open PIC is enabled) Internal SCC Tx A DMA interrupt (when Open PIC is disabled)
SIOInt/ SCCRXA_DMA_L	Bidir	1	SIO interrupt input (when Open PIC is enabled) Internal SCC Rx A DMA interrupt output (with Open PIC disabled)
ExtInt1_L/ SCCTXB_DMA_L	Bidir	1	External interrupt 1 input (when Open PIC is enabled) Internal SCC Tx B DMA interrupt output (with Open PIC disabled)
ExtInt2_L/ SCCRXB_DMA_L	Bidir	1	External interrupt 2 input (when Open PIC is enabled) Internal SCC Rx B DMA interrupt output (with Open PIC disabled)



Table 45. MIO chip signal lines (*continued*)

Signal	Type	Pins	Description
ExtInt3_L/ Mesh_Dev_L	Bidir	1	External interrupt 3 input (when Open PIC is enabled) Internal MESH device interrupt output (with Open PIC disabled)
ExtInt4_L/ SCCA_Dev_L	Bidir	1	External interrupt 4 input (when Open PIC is enabled) Internal SCC channel A device interrupt output (with Open PIC disabled)
ExtInt5_L/ SCCB_Dev_L	Bidir	1	External interrupt 5 input (when Open PIC is enabled) Internal SCC channel B device interrupt output (with Open PIC disabled)
ExtInt6_L/ VIA_Dev_L	Bidir	1	External interrupt 6 input (when Open PIC is enabled) Internal VIA device interrupt output (with Open PIC disabled)
ExtInt7_L/ ADB_Dev_L	Bidir	1	External interrupt 7 input (when Open PIC is enabled) Internal ADB device interrupt output (with Open PIC disabled)
SlotInt	Output	1	Slot interrupt to 8259 chip, pin Int15
RESET_CPU0_L/ ScanOut3	Output	1	Processor reset signal from Open PIC to processor 0 or Scan chain output 3
RESET_CPU1_L/ ScanOut4	Output	1	Processor reset signal from Open PIC to processor 1 or Scan chain output 4
<i>Total Interrupt pins</i>		<i>13</i>	
<b>Miscellaneous Signals</b>			
Cache_PD0/SCLK	Bidir	1	Cache presence detect bit 0 or SCLK to EEPROM
Cache_PD1/SDAT	Bidir	1	Cache presence detect bit 1 or SDAT to EEPROM
Cache_PD2/ ScanIn1	Input	1	Cache presence detect bit 2 or Scan chain 1 input
Cache_PD3/ ScanIn2	Input	1	Cache presence detect bit 3 or Scan chain 2 input
Clk31_3344MHz	Input	1	Clock: 31.3344 MHz
Clk50MHz	Input	1	Clock: 50 MHz; used for MESH.
CFW_L	Input	1	Clock Fail Warning. CFW_L is asserted when sleep mode is requested. The MIO chip requests ownership of the PCI bus and performs an orderly shut-down of its I/O.
CPU_ID3	Input	1	CPU ID bit 3.
CPU_ID[2:0]	Bidir	3	CPU ID bits [2:0]. Extended mode is supported on these three bits. Bit 3 is input only.
<i>Total miscellaneous pins</i>		<i>11</i>	

Table 45. MIO chip signal lines (*continued*)

Signal	Type	Pins	Description
<b>Test Interface</b>			
ScanClkIn	Input	1	Scan Clock
Test_Mode_L	Input	1	Test Mode Pin
<i>Total test pins</i>		2	
<b>Power and Ground</b>			
VSS		27	Ground
VCC		16	5 V pad
<i>Total power pins</i>		43	

## 2.13.2 Pin Assignments

Table 46 lists the pin assignments for the current MIO chip configuration.

Table 46. MIO chip pin assignments

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	PciAD(29)	41	PciAD(12)	81	SCSI_FREQ_L	121	ExtInt5_L
2	PciAD(28)	42	VCC	82	VCC	122	ExtInt6_L
3	VSS	43	PciAD(11)	83	SCSI_FACK_L	123	ExtInt7_L
4	PciAD(27)	44	PciAD(10)	84	VSS	124	CpuInt1_L
5	PciAD(26)	45	PciAD(9)	85	SCSI_REQ_L	125	VSS
6	PciAD(25)	46	VSS	86	SCSI_ACK_L	126	Clk31_3344MHz
7	VCC	47	PciAD(8)	87	VSS	127	VSS
8	PciAD(24)	48	PciC_BE(0)	88	SCSI_IXO_L	128	ADB_IO
9	PciC_BE(3)	49	PciAD(7)	89	SCSI_CXD_L	129	VCC
10	PciID_Sel	50	VCC	90	SCSI_MSG_L	130	ADBReset_L
11	VSS	51	PciAD(6)	91	SCSI_RST_L	131	CFW_L

Table 46. MIO chip pin assignments (*continued*)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
12	PciAD(23)	52	PciAD(5)	92	SCSI_BSY_L	132	ScanClkIn
13	PciAD(22)	53	PciAD(4)	93	SCSI_SEL_L	133	RESET_CPU0_L
14	VCC	54	VSS	94	SCSI_ATN_L	134	RESET_CPU1_L
15	PciAD(21)	55	PciAD(3)	95	VSS	135	VSS
16	PciAD(20)	56	PciAD(2)	96	TXDA	136	Cache_PD0
17	PciAD(19)	57	PciAD(1)	97	RTSA_L	137	Cache_PD1
18	VSS	58	VCC	98	DTRA_L	138	Cache_PD2
19	PciAD(18)	59	PciAD(0)	99	VCC	139	VCC
20	PciAD(17)	60	VSS	100	RXDA	140	Cache_PD3
21	VCC	61	CPU_ID(3)	101	GPIOA_L	141	Test_Mode_L
22	PciAD(16)	62	CPU_ID(2)	102	TRXCA	142	PciReset_L
23	PciC_BE(2)	63	CPU_ID(1)	103	VSS	143	VSS
24	PciFrame_L	64	CPU_ID(0)	104	TXDB	144	PciClk
25	VSS	65	VSS	105	RTSB_L	145	VSS
26	PciIRdy_L	66	Clk50MHz	106	DTRB_L	146	Host_Req_L
27	PciTRdy_L	67	VSS	107	VCC	147	Host_Gnt_L
28	PciDevSel_L	68	SCSI_DB(8)	108	RXDB	148	ReqA_L
29	VCC	69	SCSI_DB(7)	109	GPIOB_L	149	VCC
30	PciStop_L	70	VSS	110	TRXCB	150	GntA_L
31	PciPErr_L	71	SCSI_DB(6)	111	VSS	151	ReqB_L
32	PciSErr_L	72	SCSI_DB(5)	112	CpuInt0_L	152	GntB_L
33	VSS	73	VSS	113	Slot_Int	153	ReqC_L
34	PciPar	74	SCSI_DB(4)	114	SIOInt	154	VSS
35	VCC	75	SCSI_DB(3)	115	ExtInt1_L	155	GntC_L
36	PciC_BE(1)	76	VSS	116	VCC	156	ReqD_L

Table 46. MIO chip pin assignments (*continued*)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
37	PciAD(15)	77	SCSI_DB(2)	117	ExtInt2_L	157	GntD_L
38	VSS	78	SCSI_DB(1)	118	VSS	158	PciAD(31)
39	PciAD(14)	79	VSS	119	ExtInt3_L	159	VCC
40	PciAD(13)	80	SCSI_DB(0)	120	ExtInt4_L	160	PciAD(30)



# Apple Desktop Bus Controller

# 3

## 3.1 Overview

This chapter defines the hardware controller for the Apple Desktop Bus (ADB), a low-speed bus for input devices such as keyboards, pointing devices, and tablets. The controller is designed to relieve the microprocessor from needing to perform ADB control functions. For information about using the ADB in the CHRP architecture, see *PowerPC Microprocessor Common Hardware Reference Platform: I/O Reference*. This document is listed in the bibliography.

**Note:** An Apple-designed ADB controller is part of the chip described in Chapter 2, “The Mac I/O Chip,” beginning on page 7.

The ADB controller specified in this chapter can replace other ADB implementations used in earlier Macintosh systems. It performs all ADB master functions, including autopolling, SRQ autopolling with error recovery, and keyboard-invoked reset and nonmaskable interrupt (NMI) actions.

## 3.2 Electrical Interface

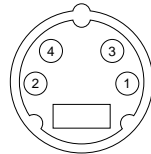
The following Apple documents, listed in the bibliography, define the electrical interface to the ADB and are incorporated in this specification:

- 062-0267-F      Specification, Apple Desktop Bus
- 062-2012-A      Engineering Specification, Macintosh Transceiver Interface, ADB

The external (user) connection for the ADB is a 4-pin mini-DIN type, as shown in Figure 19.

**Requirement 3-1:** The ADB implementation must conform to Apple specifications 062-0267-F and 062-2012-A.

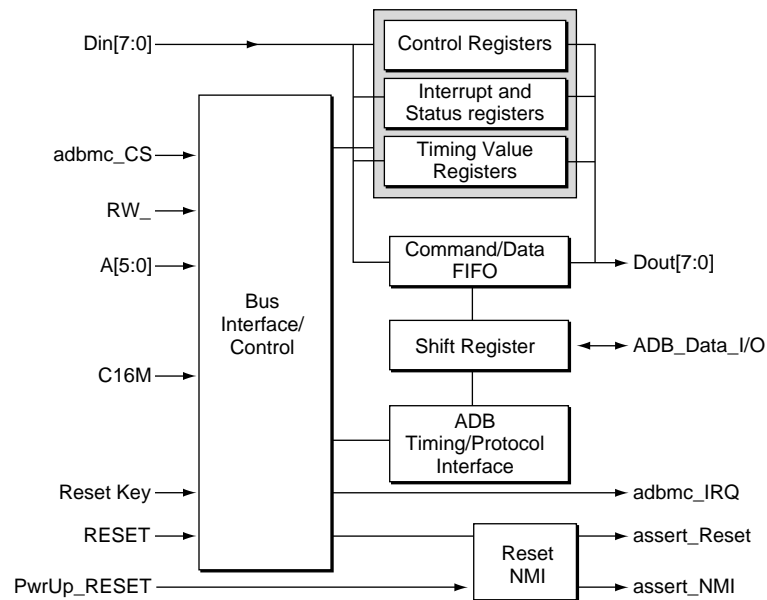
Figure 19. ADB connector



### 3.3 Architecture and Signals

The ADB controller contains the functional units shown in Figure 20.

Figure 20. ADB controller block diagram



The ADB controller uses the hardware input and output signal lines specified in Table 47.

Table 47. ADB controller signal lines

Name	Input/Output	Description
A[5:0]	Input	Address lines
Din[7:0]	Input	Data in
Dout[7:0]	Output	Data out
ADB_Data_I/O	Input/output	Serial data I/O
RW_	Input	Read/write control
adbmc_CS	Input	Chip select
adbmc_IRQ	Output	Interrupt request
assert_NMI	Output	Nonmaskable interrupt
Reset_Key	Input	Reset key input
RESET	Input	Hardware reset
PwrUp_RESET	Input	Power-up reset
assert_Reset	Output	Reset trigger
C16M	Input	16 MHz clock

**Requirement 3-2:** The ADB controller must support the signal lines listed in Table 47.

## 3.4 Communication With the System

The ADB controller and the host computer system (referred to here as *the system*) communicate with each other by means of a set of registers and a 9-byte command/data register file.

**Requirement 3-3:** The ADB controller must support the communication protocols described in Section 3.4.

### 3.4.1 Command/Data Register File

The ADB controller contains a 9-byte command/data register file (CDRF) that is used for sending commands and data from the system to the ADB and for



passing data from the ADB to the system.

When sending data from the system to the ADB controller, the CDRF is written with the command first, then from 0 to 8 bytes of data, starting with the most significant byte. When the system receives data from the ADB controller, the register file contains the command first, then from 0 to 8 bytes of data, starting with the most significant byte.

### 3.4.2 CDRF Arbitration

The CDRF queues data between the ADB and the system. Because only one register file is used, it is necessary to ensure that data collisions cannot occur. The algorithm defined in this section preserves the data integrity of the CDRF.

The System requests use of the CDRF and then waits until the ADB controller grants access to it. While the system is granted access, the ADB controller may not write data to the CDRF. While the system is not granted access it does not write data to the CDRF. The same arbitration protocol is used when the system must update a timing value register or a dynamic control register. Note that there is no hardware contained in the ADB controller to insure the data integrity of the CDRF.

When the system wants to send a command (or command and data) to the ADB controller, it requests access to the CDRF by setting the transfer access request (TAR) bit. The ADB controller interrupts the system when access is granted, provided that the transfer access grant interrupt source enable bit (TAG\_IS\_EN) is set. Alternately, the system may choose to poll the ADB controller to determine whether access has been granted.

The ADB controller is said to be *idle* when it is not communicating with the ADB (either performing an autopoll sequence or an explicit command) or when it is waiting for the next autopoll sequence to occur. When it is idle, the ADB controller checks the state of the TAR bit. If the ADB controller has written data to the CDRF from either an autopoll sequence or in response to an explicit command, it does not check the TAR bit; however, it has already set the data from bus (DFB) interrupt source bit. Receiving data while the DFB bit is set generates an interrupt to the system, assuming DFB\_IS\_EN is set. In response to the interrupt, the system reads the Interrupt Source register (ISR), where it finds the DFB bit set and the TAG bit clear. These bits cannot be set at the same time; they are mutually exclusive. The system then performs the data read sequence (see 3.4.4, “Data Transfers From the ADB to the System,” on page 86); when the data read sequence is complete, it clears the DFB interrupt source bit. This indicates to the ADB controller that the system has finished reading the data in the CDRF and that the CDRF is now available. The ADB controller remains idle until the DFB bit is cleared.

When the DFB bit is 0, the ADB controller again checks the TAR bit. If the TAR bit is set, the ADB controller suspends autopoll operations and sets the TAG interrupt source bit, which generates an interrupt to the system (if it is enabled). In response to the interrupt, or in response to polling to see if TAG is asserted, the system reads the ISR. There it finds TAG set and DFB clear. The system can now write data to the CDRF (see the next section). When the system has completed the command/data write sequence, it sets the DTB bit, clears the TAR bit, and clears the TAG bit. The order of these actions is important; the TAG bit must be cleared last in the sequence.

When the ADB controller sees that the TAG bit is clear, the TAR bit is clear, and the DTB bit is set, it sends data to the bus. If the ADB controller sees that the TAG and TAR bits are clear, but the DTB bit is not set, the system resumes autopolling (if enabled) or remains idle. This condition may occur if the system was updating a timing value register or control register. After processing the commands or data in the CDRF, the ADB controller resumes autopolling if autopolling is enabled.

### 3.4.3 Data Transfers From the System to the ADB

To transfer data to the ADB, the system writes to the ADB controller Control register and sets the transfer access request (TAR) bit. The system then waits for an interrupt from the ADB controller (or alternately polls the ADB controller Interrupt Source register), and waits until the transfer access grant interrupt source bit (TAG) is set. When the TAG bit is set the system knows it may then send data to the ADB. Because they are mutually exclusive, the DFB bit is clear if the TAG bit is set.

The system may then write the command byte and from 0 to 8 bytes of data to the ADB controller CDRF, writing the command byte to address 1 and bytes 0 to 8 to addresses 2 to 9. After the system has written a command byte and data to the CDRF, it sets the how many bytes (HMB[3:0]) bits in the Control register to the number of data bytes, plus one for the command byte. It sets the command response expected bit (CRE) if it expects a response to the command that was just written to the CDRF (as in the case of a Talk command). It also sets the data to bus (DTB) bit in the Control register. The system then clears the TAR and TAG bits. When the TAG bit is cleared, the ADB controller checks the DTB bit and send data to the ADB. It also reads any response data and posts it to the CDRF. This completes the process of transferring data from the system to the ADB.

### 3.4.4 Data Transfers From the ADB to the System

The ADB controller acquires data to send to the system in one of two ways: in response to an explicit command from the system or as the result of an autopoll sequence. The following description assumes that the ADB controller has not granted the system access to the CDRF and is waiting for the system to complete a write sequence to the CDRF. If access is granted to the system, the ADB controller cannot acquire data.

The ADB controller does not perform autopolling once it has granted access to the system. It does not receive data in response to an explicit command unless the system has completed a transfer to the CDRF, the ADB controller is now transferring the data to the ADB, and the CRE bit is set.

When the ADB controller acquires data, it fills the 9-byte CDRF with the command (either the command sent from the system or the Talk command that caused a response to an autopoll) and 0 to 8 data bytes, arranged at addresses 2 to 9 in the order they were received. The ADB controller then sets the how many bytes (HMB[3:0]) bits in the status register to the number of data bytes plus one for the command byte, sets any appropriate error bits in the status register (see the next section), and sets the DFB interrupt source bit in the status register. It also sets the autopoll data bit (APD) in the status register if the data was from an autopoll or SRQ autopoll action. The ADB controller then waits until the system clears the DFB bit before resuming autopolling.

## 3.5 Error Handling

**Requirement 3-4:** The ADB controller must recognize the error conditions listed in Table 48 and must handle them as described in Section 3.5.

Table 48. ADB error conditions

Type of Error	Description
Data lost	Additional data was expected but no data was received
No response	A device was issued a Talk command but no data was received
SRQ autopolling	An SRQ was detected during autopolling but no device was found to be requesting the bus
Inactive device response	An SRQ was asserted from a device not currently reflected in the Active Device register

### 3.5.1 Data Lost Error

A data lost error occurs when the ADB controller is in the process of reading data from the bus, at least 1 bit but less than 8 bits have been received, and no additional signal edges are detected within the maximum period time of a bit cell ( $t_{cyc}$ , shown in Figure 21). The ADB controller terminates the reading process, posts the data that was received (appending zeros to any undetected bits of an unfinished byte), sets the DLE error bit in the status register, and sets the DFB bit in the Interrupt Source register. It also generates an IRQ to the system (only if IRQs are enabled).

The system is responsible for handling data lost errors. It may respond by resending the previous command, but it should account for the possibility that the error may be repeated.

### 3.5.2 No Response Error

A no response error occurs when the ADB controller implements a command from the system by issuing a Talk command to a device that is known to be on the bus, but there is no response from the device—no start bit is detected within the maximum stop-to-start time ( $t_{tmax}$ ). The ADB controller loads the command from the system into the CDRF, sets the NRE error bit in the status register, and sets the DFB bit in the Interrupt Source register.

The system is responsible for handling no response errors. It may respond by resending the previous command, but it should account for the possibility that the error may be repeated.

### 3.5.3 SRQ Autopolling Error

An SRQ autopolling error occurs if an SRQ was detected during autopolling but no device requested the bus. The ADB controller performs the SRQ autopolling sequence and tries to resolve any errors, such as no known devices asserting the service request. If the ADB master cannot resolve the error it returns to the main idle loop and resumes autopolling, if autopolling is enabled.

### 3.5.4 Inactive Device Response Error

An inactive device response error occurs if an SRQ was detected during autopolling and the SRQ autopolling and error recovery sequence determines that the SRQ originated from a device not currently reflected in the Active Device register. The ADB controller sets the DFB bit in the Interrupt Source register to

tell the system that there is data in the CDRF. The address of the device is reflected in the command byte that is stored in the CDRF. The system is responsible for deciding what to do with this information.

## 3.6 Autopolling, SRQ Autopolling, and Error Recovery

**Requirement 3-5:** The ADB controller must perform the functions described in Section 3.6.

When the ADB controller is performing autopolling, it maintains a counter to determine when the next autopoll event is to occur. At every  $t_{AP}$  time the ADB controller performs an autopoll sequence if the autopoll enable (APE) bit in the Control register is set, CDRF write access has not been granted to the system, and the CDRF is available for use (it contains no previous data from the ADB).

An autopoll sequence consists of sending a Talk register value of 0 to the MRU (most recently used) device. If a response is received from the device, the command and the resulting data are pushed into the CDRF and the system notified as described in Section 3.4.4. If no response is received, the ADB controller returns to idle. If a service request (SRQ) is detected, the ADB controller performs an SRQ autopoll sequence.

The SRQ autopoll sequence is performed when an SRQ is detected during an autopoll sequence and no data is returned from the current MRU device. The SRQ autopoll sequence begins with checking the next most recently used (NMRU) device to determine if it asserted the service request. If it did, the NMRU device now becomes the MRU, the MRU becomes the NMRU, the data is posted to the CDRF, and the system is notified as described in Section 3.4.4. If the NMRU was not the requesting device, the ADB controller searches for the device, beginning with the active devices located at addresses 0x00 through 0x07 and then checking the active devices located at addresses 0x0F through 0x08 (skipping the MRU and NMRU devices). If a device is found to be a requesting device, it becomes the MRU, the MRU becomes the NMRU, the data is posted to the CDRF, and the system is notified as described in Section 3.4.4. If no requesting device is found among the active devices, the ADB controller searches the addresses that are not marked as active to try to find the device that asserted the SRQ. If a device is found, the data is posted to the CDRF, the IDR (inactive device response error) bit is set, and the system is notified as described in Section 3.4.4.

If no device is found, the MRU and NMRU remain unchanged. The ADB controller considers this condition to be an error, since it does not know what to

do when an SRQ is detected but no device can be found that asserted the service request. However, the system may not want to be notified of this type of error since it may not be able to respond appropriately.

If the system wants to be notified when no device is found, it can set the SRQ autopoll error enable (SAE\_EN) bit in the Error Status register. If the SAE\_EN bit is set, and this type of error occurs, the ADB controller sets the SAE bit in the Error Status register and the DFB bit in the Interrupt Source register. If enabled (by DFB\_IS\_EN), the DFB bit causes an interrupt to the system. The system should respond to the interrupt by reading the Error Status register, where it will find the SAE bit set. The system must clear the SAE and DFB bits. It is not necessary to read any data from the CDRF (or to flush it) since no data has been posted. If SAE\_EN is cleared, then this type of error is not reported to the system—that is, SAE is not set and the ADB controller resumes operation by beginning an autopoll sequence after the next  $t_{AP}$  time.

The APD bit is set in all cases where data is posted to the CDRF from an autopoll or SRQ autopoll sequence.

The ADB controller may conclude the autopoll sequence in one of the following four ways:

1. by reading no data from the MRU device and generating no SRQ
2. by reading data from the MRU
3. by responding to an SRQ and reading data from the device that requested the service
4. by responding to an SRQ and not being able to resolve the SRQ, resulting in an SRQ error

No system interaction is required for the first case. The ADB controller issues an interrupt to the system in the last three cases. Data is available in the CDRF in the second and third cases, and the APD bit is set also. The SAE error bit is set in the fourth case. For the last three cases, the DFB is set.

## 3.7 ADB Controller Special Functions

**Requirement 3-6:** The ADB controller must perform the special functions described in Sections 3.7.1 through 3.7.3.

### 3.7.1 Keyboard-Invoked Reset

The ADB controller monitors the ADB bit stream to determine if a keyboard-invoked reset key sequence has been initiated. The reset key sequence consists of the Control and Command keys held down in combination with the Reset (Power On) key. When this key combination is detected, the ADB controller asserts the reset trigger output for a minimum of 300 processor clock cycles (3 ms typical). The ADB controller also monitors the RESET\_KEY input signal from the keyboard, and issues an explicit Talk command to the modifier register of the default keyboard when this signal is detected. If the Reset key modifiers are asserted, the ADB controller asserts the reset trigger output for a minimum of 300 processor clock cycles.

### 3.7.2 Keyboard-Invoked NMI

The ADB controller monitors the ADB bit stream to determine if the keyboard-invoked nonmaskable interrupt (NMI) key sequence is initiated. The NMI key sequence consists of the Command key held down in combination with the Reset (Power On) key. When this key combination is detected, the ADB controller asserts the NMI trigger output for a minimum of 3 PCI bus clock cycles (typically 150 ns). The ADB controller also monitors the RESET\_KEY input signal from the keyboard, and sends an explicit Talk command to the Modifier register of the default keyboard when this signal is detected. If the NMI key modifiers are asserted, the ADB controller asserts the NMI trigger output for a minimum of 3 PCI bus clock cycles.

### 3.7.3 Power Message Handling

The ADB controller does not need to implement power messages because it does not have control of the power supply. The ADB controller passes the data for a Safe Off request to the system, which may then handle the request.

## 3.8 Bit Cell Determination and Timing

**Requirement 3-7:** ADB bit cell timing must occur as shown in Figure 21 and must fall within the limits shown in Table 49.

Figure 21. Bit cell timing diagram

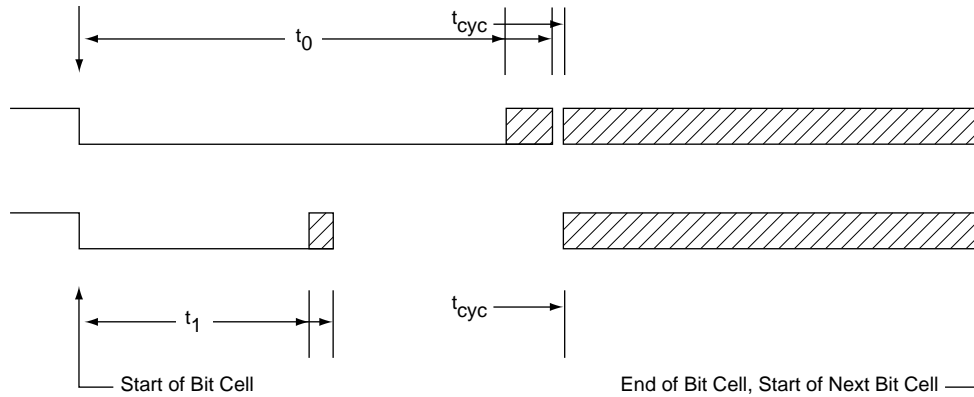


Table 49. Bit cell timing limits

Parameter	Minimum, $\mu\text{s}$	Maximum, $\mu\text{s}$
$t_0$	61.75	68.25
$t_1$	33.25	36.75
$t_{\text{cyc}}$	70	130

**Requirement 3-8:** The ADB controller must determine bit cell values as specified in Section 3.8.1.

### 3.8.1 Bit Cell Sampling Algorithm

Although they may conform to the limits shown in Table 49, the bit cell 0 time and 1 time, along with the specified cycle time, can cause an indeterminate bit cell to be specified as either 1 or 0. If  $t_0$  was  $65 \mu\text{s}$  (within specification) and  $t_{\text{cyc}}$  was  $120 \mu\text{s}$  (also within specification), it would not be possible to determine the value of a cell. Also, if  $t_1$  was  $35 \mu\text{s}$  (within specification) and  $t_{\text{cyc}}$  was  $70 \mu\text{s}$  (also within specification), it would not be possible to determine the value of the cell.

To determine bit cell values, the  $\pm 30\%$  tolerance allowed in the specification for  $t_{\text{cyc}}$  specifies the total variance in bit cell timing, and the 65/35 ratio of high/low or low/high times represents a division of the total cycle time with a  $\pm 5\%$  tolerance.



### 3.8.1.1 Bit Determination Algorithm

The algorithm to determine bit cell timing compares the low and high times of the bit cell and uses their ratio to determine the value of the cell. When a bit cell is expected (following a start bit after a command has been sent and before a stop bit), a counter with an initial value of 0x00 is enabled in a count-up mode on the falling edge of the first bit cell. The counter continues to count up until a rising edge is detected or a timeout condition is encountered (see Section 3.8.1.2). When a rising edge is detected, the counter starts counting down. The counter continues to count down until a falling edge is encountered or a timeout condition occurs. When the falling edge is encountered, the value of the sign bit of the counter is the value of the bit cell. On the falling edge, the counter is cleared and restarted in the count-up mode to start the timing for the next bit cell.

### 3.8.1.2 Bit Determination Timeout

A bit sampling timeout can occur in two instances: the last bit cell encountered was the last bit of the last word of a data transfer, or the ADB bit stream stopped for some unknown reason. A timeout is detected by the absence of an expected edge within the tolerance limit of the bit cell cycle time. In the first timeout case, where the controller is waiting for a bit cell that is the first bit of a data byte, no error is present if the controller has received previous data bytes; it has just reached the end of the data transaction. At this time the data receive transaction is complete, and data is ready to be sent to the system. In the second timeout case, where the ADB bit stream stops, the ADB controller ends the data receive transaction and reports an error to the system.

## 3.8.2 Miscellaneous ADB Timing Cautions

Some ADB devices do not meet the specified 65  $\mu$ s / 35  $\mu$ s bit cell timing. For example, one Apple mouse device model uses 55  $\mu$ s / 35  $\mu$ s timing. The up/down count method used to determine bit cell timing resolves this problem.

Some ADB devices do not meet the 200  $\mu$ s  $\pm$ 30%  $t_{lt}$  stop-to-start timing requirement. For example, some keyboards require 265  $\mu$ s to respond to a Talk command. This value is programmable; the default value allows 265  $\mu$ s for  $t_{lt}$ .

Some ADB devices do not properly assert SRQ. Some devices wait until the end of the stop bit has been transmitted and the data line is pulled high before asserting the data line low to indicate the SRQ. The ADB controller waits 5  $\mu$ s after the end of the stop bit before sampling to determine if an SRQ is present. The algorithm for sampling SRQ timing includes a state to allow this delay.

The ADB architecture requires a 50% bus utilization ratio. Standard implementations, including all ADB controllers designed by Apple, insert a 1.5 to 1.6 ms wait between transactions to satisfy this requirement.

## 3.9 Controller Hard Reset

**Requirement 3-9:** The ADB controller must handle reset commands as described in Section 3.9.

The system can instruct the ADB controller to issue an ADB hard reset at any time by setting the ADB\_RST bit.

When the ADB controller finds the ADB\_RST bit set, it drives the ADB data out line low for a minimum of 3 ms, then releases the line. The open collector output is pulled up to deassertion state. The ADB controller waits for the programmed holdoff time, then clears the ADB\_RST bit, signalling to the system that the ADB reset sequence is complete.

**Note:** The ADB controller cell does not interpret the ADB SENDRESET command (0bxxxx0000) to indicate that it should perform an ADB hard reset sequence as described above. If the system wants a hard reset to be sent to ADB instead of the SENDRESET command, it must interpret this command and assert the ADB\_RST bit, as described above, instead of writing the SENDRESET command to the CDRF. The ADB controller sends a SENDRESET command like any other ADB command.

The System may set the APE (autopoll enable) bit or the TAR (transfer access request) bit after it sets the ADB\_RST bit. The ADB controller responds as soon as the reset sequence is completed. There is no need for the ADB controller to notify the system that the reset cycle has completed. Either the TAG bit will be set (in response to TAR) or the DFB bit will be set upon completion of an autopoll sequence, resulting in data received or an error. If the system needs to know when a reset sequence has been concluded it may poll the state of the ADB\_RST bit, which the ADB controller clears upon completion of the reset sequence.

## 3.10 System Interface to the ADB

The ADB controller communicates with the host computer system by means of the simple interface protocol described in this section. The interface consists of an 8-bit data bus, a 5-bit address bus, a read/write control signal (R/W), and an address-decoded chip select signal (CS).

### 3.10.1 Status and Control Registers

**Requirement 3-10:** The ADB controller must implement the registers listed in Table 50.

Table 50. ADB status and control registers

Name	Address
Interrupt Status	0b00000
Command	0b00001
Data1	0b00010
Data2	0b00011
Data3	0b00100
Data4	0b00101
Data5	0b00110
Data6	0b00111
Data7	0b01000
Data8	0b01001
Interrupt Source Enable	0b01010
Data Type/Count	0b01011
Error Status	0b01100
Control	0b01101
Autopoll Control	0b01110
Active Device Address HIGH	0b01111
Active Device Address LOW	0b10000
Test	0b10001

The bit assignments for the registers listed in Table 50 are shown in Figure 22, where unused bits are indicated by shading.





Figure 22. Status/control register bits (*continued*)

Active Device Address LOW Register							Address 0b10000	
7	6	5	4	3	2	1	0	
AD_AD[F]	AD_AD[E]	AD_AD[D]	AD_AD[C]	AD_AD[B]	AD_AD[A]	AD_AD[9]	AD_AD[8]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Test Register							Address 0b10001	
7	6	5	4	3	2	1	0	
TST[7]	TST[6]	TST[5]	TST[4]	TST[3]	TST[2]	TST[1]	TST[0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

The meanings of the status and control bits shown above are specified in the following sections.

**Requirement 3-11:** The ADB controller must implement the bit actions described in Sections 3.10.1.1 through 3.10.1.19.

### 3.10.1.1 Command/Data Register File (CDRF)

The 9-byte CDRF file consists of the command byte at address 0b00001 and the 8 data bytes at addresses 0b00010 through 0b01001.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read/write
<i>0 means:</i>	depends on data
<i>1 means:</i>	depends on data
<i>Action on reset:</i>	none required

### 3.10.1.2 Transfer Access Request (TAR)

The System uses the TAR bit to inform the ADB controller that it would like to send an explicit command (or command and data) to the ADB. For details of its operation, see 3.10.1.8, “Data Transfer Process,” on page 99.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read only
<i>0 means:</i>	done, or no data to send
<i>1 means:</i>	system has data to send
<i>Action on reset:</i>	cleared (no transfer request)

### 3.10.1.3 Transfer Access Grant (TAG)

The TAG bit determines whether or not a transfer may take place. For details of its operation, see 3.10.1.8, “Data Transfer Process,” on page 99.

<i>System access:</i>	read/clear (write 0)
<i>ADB controller access:</i>	read/set (write 1)
<i>0 means:</i>	not OK to transfer
<i>1 means:</i>	OK to transfer
<i>Action on reset:</i>	cleared (no transfer grant)

### 3.10.1.4 TAG Interrupt Source Enable (TAG\_IS\_EN)

The system must set the TAG\_IS\_EN bit if it wants the ADB controller to issue an interrupt when write access to the CDRF is granted (that is, when the TAG bit is set). See 3.10.1.8, “Data Transfer Process,” on page 99.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read only
<i>0 means:</i>	TAG interrupt source disabled
<i>1 means:</i>	TAG interrupt source enabled
<i>Action on reset:</i>	cleared (TAG interrupt source disabled)

### 3.10.1.5 Data From Bus (DFB)

The value of the DFB bit indicates whether or not data from the ADB is present. See 3.10.1.8, “Data Transfer Process,” on page 99.

<i>System access:</i>	read/clear (write 0)
<i>ADB controller access:</i>	read/set (write 1)
<i>0 means:</i>	no data or error message from ADB
<i>1 means:</i>	valid data or error message from ADB
<i>Action on reset:</i>	cleared

### 3.10.1.6 DFB Interrupt Source Enable (DFB\_IS\_EN)

The system must set the TAG\_IS\_EN bit if it wants the ADB controller to issue an interrupt when data from the ADB is present. For further details, see 3.10.1.8, “Data Transfer Process,” on page 99.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read only
<i>0 means:</i>	DFB interrupt source disabled
<i>1 means:</i>	DFB interrupt source enabled
<i>Action on reset:</i>	cleared (DFB interrupt source disabled)

### 3.10.1.7 Data to Bus (DTB)

The value of the DTB bit indicates whether or not data from the system should be sent to the ADB. For details of its operation, see the next section.

<i>System access:</i>	read/set (write 1)
<i>ADB controller access:</i>	read/clear (write 0)
<i>0 means:</i>	no data to transfer to the ADB
<i>1 means:</i>	valid data to transfer to the ADB
<i>Action on reset:</i>	cleared

### 3.10.1.8 Data Transfer Process

The TAR, TAG, TAG\_IS\_EN, DFB, DFB\_IS\_EN, and DTB bits are used to facilitate data transfer from the system to the ADB controller, while preventing collisions in the CDRF.

The TAR bit is used by the system to inform the ADB controller that it would like to send an explicit command (or command and data) to the ADB. The System sets TAR, then waits for an interrupt from the ADB controller. The System may elect to poll the Interrupt Source register to determine if access is granted immediately. The System must enable TAG\_IS\_EN if it wants the ADB controller to issue an interrupt when write access to the CDRF is granted (that is, when TAG is set).

The ADB controller checks the TAR bit in its main idle loop, after completing autopolling and while waiting for the next autopoll sequence to begin. If TAR is set while the ADB controller is still idle, the ADB controller sets the TAG bit. The TAG bit is an interrupt source, so an IRQ is generated if TAG\_IS\_EN is set. On responding to the IRQ, the system reads the Interrupt Source register and sample the TAG and DFB bits.

If DFB is set and TAG is cleared (TAG = 0 and DFB = 1), then either the CDRF contains data from the ADB or an error has occurred and the error status register reflects the current error condition. In the latter case, the system must read the data from the CDRF or clear the CDRF pointers (essentially flushing the CDRF), along with the error status bits (DLE and NRE, discussed below). When the system finishes reading from the CDRF it must clear DFB. Note that the ADB controller does not resume operation until DFB is cleared, since it could cause a data overwrite in the CDRF. The clearing of DFB is an indication to the ADB controller that the system has read the data in the CDRF and has read the error status bits. If the system does not wish to read the data in the CDRF (as may be the case for some error conditions), it may simply clear the DFB bit.

If the TAG bit is set and the DFB bit is cleared on responding to the IRQ (TAG = 1 and DFB = 0), then the system is granted write access to the CDRF



and may fill it. The system must also set HMB[3:0] and may set CRE (see Section 3.10.1.14). When the system has finished writing to the CDRF it must set DTB, clear TAR, and clear TAG. When the ADB controller sets TAG in response to TAR being set by the system, it waits until TAG is cleared before resuming operation, at which time the ADB controller checks DTB to determine if there is data in the CDRF that must be sent to the ADB. If DTB is set, the ADB controller sends the data to the system as specified by HMB[3:0], and looks for a response from the addressed device if CRE is set. Upon completion of the data transfer to the bus, the ADB controller clears the DTB bit.

### 3.10.1.9 Interrupt Request (IRQ)

The IRQ bit is the OR-combined value of the TAG and DFB interrupt source bits. This bit is set when interrupts are enabled and either the TAG or DFB bits are set—(TAG and TAG\_IS\_EN) or (DFB and DFB\_IS\_EN). When the IRQ bit is set, the IRQ output from the ADB controller cell is driven low. When the bit is clear, the IRQ output from the ADB controller cell is driven high. There is no direct enable for this interrupt, because there are individual enables for the TAG and DFB bits.

<i>System access:</i>	read only
<i>ADB controller access:</i>	OR-combined TAG and DFB (if enabled)
<i>0 means:</i>	no interrupt generated
<i>1 means:</i>	interrupt generated
<i>Action on reset:</i>	cleared (no interrupt)

### 3.10.1.10 Data Lost Error (DLE)

The DLE bit is set when the ADB controller is in the process of reading data from the bus, at least 1 bit but less than 8 bits have been received, and no additional signal edges are detected within the maximum period time of a bit cell ( $t_{cyc\ max}$ ). The ADB controller terminates the reading process, posts the data that was received (appending zeros to any undetected bits of an unfinished byte), sets the DLE error bit in the status register, and sets the DFB bit in the Interrupt Source register. It also generates an IRQ to the system (only if IRQs are enabled).

<i>System access:</i>	read/clear (write 0)
<i>ADB controller access:</i>	set only (write 1)
<i>0 means:</i>	no DLE error
<i>1 means:</i>	data lost error (DLE)
<i>Action on reset:</i>	cleared

### 3.10.1.11 No Response Error (NRE)

The NRE bit is set if, in response to a command from the system, the ADB controller issues a Talk command to a device that is known to be on the bus and there is no response from the device. In other words, no start bit is detected within the  $t_{lt\ max}$  time (stop-to-start timing, worst case). The DFB bit is also set, resulting in an IRQ to the system.

<i>System access:</i>	read/clear (write 0)
<i>ADB controller access:</i>	set only (write 1)
<i>0 means:</i>	no NRE error
<i>1 means:</i>	no response error (NRE)
<i>Action on reset:</i>	cleared

### 3.10.1.12 Autopoll Data (APD)

The APD bit is set by the ADB controller when the data posted to the CDRF was obtained as the result of an autopoll sequence. It is left to the system to determine what to do with this information. The ADB controller clears the APD bit at the start of any autopoll sequence.

<i>System access:</i>	read only
<i>ADB controller access:</i>	write only
<i>0 means:</i>	data not from autopoll
<i>1 means:</i>	data from autopoll
<i>Action on reset:</i>	cleared

### 3.10.1.13 How Many Bytes (HMB[3:0])

The four HMB bits are used to indicate the number of valid command and data bytes that have been stored in the CDRF. The ADB controller sets the HMB bits when data is being transferred from the ADB to the system; the system sets them when data is being transferred from the system to the ADB. The HMB bits contain a 4-bit binary code from 0 to 9 that indicate the number of valid bytes in the CDRF. They must never be set to a value greater than 9.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read/write
<i>0 means:</i>	binary byte count
<i>1 means:</i>	binary byte count
<i>Action on reset:</i>	cleared

### 3.10.1.14 Command Response Expected (CRE)

The system sets the CRE bit to inform the ADB controller that it expects a response from the ADB device addressed by the command that is currently in the CDRF. This occurs when the system is sending a Talk command (performing a read from the ADB). When the CRE bit is set, the ADB controller samples the bus for a response from the device that was addressed. This may result in an error condition, or may result in data being posted to the CDRF.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read only
<i>0 means:</i>	no response expected
<i>1 means:</i>	response expected
<i>Action on reset:</i>	cleared

### 3.10.1.15 Autopoll Enable (APE)

The system sets the APE bit when it wants the ADB controller to perform autopolling. The ADB controller responds by performing autopolling if the APE bit is set, or does not if it is cleared.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read only
<i>0 means:</i>	autopolling disabled
<i>1 means:</i>	autopolling enabled
<i>Action on reset:</i>	cleared (autopolling disabled)

### 3.10.1.16 Active Device Address Registers

The bits in the AD\_AD registers indicate which addresses on the ADB contain currently active devices. The system sets each bit accordingly during the power-up sequence. The ADB controller uses the information in these registers during the SRQ Autopolling sequence.

<i>System access:</i>	read/write
<i>ADB controller access:</i>	read only
<i>0 means:</i>	no device at address
<i>1 means:</i>	device present at address
<i>Action on reset:</i>	cleared

### 3.10.1.17 ADB Reset (ADB\_RST)

When the system sets the ADB\_RST bit, the ADB controller issues an ADB reset. The ADB controller clears the ADB\_RST bit upon completion of the ADB reset action.

<i>System access:</i>	read/set only (write 1)
<i>ADB controller access:</i>	read/clear only (write 0)
<i>0 means:</i>	no ADB reset
<i>1 means:</i>	reset ADB
<i>Action on reset:</i>	cleared

### 3.10.1.18 Monostable Reset Enable (MR\_EN)

This bit is no longer supported.

### 3.10.1.19 ADB Controller Test Register (TST[7:0])

The definition and function of the TST bits are currently not determined.

## 3.10.2 Timing Value Registers

An additional set of registers in the ADB controller provides flexibility to adjust the ADB bit cell and autopoll timing parameters during prototype testing and evaluation.

These timing value registers can be directly addressed; they are read/write to the system and read-only to the ADB controller. They should be initialized with a reasonable predetermined timing value. The absolute minimum and maximum timing values for each of these parameters are limited only by the physical sizes of the registers.

Table 51 details the register sizes, the minimum and maximum values that may be programmed with their associated timing values, and the recommended default register and timing values.

**Requirement 3-12:** The ADB controller must implement timing registers as specified in Table 51.

Table 51. Timing register parameters

Register	Size, Bits	Maximum Value	Maximum Time, $\mu\text{s}$	Minimum Value	Minimum Time, $\mu\text{s}$	Default Value	Default Time, $\mu\text{s}$
Zero low time	6	33	67.32	31	63.24	32	65.28
One low time	5	18	36.72	17	34.68	17	34.68
Reset pulse width	12	4095	8353.8	1471	3000.8	1471	3000.8
Synch pulse width	6	32	65.28	31	63.24	32	65.28
Attention pulse width	9	403	822.12	381	777.24	393	801.72
Stop pulse width	6	35	71.4	34	69.36	35	71.4
Bit cell period $t_0$	7	63	128.52	35	71.4	64	130.56
Stop-to-start pulse width	8	127	259.08	69	140.76	125	255.31
Autopoll time period	6	63	15378	1	244.1	46	11288

# SCSI Support

# 4

## 4.1 Overview

The Small Computer System Interface (SCSI) bus can be used to support mass data devices, such as hard disk drives and CD-ROM drives, in addition to any IDE bus the computer may have. This chapter defines MESH (Macintosh Enhanced SCSI Hardware), the Macintosh implementation of the SCSI bus controller. The basic controller architecture defined here is intended to be relatively simple, allowing the maximum latitude in cost, performance, scalability, and flexibility. Its general functionality resembles that of the NCR 53C9x family of chips, with certain differences as described in Section 4.8.3.

**Note:** For information about using MESH in the CHRP architecture, see *PowerPC Microprocessor Common Hardware Reference Platform: I/O Reference*. This document is listed in the bibliography.

The fastest asynchronous transfer rate on a SCSI bus controlled by MESH is about 12 MB per second for reads and 8 MB per second for writes, assuming no performance degradation by the connecting cable.

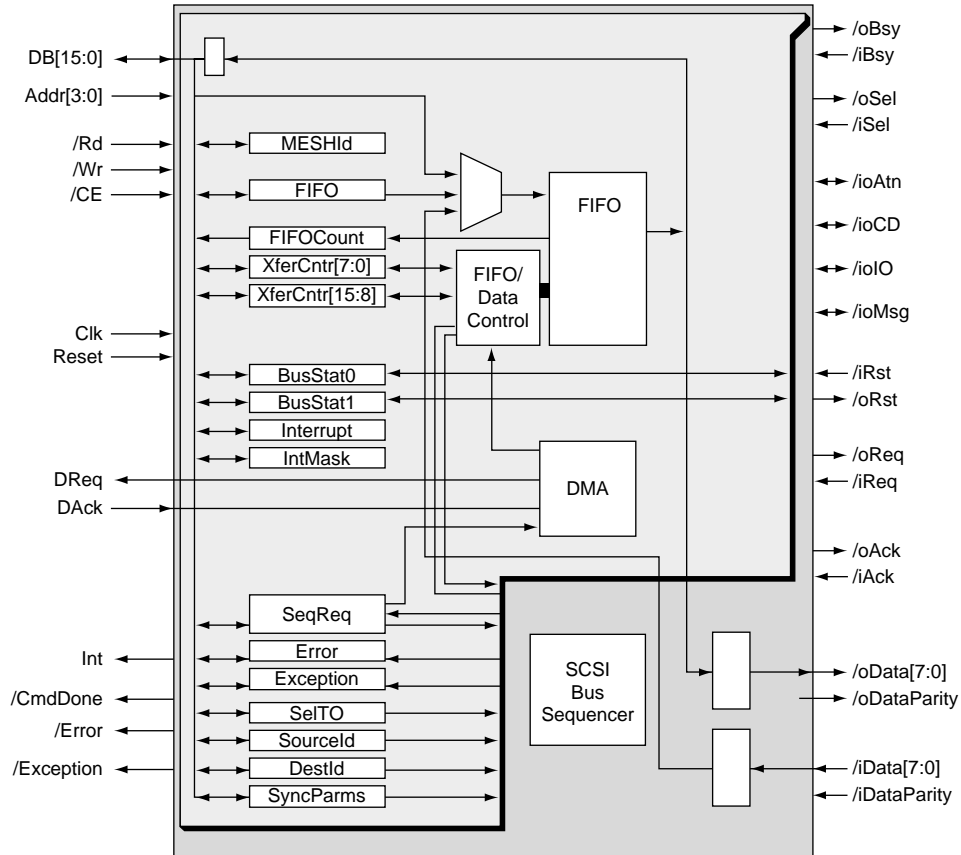
This chapter discusses the SCSI bus and its interface from the viewpoint of the host computer system. However, it also provides useful information for designers of target SCSI implementations in peripheral devices. For further information about building SCSI peripheral devices compatible with the Mac OS see *Designing PCI Cards and Drivers for Power Macintosh Computers*, listed in the bibliography.

**Note:** An Apple-designed MESH SCSI controller is included in the Mac I/O chip, as described in Section 2.11.4, “SCSI Support,” on page 69.

## 4.2 Architecture

The general architecture of the MESH SCSI controller is diagrammed in Figure 23.

Figure 23. MESH controller architecture



The interface signals shown in Figure 23 are listed in Table 52 and are further defined in Section 4.5.

**Requirement 4-1:** The MESH SCSI controller must implement the signals specified in Table 52.

Table 52. SCSI controller interface

Name	I/O Type	Description
/DAck	TTL input	DMA input from DMA controller (see Chapter 5)
/CmdDone	TTL output	Indicates that the last command has finished
/Error	TTL output	Indicates an error from the last command
DBus [15:0]	TTL I/O	Bidirectional data bus for DMA and register access
/iData [7:0]	TTL input	SCSI input data bus
/iDataParity	TTL input	SCSI input data bus parity
/oData [7:0]	SCSI output	SCSI output data bus
/oDataParity	SCSI output	SCSI output data bus parity
/oSel	SCSI output	SCSI bus select output
/oBsy	SCSI output	SCSI bus busy output
/oReq	SCSI output	SCSI bus request output
/oAck	SCSI output	SCSI bus acknowledge output
/ioMsg	SCSI I/O	SCSI bus message
/ioCD	SCSI I/O	SCSI bus control and data
/ioIO	SCSI I/O	SCSI bus I/O
/ioAtn	SCSI I/O	SCSI bus attention
/oRst	SCSI output	SCSI bus reset output
/iSel	SCSI input	SCSI bus select input
/iBsy	SCSI input	SCSI bus busy input
/iReq	SCSI input	SCSI bus request input
/iAck	SCSI input	SCSI bus acknowledge input
/iRst	SCSI input	SCSI bus reset input
/Int	TTL output	Interrupt
Reset	TTL input	Chip reset
/Wr	TTL input	Write to internal registers



Table 52. SCSI controller interface (*continued*)

Name	I/O Type	Description
/Rd	TTL input	Read internal registers
/CE	TTL input	Chip enable to access internal registers
Addr [3:0]	TTL input	Address internal registers
Clk50	TTL input	Clock
/Exception	TTL output	Unexpected SCSI exception
/RawReq	SCSI input	Unfiltered SCSI bus synchronous request (See Figure 45 on page 137)
ScanEnable	TTL input	For scan testing of controller (not used in operation)
/TestMode	TTL input	Test input (not used in operation)
DReq	TTL output	DMA request for a DMA transfer

### 4.3 Electrical Interface

This section defines the electrical signals to and from the MESH SCSI controller chip.

**Requirement 4-2:** The electrical representation of the I/O types shown in the second column of Table 52 must conform to the limits shown in Table 53.

Table 53. SCSI and TTL signal levels

I/O Type	Electrical Limits
TTL input	$V_{il} \leq 0.8 \text{ V}$ ; $V_{ih} \geq 2.0 \text{ V}$
TTL output	$V_{ol} \leq 0.4 \text{ V @ } I_{ol} = 4 \text{ mA}$ ; $V_{oh} \geq 2.4 \text{ V @ } I_{oh} = -2 \text{ mA}$
SCSI input	$V_{il} = 0.8 \text{ V}$ and $V_{ih}$ of 2.0 V. Hysteresis of 0.3 V on iReq, iAck, and RawReq with a nominal center voltage of $1.5 \text{ V} \pm 0.1 \text{ V}$
SCSI output	0.5 V @ $I_{ol} = 48 \text{ mA}$ , open collector (open drain)
TTL I/O	Has characteristics of both TTL input and TTL output
SCSI I/O	Has characteristics of both SCSI input and SCSI output

## 4.4 Registers

**Requirement 4-3:** The MESH SCSI controller must communicate through 16 one-byte registers as specified in Table 54.

Table 54. SCSI controller registers

Name	Offset	Read/Write	Action on Reset
XferCount0	0x0	Read/write	
XferCount1	0x1	Read/write	
FIFO	0x2	Read/write	
Sequence	0x3	Read/write	Cleared
BusStatus0	0x4	Read/write	
BusStatus1	0x5	Read/write	
FIFOCount	0x6		Cleared
Exception	0x7		Set to ones
Error	0x8		Set to ones
Interrupt mask	0x9		Cleared
Interrupt	0xA		
SourceID	0xB		
DestID	0xC		
SyncParms	0xD		
MESHId	0xE	Read only	
SelTO	0xF		

**Requirement 4-4:** The registers listed in Table 54 must operate as describes in Sections 4.4.1 through 4.4.14.

### 4.4.1 Transfer Count Registers

The Transfer Count registers, XferCount0 and XferCount1, are used to indicate the number of bytes to be transferred in the subsequent information transfer

phase of a SCSI bus transaction. XferCount0 contains the less significant byte of the count; XferCount1 contains the more significant byte. See “Sequence Register” on page 111 for more details on using the Transfer Count registers.

If a phase’s Sequence command exhausts the count before a phase change or error occurs, then the CmdDone signal becomes true without an exception interrupt. This allows for simple sequencing of phases as long as the interrupt signal is false.

The transfer counter (TC) is decremented whenever a byte is written into the FIFO. For data transfers to the SCSI bus, the TC is decremented every time the processor writes to the FIFO or a DMA word is transferred. This behavior is independent of whether or not a command is pending. When the TC reaches 0 and the FIFO becomes empty, the CmdDone bit is set, indicating the completion of the command. On transfers into the MESH controller, the TC is decremented every time a byte is brought into MESH from the SCSI bus (under hardware control only).

The order of any data transfer should start with setting up the TC, issuing the command, then transferring the data. Technically, the data could be put in the FIFO before the command is issued, but this limits the general model to transferring no more than 16 bytes before issuing the command.

Transfer counter behavior is subject to these special cases:

- In synchronous DataIn, the TC decrements on the DMA transfer, not on the SCSI bus write action into the FIFO.
- If all 16 bits of the TC are zeros at the beginning of a command, 64 KB are transferred.

**Note:** A problem can arise in some programming cases. For instance, if software loads the FIFO with the command descriptor block (CDB), loads the TC with the CDB size (say 6), and then sends a CommandPhase command to the MESH controller, the CmdDone bit will never be set because the TC never goes down to 0 because no bytes are loaded into the FIFO after setting the TC.

## 4.4.2 Bus FIFO Register

The FIFO register accesses the SCSI bus using programmed I/O, or as the buffer in DMA mode. The current number of bytes in the FIFO is indicated in the FIFO Count register. The FIFO depth is 16 bytes.

If the MESH controller is reselected as a host (or selected as a target), the FIFO register contains the image of the selection phase data. For example, if MESH’s ID was 6, and the reselecting target’s ID was 1, the value in the FIFO after the reselection would be 0b01000010.

### 4.4.3 Sequence Register

The Sequence register is used to direct the controller into a SCSI bus phase. In the initiator mode, the requested sequence is the one that the controller expects the target will enter into. In the target mode, the phase is the one that the controller will enter into next. The bits in the Sequence register have the functions shown in Figure 24.

Figure 24. Sequence register format

DMA	TMode	Atn	ActNeg	seq[3]	seq[2]	seq[1]	seq[0]
-----	-------	-----	--------	--------	--------	--------	--------

The ways in which these bits are used are described in the next sections.

#### 4.4.3.1 DMA Bit

The DMA bit indicates that the transfer is to be done using DMA. If this bit is enabled, FIFO accesses are disabled.

#### 4.4.3.2 TMode Bit

When set, the TMode bit puts the controller into target mode for executing the command. The TMode bit is cleared for initiator mode.

#### 4.4.3.3 Atn Bit

In selection and information transfer phases, the Atn bit indicates that the Atn signal is also asserted.

#### 4.4.3.4 ActNeg Bit

If the ActNeg bit is set, the SCSI controller adopts active negation mode. In this mode it actively negates the Req and Ack signals by pulling them high instead of relying on line terminators. Only the Req and Ack signals are affected. If both ActNeg and TMode are set, the controller pulls Req actively both low and high; if TMode is clear, then Ack is actively negated. In active negation, if  $I_{oh} \leq 20$  mA, then  $V_{oh} \leq 3.0$  V; if  $I_{oh} < 7$  mA, then  $2.0$  V  $\leq V_{oh} \leq 3.24$  V.

### 4.4.3.5 Seq Bits

The four Seq bits, seq3..seq0, contain the encoded value of the commands defined in Table 55.

Table 55. Seq bit encoding

Command	Value	Normal Completion	Exception	Error
Arb	0x1	Arbitration won (2.4µs)	ArbLost, SCSIReset	
Sel	0x2	Destination selected	SelTO, SCSIReset	
Cmd	0x3	Transfer count exhausted	PhaseMM, SCSIReset	
Status	0x4	Transfer count exhausted	PhaseMM, SCSIReset	ParErr
DataOut	0x5	Transfer count exhausted	PhaseMM, SCSIReset	
DataIn	0x6	Transfer count exhausted	PhaseMM, SCSIReset	ParErr
MsgOut	0x7	Transfer count exhausted	PhaseMM, SCSIReset	
MsgIn	0x8	Transfer count exhausted	PhaseMM, SCSIReset	ParErr
BusFree	0x9	Busy signal cleared	PhaseMM, SCSIReset	
EnParChk	0xA	3 clock cycles after write	No interrupts	
DisParChk	0xB	3 clock cycles after write	No interrupts	
EnReSel	0xC	3 clock cycles after write	No interrupts	
DisReSel	0xD	3 clock cycles after write	No interrupts	
RstMESH	0xE	3 clock cycles after write	No interrupts	
FlushFIFO	0xF	3 clock cycles after write	No interrupts	

These commands are described in the next sections.

#### 4.4.3.5.1 Arbitrate Command

Software can use the value in the SourceId register to perform SCSI bus arbitration. The Arbitrate command makes sure that the Bsy and Sel signals have been false for 1200 ns before asserting its encoded SCSI ID along with the Bsy signal. After asserting the Bsy signal, the controller waits for 2400 ns and then compares the IDs on the bus. If SourceID is the highest, the controller asserts the Sel signal and wins arbitration. If some other device has a higher ID or has asserted the Sel signal, then the controller loses arbitration and releases its Bsy

signal and arbitration ID. At this point the controller sets ArbLost bit in the Exception register and generates an interrupt.

#### 4.4.3.5.2 Selection Command

In response to the Selection command, the controller asserts the ID indicated in the DestID register (along with SourceID) and then deasserts the Bsy signal. The controller waits up to the time indicated in the SelTimeOut register (nominally 250 ms) for the Bsy signal to be asserted by the destination device.

The Selection command assumes that the immediately prior command was the Arbitrate command and that the arbitration was won.

If the Atn bit is set, the controller asserts the Atn signal before releasing the Bsy signal, indicating the Selection phase.

If the TMode bit is set, the controller sets the IO bit and enters the Reselection phase.

#### 4.4.3.5.3 Information Transfer Commands

The Cmd, Status, DataIn, DataOut, MsgIn, and MsgOut commands execute until either the transfer count is exhausted, a phase mismatch occurs, or an error occurs. In the initiator mode of any information transfer command, the Ack signal is left asserted until a subsequent command implies that it must be negated. This behavior is helpful in transitioning to the MessageOut phase or generally responding to messaging or data parity errors. An Information Transfer command can be overwritten at any time.

If the TMode bit is set, the MESH controller sets the Msg, CD, and IO signals to appropriate states and begins the Req/Ack action. In target mode, the Atn bit in the Sequence register is used as a phase comparison bit; it is compared against the SCSI bus Atn signal on the trailing edge of the Ack signal. If the value in the Sequence register does not match the SCSI bus Atn on the trailing edge of any Ack, the PhaseMM exception is asserted and the MESH controller stops transferring data.

At the end of any Async data transfer, when the transfer counter counts down to zero, the Ack signal remains low. This happens for a variety of reasons—the host may choose to force a disconnect or there may have been a parity error. The Atn signal may need to be asserted prior to the trailing edge of Ack to go into the MessageOut phase and to keep the target from starting a synchronous data transfer before the Sync parameters are set up in the MESH controller. The Ack signal is negated upon issuance of the next command.

Synchronous data transfer mode is enabled only during DataIn or DataOut phases. The MESH controller remains in synchronous mode between synchronous DataIn or DataOut commands. The Ack signal does not hang in synchronous mode. For further information about synchronous data transfers, see Section 4.4.12.3.

#### 4.4.3.5.4 BusFree Command

The pseudo-phase BusFree command is used to indicate that the bus is expected to transition to the Bus Free phase, which is defined as the Bsy and Sel signals being both false. In initiator mode this implies that the Ack signal must be negated first and then the Bsy signal must be inspected to detect when it goes false. If the MESH controller detects assertion of the Req signal before Bsy becomes false, it sets the PhaseMM bit. If the TMode bit is set, the MESH controller releases the Bsy signal from the bus.

#### 4.4.3.5.5 RstMESH Command

The RstMESH command is used to reset the MESH chip through software. Interrupts are masked by this command, so no interrupts are generated. The CmdDone signal remains active until cleared through the Interrupt register.

#### 4.4.3.5.6 EnParChk Command

The EnParChk command enables parity checking on data transfers. Parity is always checked on reselection phases and generated on data transfer phases. No interrupts are generated by this command.

#### 4.4.3.5.7 DisParChk Command

The DisParChk command disables parity checking on data transfers. Parity is always checked on reselection phases and generated on data transfer phases. No interrupts are generated by this command.

#### 4.4.3.5.8 EnReSel Command

If the TMode bit is false, the EnReSel command enables reselection as a host. It is active until a SCSI bus reset occurs, the DisResel command is issued, or the controller is reselected as a host. No interrupts are generated from issuing this command, but the reselected exception bit is set if the MESH controller is reselected as a host.

If the TMode bit is set when this command is issued, it enables the MESH controller to be selected as a target.

#### 4.4.3.5.9 DisReSel Command

The DisReSel command disables selection or reselection as a host or target. Like the EnReSel command, depending on the state of TMode, it disables either reselection as a host or selection as a target. No interrupts are generated by this command.

#### 4.4.3.5.10 FlushFIFO Command

The FlushFIFO command clears the Bus FIFO Count register (see Section 4.4.5) and sets the FIFO read and write pointers to 0.

#### 4.4.4 Bus Status Registers

The Bus Status registers represent the state of the SCSI bus when read, and can be used to assert signals when written. Normally, these registers are only read to obtain the state of the bus. These registers can be written only when the Exception register is clear. The meanings of the bits in bus Status0 and bus Status1 are given in Figure 25 and Figure 26.

**Note:** In these and other SCSI register diagrams, reserved fields are labeled in italics.

Figure 25. Bus Status0 register format

<i>Req32</i>	<i>Ack32</i>	Req	Ack	Atn	CD	Msg	IO
--------------	--------------	-----	-----	-----	----	-----	----

Figure 26. Bus Status1 register format

Rst	Bsy	Sel	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>
-----	-----	-----	-------------	-------------	-------------	-------------	-------------

The Req32 and Ack32 signals are reserved for potential use in systems that support 32-bit-wide SCSI.

#### 4.4.5 Bus FIFO Count Register

The lower 5 bits of the FIFO Count register (4:0) are continually updated to represent the number of bytes in the FIFO. This register is synchronized with the processor interface so that it does not have to be protected from bounce.

For a data transfer command to be considered complete, both the XferCount and FIFO Count registers must be clear. For example, if 6 bytes of CDB are put into the FIFO and the XferCount register is not 0, the MESH controller expects to receive XferCount more bytes before finishing. In this case, the XferCount register must be set to zero before issuing the Cmd sequence command.

#### 4.4.6 Exception Register

The Exception register contains information about conditions that are not usually errors, but can cause the sequence to stop for processor intervention. It is cleared by writing ones into the bits to be cleared. The meanings of the bits in the Exception register are given in Figure 27.



Figure 27. Exception register format

<i>rsvd</i>	<i>rsvd</i>	SelWAtn	Selected	Reselected	ArbLost	PhaseMM	SelTO
-------------	-------------	---------	----------	------------	---------	---------	-------

The bits in the Exception register are discussed in the next sections.

#### 4.4.6.1 SelWAtn Bit

The SelWAtn bit is set if the MESH controller was selected as a target and the Atn signal on the SCSI bus was asserted.

#### 4.4.6.2 Selected Bit

The Selected bit is set if the controller is selected as a target and the Atn signal was not asserted at the time of the selection. When this bit becomes true, the SCSI bus IDs of the target and host are in the last byte of the FIFO.

#### 4.4.6.3 Reselected Bit

The Reselected bit is set if the controller is reselected as a host. When this bit is true, the SCSI bus IDs of the target and host are in the last byte of the FIFO.

#### 4.4.6.4 ArbLost Bit

The ArbLost bit indicates that arbitration was lost when executing the Arbitration command. This exception may represent one or more lost arbitrations, depending on the hardware implementation.

#### 4.4.6.5 PhaseMM Bit

The Phase Mismatch (PhaseMM) bit is set when the pending phase command expects a certain phase but another phase is driven by the target. When this bit is set, the software must interrogate the BusControl0 register to determine the current phase of the bus and take action appropriately.

#### 4.4.6.6 SelTO Bit

The SelTO bit indicates that the time indicated by the SelectionTO register was exhausted before the destination device was selected.

### 4.4.7 Error Register

The Error register contains bits that indicate true error conditions. This register is cleared by writing ones to the bits to be cleared or by writing a 1 to the Error bit in the Interrupt register. The meanings of the bits in the Exception register are given in Figure 28.

Figure 28. Error register format

<i>rsvd</i>	UnExpDisc	SCSIRst	SeqErr	<i>ParityErr3</i>	<i>ParityErr2</i>	<i>ParityErr1</i>	ParityErr0
-------------	-----------	---------	--------	-------------------	-------------------	-------------------	------------

The bits in the Error register are discussed in the next sections.

#### 4.4.7.1 UnExpDisc Bit

The UnExpDisc bit indicates that the target released the Bsy signal during the time interval between successful selection and issuing the BusFree command.

#### 4.4.7.2 SCSIRst Bit

The SCSIRst bit indicates that the Rst signal was (or is) asserted. After receiving this interrupt, the SCSI bus Rst signal (bit 7 of Bus Status1) must be polled until the Rst signal is deasserted.

#### 4.4.7.3 SeqErr Bit

The Sequence Error bit indicates that a command was issued to the MESH controller while an exception or error interrupt was pending.

#### 4.4.7.4 ParityErr Bits

The ParityError0 bit is set if the calculated parity of the incoming data does not match the Parity0 bit on the SCSI bus.

The ParityErr3..ParityError1 bits are reserved for future use with SCSI implementations that use up to 32 parity error bits.

## 4.4.8 Interrupt Mask Register

The Interrupt Mask register is used to mask out interrupts from any or all of the interrupt sources. The value of the register is all zeros upon reset. A value of 0 in a particular location masks (disables) the corresponding interrupt source. A value of 1 enables interrupts from the source. The meanings of the bits in the Exception register are given in Figure 29.

Figure 29. Interrupt Mask register format

<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	ErrMask	ExcMask	CDoneMask
-------------	-------------	-------------	-------------	-------------	---------	---------	-----------

## 4.4.9 Interrupt Register

The Interrupt register combines the Error, Exception, and CmdDone status bits into one interrupt source. The meanings of the bits in the Interrupt register are given in Figure 30.

Figure 30. Interrupt register format

<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	<i>rsvd</i>	Error	Exception	CmdDone
-------------	-------------	-------------	-------------	-------------	-------	-----------	---------

The bits in the Interrupt register are discussed in the next sections.

### 4.4.9.1 Error Bit

The Error bit represents an OR-combination of all the bits in the Error register. It can be cleared by writing ones to the appropriate bits in the Error register or by writing 1 to this bit in the Interrupt register.

### 4.4.9.2 Exception Bit

The Exception bit represents an OR-combination of all the bits in the Exception register. It can be cleared by writing ones to the appropriate bits in the Exception register or by writing 1 to this bit in the Interrupt register.

### 4.4.9.3 CmdDone Bit

The CmdDone bit indicates that the last command issued has been completed. This bit is automatically cleared by issuing another command to the Command register. It can also be cleared by writing 1 to this bit in the Interrupt register.

After a reset action, the CmdDone signal is asserted. Because the Interrupt Mask register is cleared on reset, no interrupts are generated.

### 4.4.10 SourceID Register

The SourceID register contains the SCSI bus ID used for arbitration. It is also the register that the MESH controller responds to during the Selection phase (in target mode) or Reselection phase (in initiator mode).

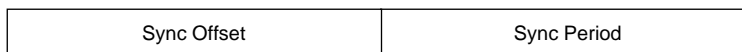
### 4.4.11 DestinationID Register

The DestinationID register contains the SCSI bus ID of the device to be selected or reselected.

### 4.4.12 SyncParms Register

The SyncParms register contains two fields, as shown in Figure 31.

Figure 31. SyncParms register format



The two SyncParms register fields are discussed in the next sections.

#### 4.4.12.1 SyncOffset Field

The SyncOffset field contains the depth to which the FIFO is capable of accepting synchronous data without delay. In the case of the MESH controller, this value may be from 1 to 15. If it is zero (as upon reset) then all transfers are done asynchronously. See the ANSI X3T9.2 *SCSI-3 SCSI Parallel Interface (SPI)* specification for the complete definition of synchronous data transfer mode and the corresponding synchronous data transfer request (SDTR) and related negotiations that must be done before synchronous data mode is enabled. The ANSI specification is listed in the bibliography.

When the synchronous offset value in the SyncParms register is nonzero, the value in the SyncPeriod field is used to specify the period of the Ack signal output. When the synchronous offset value is 0, asynchronous transfer mode is specified. In addition, the value in the SyncPeriod field specifies the minimum number of clock cycles between the negation of Ack and the time when the Req signal should be evaluated. Because of the dual-ranked synchronizers in hardware, the actual time may be greater by as much as one clock cycle. For example, if the SCSI bus is known to settle in 80 ns, a value of 4 in the SyncPeriod field guarantees that the Req or Ack signals are not evaluated in the state machine for at least 80 ns.

#### 4.4.12.2 SyncPeriod Field

The SyncPeriod field contains the period of the Req or Ack pulse when the MESH controller is in synchronous data transfer mode. The value must be adjusted to match the SyncPeriod time agreement in the SDTR messages. Although the ANSI SCSI specification allows resolution down to 4 ns, the driving software must negotiate a period that is acceptable to both the controller and the target device. A value of 0 indicates 10 MB per second, and 3 indicates 5 MB per second. The equation for determining the period of the synchronous Ack signal is  $(4 * clk + 2 * clk * SyncPer)$  where *clk* represents one MESH Clk period and SyncPer is the value in the lower nybble of the SyncParms register. Fast synchronous operation is a special case, where 0 means 100 ns.

For asynchronous transfer modes, the SyncPeriod field must be set to a value of 2 or greater. This field is used to mask the trailing edge of the Req signal so that the internal logic does not incorrectly respond to SCSI bus noise created by cable reflections and other impedance mismatch side-effects. The SyncPeriod value establishes the number of clocks delay after the trailing edge of Req before the MESH controller evaluates the Req signal again.

The minimum value in the SyncPeriod field is 2 for asynchronous transfers. Anything less results in shorter setup times for data relative to Ack on writes. Achieving the fastest asynchronous transfer rate with the MESH controller requires a SyncPeriod value of 2.

#### 4.4.12.3 Synchronous Data Transfers

Observe these cautions when performing synchronous data transfers with the MESH controller:

- The MESH controller is not able to handle unsolicited synchronous data. Before a command is issued, the SyncParms register must be set to match the expected transfer mode.

- Target mode does not support synchronous data transfers, regardless of the SyncParms value.
- The count for the Command Descriptor Block may not be arbitrarily set to a value higher than the number of bytes sent. If this happens, the CmdDone bit is set with a corresponding exception indicating a phase mismatch (the target went from command phase to data phase unexpectedly). If the target then begins to transfer data in synchronous mode, the MESH controller will not be able to handle the Req signals or any corresponding incoming data.

#### 4.4.13 MESH ID Register

The MESH ID register is used to indicate the version of the MESH controller and also to differentiate MESH from other controllers. This register is read-only; its current value is 0xE2.

#### 4.4.14 Selection TimeOut Register

The Selection TimeOut register is used for selection and reselection timeouts with its 8-bit value representing the amount of time before the selection timeout is reached. Each count represents 10 ms when the controller is running at 50 MHz.

### 4.5 Signal Interface

**Requirement 4-5:** The signal interface to the MESH SCSI controller must conform to Section 4.5.

**Note:** For reference, the MESH SCSI controller is used in what would be the Mode 1 interface mode of an NCR 53C96 chip.

#### 4.5.1 Control and Status Signals

The MESH SCSI controller's interface with the system is shown in Table 56. The following additional specifications apply:

- All signals are TTL-level, as specified in Section 4.3.
- Outputs must supply a minimum of 4 mA, open collector.

Table 56. Control and status signals

Name	Description	Logic States
<b>DMA Control</b>		
Dreq	DMA request indicating that the MESH controller is prepared to transfer data	Output
DAck	DMA acknowledge indicating that the DMA interface is prepared to transfer data	Input
<b>Register and DMA Interface</b>		
Addr[3:0]	4-line read/write register address bus	Input
DB[15:0]	16-line data bus; register accesses use bits 7:0, DMA uses all 16 bits	Tristate output
/Rd	Register read	Active-low input
/Wr	Register write	Active-low input
/CE	Chip enable	Active-low input
<b>Interrupt and Status</b>		
/Error	Indicates that the Error register is nonzero	Active-low output
/Exception	Indicates that the Exception register is nonzero	Active-low output
/CmdDone	Indicates that the last command issued is complete	Active-low output
/Int	Indicates that there was an error, an exception, or a command completion; /Int is asserted when the Interrupt register is nonzero and the appropriate mask bit is set in the Interrupt Mask register.	Active-low output
<b>Clock</b>		
Clk50	Clock signal; for 10 MB/sec fast SCSI performance, the clock rate should be 50 MHz	Input
<b>Test and Scan</b>		
/TestMode	If this signal is asserted and the ScanEnable signal is false, the outputs are disabled to allow for testing of other chips at the board level. If this signal is asserted and the ScanEnable signal is true, the MESH controller goes to Scan test mode. If Test-Mode is deasserted, all outputs behave normally.	Active-low input
ScanEnable	Besides producing the TestMode signal effects, the ScanEnable signal lets the device be scanned using scan chains.	Input

## 4.5.2 SCSI Bus Signals

The SCSI bus interface provided by the MESH SCSI controller is specified in Table 57. The following additional specifications apply:

- All signals are active low.
- All outputs must supply a minimum of 48 mA, open collector.
- Inputs are TTL with hysteresis.
- Signals must not present any load on the SCSI bus with or without power applied to the part.
- Signals must not create any noise on the SCSI bus when transitioning between power applied and power removed.
- Outputs must meet the ANSI X3T9.2 *SCSI-3 SCSI Parallel Interface (SPI)* specification for SCSI bus signals. This specification is listed in the bibliography. If there is a conflict or ambiguity between the definitions in this chapter and the ANSI specification, the ANSI specification takes precedence.

Table 57. SCSI bus signals

Name	Description
ioRst	SCSI bus reset. This signal causes all MESH outputs to be disabled.
ioBsy	Used for arbitration and selection phases. Generally indicates that a transaction is active.
ioSel	Used in the selection phase to select a destination device.
ioReq	Data request signals from target.
ioAck	Data acknowledge signals from initiator.
ioAtn	Bidirectional: used by initiator to request a message out phase.
ioCD	Bidirectional: command or data signal from target.
ioIO	Bidirectional: input/output signal from target.
ioMsg	Bidirectional: message signal from target.
ioData[7:0]	Data bus signals.
ioDataParity	Data bus parity (odd) signals.
RawReq	Data request input signal for use in synchronous modes.



The SCSI bus signals listed in Table 57 are defined in Table 58.

Table 58. SCSI bus signal definitions

Name	Description	Direction
ioRst	Used to reset the SCSI bus. All devices must release any signals within 100 ns of Rst asserted.	From any device.
ioBsy	Indicates bus activity. Used by initiator during arbitration and by target for all information transfer phases.	From any device during arbitration. From any device to acknowledge its selection or reselection. From a target only during information transfer phases.
ioSel	Used for selection or reselection phases only.	From any device that is attempting to select or reselect another device.
ioReq	Target information transfer handshake signal.	From a target only.
ioAck	Host (initiator) information transfer handshake signal.	From an initiator only.
ioAtn	Used by initiator to request that the target enter a Message Out phase.	From an initiator
ioCD ioIO ioMsg	Encoded to indicate one of six acceptable information transfer phases (see Table 59).	From a target only.
ioData[7:0]	Data bus; used for arbitration, selection, and information transfer phases.	If ioIO is true, the data is toward the initiator; if ioIO is false, it's toward the target.
ioDataParity	Odd parity—this bit is asserted when there are an even number (or zero) of signals asserted on Data[7:0].	If ioIO is true, parity is toward the initiator; if ioIO is false, it's toward the target.
RawReq	Wire-ORed with the ioReq signal; used as a synchronous data request.	From a target only.

### 4.5.3 Operating Phases

This section discusses the SCSI bus operating phases from the viewpoint of the host interface.

### 4.5.3.1 Phase Types

The SCSI bus exhibits two types of operating phases:

- Information transfer phases: Command, Status, MsgIn, MsgOut, DataIn, and DataOut. These phases are encoded by the ioCD, ioIO, and ioMsg signals, as shown in Table 59.
- Connection phases: Arbitration and selection (or reselection).

**Note:** Timing diagrams of sample arbitration and selection phases are given in Section 4.7.

In addition, the SCSI bus may be in a Bus Free phase, where the Busy and Select signals are false.

Table 59. Information transfer phases

ioMsg	ioCD	ioIO	Phase
0	0	0	DataOut
0	0	1	DataIn
0	1	0	Command
0	1	1	Status
1	0	0	<i>Reserved</i>
1	0	1	<i>Reserved</i>
1	1	0	MsgOut
1	1	1	MsgIn

### 4.5.3.2 Phase Expecting

The method for sequencing the MESH controller through a SCSI transaction is called *phase expecting*. With phase expecting, the low-level host interface knows which phases the SCSI bus must go through to complete a transaction. As long as the target enters the correct phase, the phase finishes with minimum support.

## 4.6 Timing

This section describes the timing relations and limits of the MESH SCSI controller signals.

**Requirement 4-6:** The timing characteristics of the MESH SCSI controller must be as described in Section 4.6.

**Note:** For reference, MESH register read/write and DMA timing characteristics must be the same as, or better than, those of the NCR 53C96 chip.

### 4.6.1 Clock Input

Clock input timing is diagrammed in Figure 32, with parameter limits specified in Table 60.

Figure 32. Clock input timing

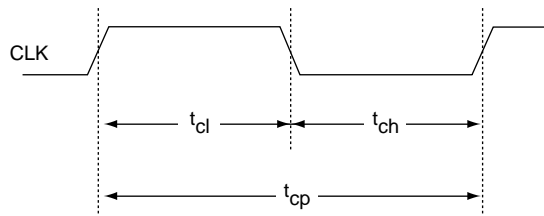


Table 60. Clock timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Frequency		45	50	MHz
High time	$t_{cl}$	9.5		ns
Low time	$t_{ch}$	9.5		ns
Period	$t_{cp}$	20		ns

The clock timing parameters shown in Table 60 are used to determine many of the timing parameters specified in the next sections.

## 4.6.2 Register Writes

Register write timing and parameter limits are shown in Figure 33 and Table 61.

Figure 33. Register write timing

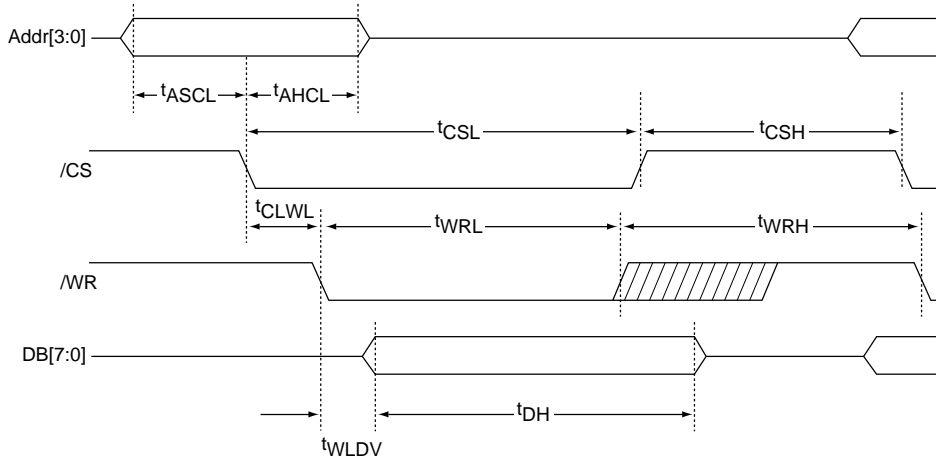


Table 61. Register write timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Address setup to CS low	$t_{ASCL}$	10		ns
Address hold from CS low	$t_{AHCL}$	10		ns
CS low to WR low	$t_{CLWL}$	0		ns
WR low to data valid	$t_{WLDV}$		$t_{cp}$	ns
Data hold	$t_{DH}$	$2 * t_{cp}$		ns
CS low	$t_{CSL}$	$2 * t_{cp}$		ns
CS high	$t_{CSH}$	$2 * t_{cp}$		ns
CS period	$t_{CSP}$	$4 * t_{cp}$		ns
WR low	$t_{WRL}$	$2 * t_{cp} + 10$		ns
WR high	$t_{WRH}$	$t_{cp} + 10$		ns
WR period	$t_{WRP}$	$4 * t_{cp}$		ns

### 4.6.3 Register Reads

Register read timing and parameter limits are shown in Figure 34 and Table 62.

Figure 34. Register read timing

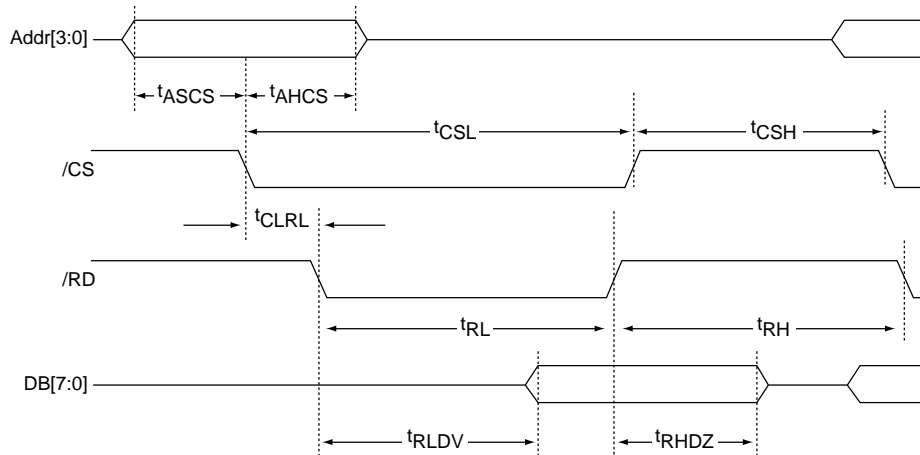


Table 62. Register read timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Address setup to CS low	$t_{ASCL}$	10		ns
Address hold from CS low	$t_{AHCL}$	10		ns
CS low to RD low	$t_{CLRL}$	0		ns
RD low to data valid	$t_{RLDV}$		$3 * t_{cp}$	ns
RD high to data high Z	$t_{RHDZ}$	10		ns
CS low	$t_{CSL}$	$2 * t_{cp}$		ns
CS high	$t_{CSH}$	$2 * t_{cp}$		ns
CS period	$t_{CSP}$	$4 * t_{cp}$		ns
RD low	$t_{RL}$	$t_{cp} + 10$		ns
RD high	$t_{RH}$	$t_{cp} + 10$		ns
RD period	$t_{RP}$	$4 * t_{cp}$		ns

## 4.6.4 DMA Writes

DMA write timing and parameter limits are shown in Figure 35 and Table 63.

Figure 35. DMA write timing

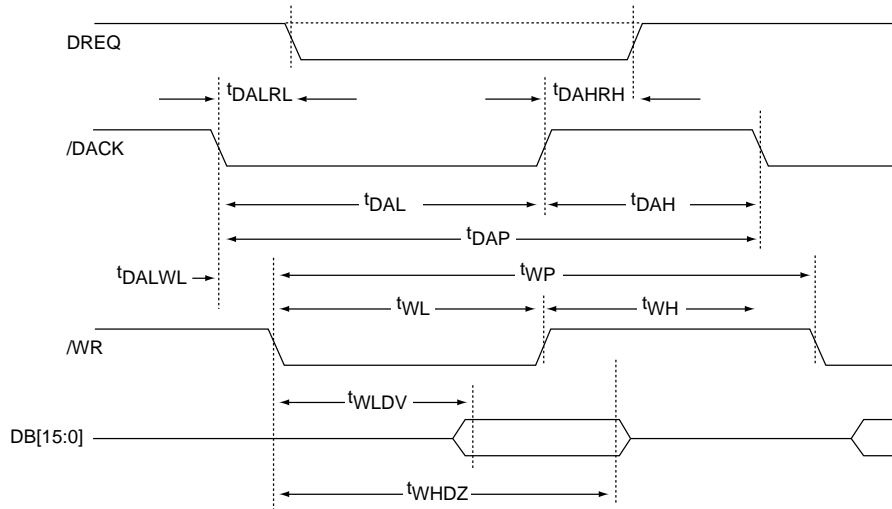


Table 63. DMA write timing parameters

Parameter	Symbol	Minimum	Maximum	Units
DAck low to DReq low	$t_{DALRL}$		$3 * t_{cp}$	ns
DAck high to DReq high	$t_{DAHRH}$	10		ns
DAck low	$t_{DAL}$	$2 * t_{cp}$		ns
DAck high	$t_{DAH}$	$t_{cp} + 10$		ns
DAck period	$t_{DAP}$	$4 * t_{cp}$		ns
DAck low to WR low	$t_{DALWL}$	0		ns
WR low to data valid	$t_{WLDV}$		$t_{cp} - 10$	ns
WR low to data high Z	$t_{WLDZ}$	$3 * t_{cp}$		ns
WR low	$t_{WL}$	$t_{cp} + 10$		ns
WR high	$t_{WH}$	$t_{cp} + 10$		ns
WR period	$t_{WP}$	$4 * t_{cp}$		ns

## 4.6.5 DMA Reads

DMA read timing and parameter limits are shown in Figure 36 and Table 64.

Figure 36. DMA read timing

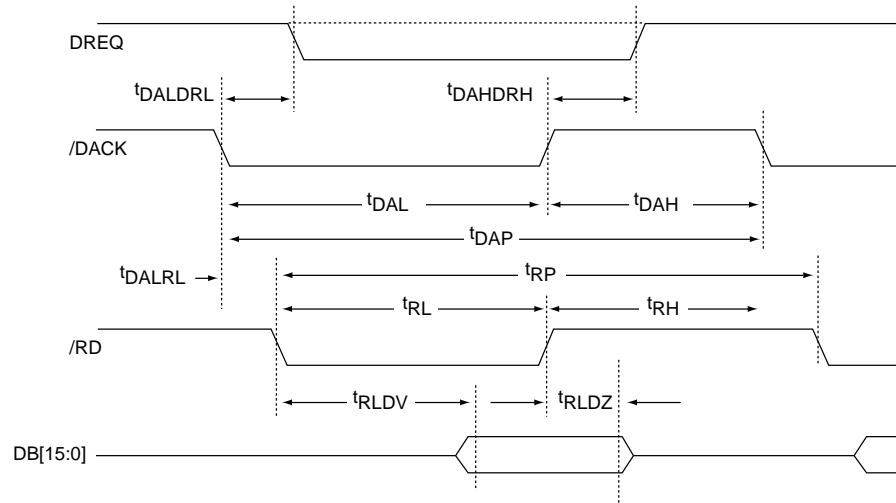


Table 64. DMA read timing parameters

Parameter	Symbol	Minimum	Maximum	Units
DACK low to DReq low	$t_{DALDRL}$		$3 * t_{cp}$	ns
DACK high to DReq high	$t_{DAHDRH}$	10		ns
DACK low	$t_{DAL}$	$2 * t_{cp}$		ns
DACK high	$t_{DAH}$	$t_{cp} + 10$		ns
DACK period	$t_{DAP}$	$4 * t_{cp}$		ns
DACK low to RDlow	$t_{DALRL}$	0		ns
RD low to data out valid	$t_{RLDV}$		$3 * t_{cp}$	ns
RD low to data high Z	$t_{RLDZ}$	10		ns
RD low	$t_{RL}$	$t_{cp} + 10$		ns
RD high	$t_{RH}$	$t_{cp} + 10$		ns
RD period	$t_{RP}$	$4 * t_{cp}$		ns

## 4.6.6 Interrupts

Interrupt timing and parameter limits are shown in Figure 37 and Table 65.

Figure 37. Interrupt timing

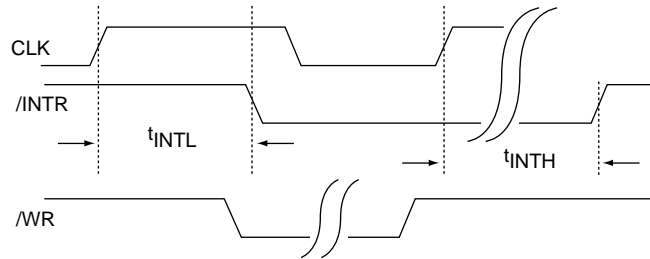


Table 65. Interrupt timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Interrupt low from CLK	$t_{INTL}$		10	ns
Interrupt high from Interrupt register write	$t_{INTH}$		$3 * t_{cp} + 10$	ns

## 4.6.7 Resets

Reset timing and cycle limits are shown in Figure 38 and Table 66.

Figure 38. Reset timing



Table 66. Reset timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Reset high time	$t_{RST}$	$2 * t_{cp}$		ns



## 4.6.8 Host Asynchronous Transmission

Host asynchronous transmit (SCSI DataOut) timing is diagrammed in Figure 39, with parameter limits specified in Table 67.

Figure 39. Host transmit timing

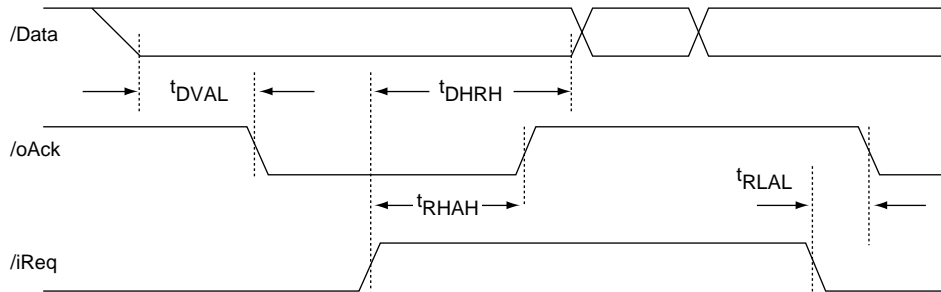


Table 67. Host transmit timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Data valid to Ack low	$t_{DVAL}$	55		ns
Data hold from iReq high	$t_{DHRH}$	$2 * t_{cp}$		ns
Req high to Ack high	$t_{RHAH}$	10		ns
Req low to Ack low	$t_{RLAL}$	10		ns

## 4.6.9 Host Asynchronous Reception

Host asynchronous receive (SCSI DataIn) timing is diagrammed in Figure 40, with parameter limits specified in Table 68.

Figure 40. Host receive timing

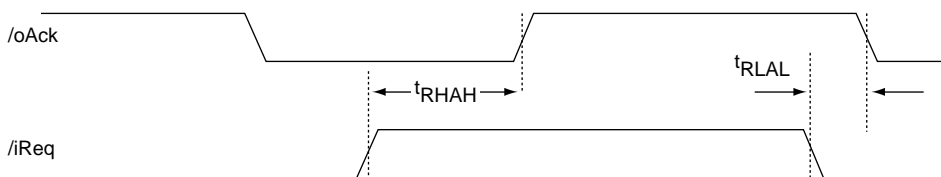


Table 68. Host receive timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Req high to Ack high	$t_{RHAH}$	10		ns
Req low to Ack low	$t_{RLAL}$	10		ns

### 4.6.10 Target Asynchronous Transmission

Target asynchronous transmit (SCSI DataIn) timing is diagrammed in Figure 41, with parameter limits specified in Table 69.

Figure 41. Target transmit timing

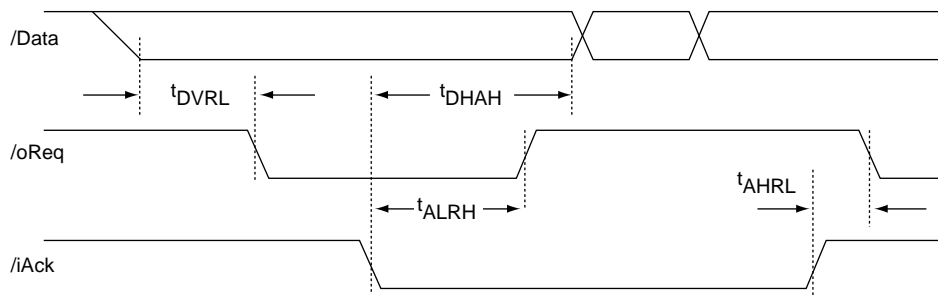


Table 69. Target transmit timing parameters

Parameter	Symbol	Minimum	Maximum	Units
Data valid to oReq low	$t_{DVRL}$	55		ns
Data hold from iAck high	$t_{DHAH}$	$2 * t_{cp}$		ns
iAck low to oReq high	$t_{ALRH}$	10		ns
iAck high to oReq low	$t_{AHRL}$	10		ns

### 4.6.11 Target Asynchronous Reception

Target asynchronous receive (SCSI DataOut) timing is diagrammed in Figure 42, with parameter limits specified in Table 70.

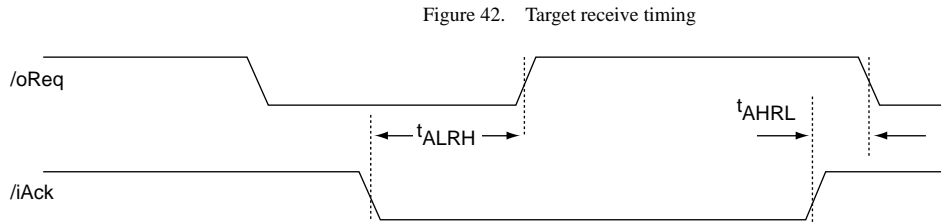


Table 70. Target receive timing parameters

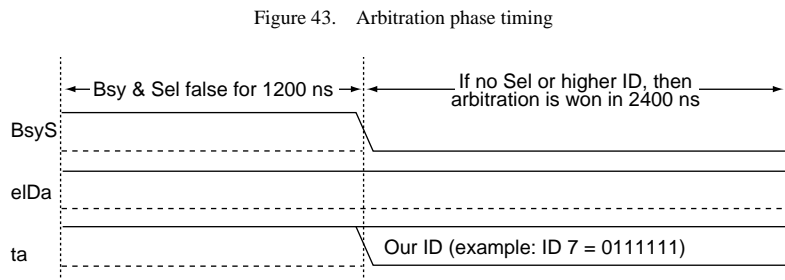
Parameter	Symbol	Minimum	Maximum	Units
iAck low to oReq high	$t_{ALRH}$	10		ns
iAck high to oReq low	$t_{AHRL}$	10		ns

## 4.7 Sample Phase Timing

This section contains sample timings of SCSI bus phases driven by the MESH SCSI controller.

### 4.7.1 Arbitration Phase

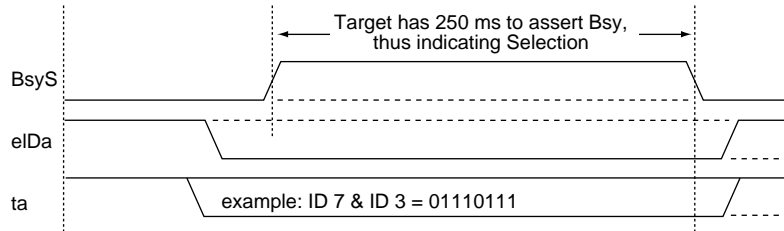
Sample timing of a SCSI bus arbitration phase is shown in Figure 43.



### 4.7.2 Selection Phase

Sample timing of a SCSI bus selection phase is shown in Figure 44.

Figure 44. Selection phase timing



## 4.8 Implementation Notes

This section provides implementation notes for the SCSI controller designer, including details of the external bus interface and a summary of differences between the MESH controller and the 53C9x family of chips.

### 4.8.1 SCSI Bus Connector

Power Macintosh computers use a SCSI bus for external SCSI devices and the internal CD-ROM drive. The external SCSI connector is a 25-pin D-type connector, such as a DB-25; the internal CD-ROM drive uses a 50-pin ribbon cable connector.

**Requirement 4-7:** The external SCSI connection must be a 25-pin D type connector with pin assignments as shown in Table 71.

#### 4.8.1.1 Pin Assignments

Table 71 shows the pin assignments for the internal and external SCSI connectors in Power Macintosh computers.

Table 71. SCSI connector pin assignments

Internal 50-Pin Connector	External 25-Pin Connector	Name	Description
2	8	/DB0	Bit 0 of SCSI data bus
4	21	/DB1	Bit 1 of SCSI data bus
6	22	/DB2	Bit 2 of SCSI data bus

Table 71. SCSI connector pin assignments (*continued*)

Internal 50-Pin Connector	External 25-Pin Connector	Name	Description
8	10	/DB3	Bit 3 of SCSI data bus
10	23	/DB4	Bit 4 of SCSI data bus
12	11	/DB5	Bit 5 of SCSI data bus
14	12	/DB6	Bit 6 of SCSI data bus
16	13	/DB7	Bit 7 of SCSI data bus
18	20	/DBP	Parity bit of SCSI data bus
25	n.a.	NC	No connection
26	25	TPWR	+5 V terminator power
32	17	/ATN	Attention
36	6	/BSY	Bus busy
38	5	/ACK	Handshake acknowledge
40	4	/RST	Bus reset
42	2	/MSG	Message phase
44	19	/SEL	Select
46	15	/C/D	Control or data
48	1	/REQ	Handshake request
50	3	/IO	Input or output
20, 22, 24, 28, 30, 34, and all odd pins except pin 25	7, 9, 14, 16, 18, and 24	GND	Ground

#### 4.8.1.2 Bus Termination

The external SCSI port in Power Macintosh computers provides automatic SCSI bus termination. When there are no external SCSI devices connected, the automatic termination is active. When one or more external SCSI devices are connected, the automatic termination is removed. As usual, the external SCSI device at the end of the SCSI bus requires termination. The internal end of the SCSI bus is terminated by a 110  $\Omega$  passive terminator.

## 4.8.2 Request and Acknowledge Signal Circuitry

Instead of using special filtering in the ASIC pad for Req and Ack signals, the MESH architecture supports an asynchronous filtered input and an unfiltered, synchronous input. The associated circuitry is intended to be implemented as shown in Figure 45 and Figure 46.

Figure 45. Req signal circuitry

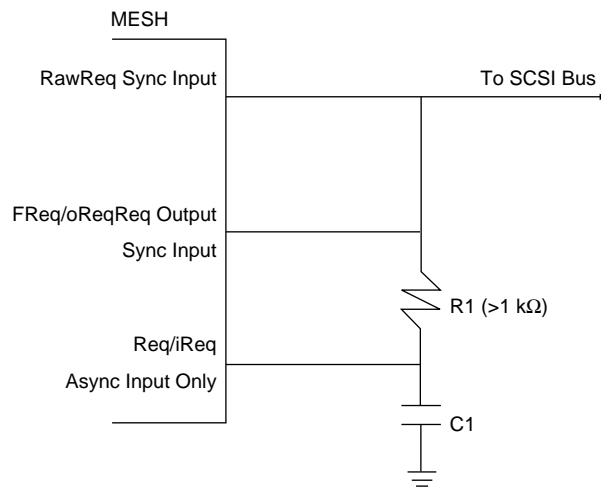
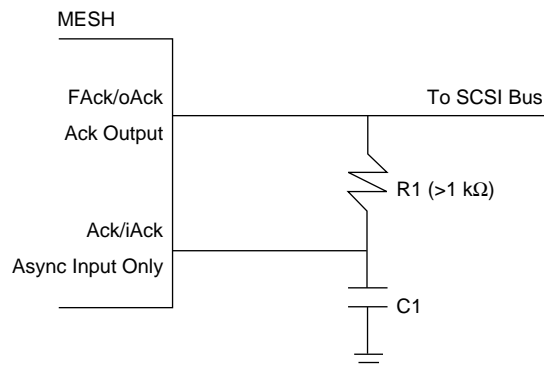


Figure 46. Ack signal circuitry



The RC time-constant of R1 and C1 should be set to a value appropriate to the worst-case cabling environment for the system. A value of 40 ns is typical.

**Note:** Since the MESH controller doesn't support synchronous transfers in target mode, there is no corresponding synchronous input circuitry for Ack.

### 4.8.3 Comparison of MESH and 53C9x

For reference, the following are some of the ways that the MESH SCSI controller defined in this chapter differs from controllers in the NCR 53C9x family of chips:

- The MESH controller has three outputs, called CmdDone, Exception, and Error. These outputs immediately notify the hardware when a command is done and whether it had any problem processing the command. CmdDone always goes active at the end of any command even if the command was terminated by an exception or error. This difference is important because if Exception and Error are both false when CmdDone goes true, then the DMA hardware can start the next activity immediately. In this way an entire transaction can be handled by relatively simple hardware and can be completed with only one interrupt at the final Bus Free phase.
- Every command (sequence) is autonomous. There are no commands, such as SelWithAtn or CmdCmpltSeq, that perform multiple activities for one command.
- Information transfer commands finish when either the count is exhausted, or there is some kind of exception or error. The 53C9x finishes when these conditions are met and the target switches to another phase. The MESH controller doesn't wait for a phase change.
- The Ack signal on the MESH SCSI bus is held active after all asynchronous information transfer phases. This lets software assert Atn, thereby causing a MsgOut phase.
- The FIFOCount register is properly debounced in the MESH controller so that it always correctly reflects the number of bytes in the FIFO.
- The MESH controller is always in the Mode1 hardware configuration. There are no mode signals.
- There are no Threshold 8, Burst Mode, or any other optional DMA transfer modes in the MESH controller. DMA is always 16 bits wide.

# Descriptor-Based DMA

# 5

## 5.1 Overview

This chapter defines the hardware and software operation of the Macintosh descriptor-based direct memory access (DBDMA) technology for data transfers to and from Macintosh I/O controllers.

**Note:** An Apple-designed DBDMA controller is part of the Mac I/O chip, as described in Section 2.9, “DBDMA Operation,” beginning on page 42.

The direct memory access (DMA) architecture described in this chapter is optimized for use on a computer’s main logic board. It is designed to be simple, yet efficiently support a significant number of logically distinct DMA channels.

The Macintosh DBDMA architecture uses *command descriptors*, which are structured as linked arrays. Since the base architecture supports memory-mapped interrupts, it is applicable to other 32-bit expansion bus DMA designs as well. Optional command extensions also support 64-bit address extensions. For the sake of simplicity, the scalability of this model has been limited; for example, there are no mechanisms for sharing DMA devices among processors or for efficiently reporting completion status from large numbers of DMA devices. A more complex model is needed in platforms containing large numbers of processors or I/O devices. For more information about command descriptors, see Section 5.6.

The DBDMA hardware architecture constrains the locations, format, and behavior of control registers and memory-resident data structures, but does not specify voltage levels or physical packaging. The software architecture provides the framework for the design of device drivers and other I/O software.



### 5.1.1 Design Objectives

The Macintosh descriptor-based DMA architecture was developed to achieve the following design objectives:

- **Existing bus standards.** The DBDMA architecture should be implementable on the motherboard or on an extension I/O bus such as NuBus or PCI.
- **Low cost.** The cost of the motherboard implementation should not be affected by the feature set necessary to support the DBDMA architecture on I/O extension buses.
- **Flexibility.** The DBDMA architecture should support the following system capabilities:
  - **Asynchronous events.** Asynchronous events, such as disk-media removals, should be reported without disrupting normal data transfers.
  - **Initiator independence.** To support a variety of bus and processor architectures, memory-mapped interrupts (32-bit writes to 32- or 64-bit addresses) should be implemented.
  - **Isochronous commands.** Data transfers to and from isochronous data streams should be supported.
- **Autonomous flow control.** A signaling mechanism should allow an external agent (or another channel) to regulate the progress of command list execution. This is needed to support operations such as beam-chasing for graphics applications, and would allow autonomous data transfers between devices that have different peak bandwidth capabilities.
- **High bandwidth applications.** The DBDMA architecture should provide excellent performance with very high bandwidth real-time streams. This requires minimizing the latency associated with data transfers.

### 5.1.2 Design Strategies

To achieve its design objectives, the Macintosh DBDMA architecture uses the following architectural design strategies:

- **Fixed-size commands.** All command entries are 16 bytes long, a size that is sufficient to specify command parameters and data-transfer addresses independently, as well as to provide fields for returned status information.
- **Fewest processor interrupts.** No more than one interrupt is needed per I/O operation.

- **Initiator-resident data structures.** All command and status list components are expected to be located in initiator-resident system memory, with the following benefits:
  - **Low cost.** System memory is more cost-effective than specialized memory in the host bus bridge or I/O device.
  - **Scalability.** The size of system memory is unlikely to limit the potential size of command and status list structures.
- **Simple command structure.** Descriptors are organized in a simple linear array. Linking can be accomplished through an optional branch action, either associated with a data transfer command or through a NOP command that specifies a branch.
- **Logically distinct channels.** To avoid dynamic usage conflicts, the following distinct DMA channels are provided for controlling transfers of independent data streams:
  - **Duplex channels.** Different DMA channels are provided for logically distinct data-transfer paths, such as Ethernet transmit and receive.
  - **Event channels.** A special DMA channel can be provided for reporting asynchronous events from any controller channel.
- **Conditional waits.** DMA commands can be conditionally suspended, based on the DMA controller's internal ChannelStatus bits. Since external access to certain of these state bits is provided, they can be used to temporarily suspend DMA activity until a flow-control conflict is resolved.
- **Embedded branches.** To minimize command latency in channel programs that require branching, the INPUT, OUTPUT, and NOP commands have a conditional branch capability.

### 5.1.3 Contents of This Chapter

The rest of this chapter contains the following major sections:

- Section 5.2, beginning on page 142, explains various design conventions followed by the DBDMA architecture and its documentation.
- Section 5.3, beginning on page 145, summarizes the way the Macintosh DBDMA works.
- Section 5.4, beginning on page 148, discusses the DBDMA design model, including the structure of the descriptor-based DMA components.

- Section 5.5, beginning on page 151, specifies the format and function of the DBDMA controller's registers.
- Section 5.6, beginning on page 162, specifies the format of DBDMA command entries.
- Section 5.7, beginning on page 173, specifies the formats of other memory-resident data structures.
- Section 5.8, beginning on page 173, summarizes the optional features of the DBDMA architecture.
- Section 5.9, beginning on page 174, provides examples of command-list programs.

## 5.2 Conventions

The discussion of DBDMA in this specification uses the conventions described in this section.

### 5.2.1 Terminology

The term *command descriptor* refers to DMA command list elements, which are often simply buffer descriptors. The term *DMA* indicates that data transfers are performed by a relatively simple state machine (called the *target*) that processes commands generated by a relatively sophisticated processor (called the *initiator*). These and other terms used in this chapter are defined in the glossary at the end of this book.

### 5.2.2 Bit and Byte Ordering

This chapter commonly refers to registers and memory locations that are 4 or 8 bytes in size. To ensure compatibility across bus standards, the ordering of the bytes within these locations is defined by their relative addresses, not their time slot or physical position on the bus. Bus bridges are similarly expected to route data bytes from one bus to another based on their addresses, not their physical position on a bus. The routing of data bytes based on their addresses is called *address invariance*.

To support an address-invariant model, this chapter specifies the mapping of data-byte addresses to bytes within multibyte registers and memory-resident data locations. However, the DBDMA architecture may be used with a variety

of bus standards, and each bus standard may have a preferred (big-endian or little-endian) byte-significance convention.

This chapter defines both big-endian and little-endian variants of the DBDMA architecture. These variants differ in the order in which bytes are located within quadlets, as described in the following two sections. The remainder of this chapter applies to both variants of the DBDMA architecture.

**Requirement 5-1:** DBDMA implementations must support both big-endian and little-endian byte ordering.

### 5.2.2.1 Big-Endian Variant

The big-endian variant of the DBDMA architecture is defined by the ordering of bytes in any quadlet (4 byte) register within the DMA unit or the command/status memory locations it accesses. For such quadlets, the data byte with the smallest address is the most significant, as shown in Figure 47.

Figure 47. Big-endian byte ordering

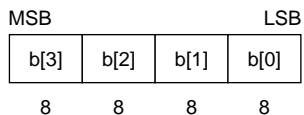


For all quadlet locations, the size of fields within the quadlet are specified—the bit position of each field is implied by the size of fields to its right or left. This convention is more compact than bit-position labels and avoids the question of whether 0 should be used to label the most or least significant bit.

### 5.2.2.2 Little-Endian Variant

The little-endian variant of the DBDMA architecture is defined by the ordering of bytes in any quadlet register within the DMA unit or the command/status memory locations it accesses. For such quadlets, the data byte with the smallest address is the least significant, as shown in Figure 48.

Figure 48. Little-endian byte ordering



### 5.2.3 Register, Field, and Constant Names

The formats of storage locations, field values, and DMA-command constants are defined in this chapter. To minimize confusion when these are referenced, a consistent nomenclature is used throughout. This nomenclature is defined in Table 72.

Table 72. Name notation examples

Sample Name	Description
CommandPtr	Register or element of a memory-resident data structure
Command.field commandDependent	Name of a field within a register or memory-resident data structure
CMD_VALUE	Name of a defined constant value

In names of registers and memory locations, the first letter is capitalized. For multiple-component names, like CommandPtr, the first letters of additional run-together words are also capitalized.

In names of fields, such as field in Command.field, the first letter is not capitalized. For multiple-component field names, like commandDependent, the first letters of additional run-together words are capitalized.

In names of defined constants, all letters within the name are capitalized. For multiple-component field names, like OUTPUT\_MORE, underscore characters separate the name components.

### 5.2.4 Reserved Values

Some of the registers and data structures defined in this chapter contain reserved fields, which are intended to support future extensions of the DBDMA architecture. In field diagrams, reserved fields are shaded and are labeled as reserved. When necessary, their names are abbreviated as res or r, as shown in Figure 49.

**Requirement 5-2:** Every DBDMA reserved field must be 0 when written and must be ignored when read. This must be true whether the reserved field is accessed by the DBDMA hardware or by device driver software.

Figure 49. Reserved field illustration

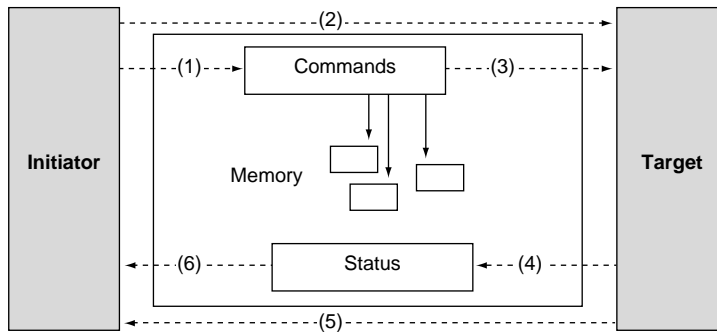


Some address pointers are constrained to be multiples of 4, 8, or 16 bytes. For example, the NewCommandPtr value is always 16-byte aligned. The least-significant bits of these aligned addresses are reserved, to ensure deterministic behavior when unaligned address values are used.

### 5.3 Summary of DBDMA Operations

DBDMA operations are expected to involve an initiator, which is the component that generates commands. In a typical operation, the initiator generates a new sequence of commands and appends them to a preexisting command list in memory. A wakeup signal is then sent to the target, to initiate the target’s processing of the recently appended commands. This process is illustrated in Figure 50.

Figure 50. DBDMA operation



A target is a component that connects to one or more I/O devices such as a disk controller or network interface. When the target is activated, it reads and executes the new commands. As command processing continues, the target returns status information to prespecified locations and sends a signal to the initiator, indicating that another descriptor has been processed.

There are no provisions for having multiple initiators concurrently append additional command entries; multiple initiators are expected to serialize their command list updates through other shared-memory semaphores that are beyond the scope of this specification. The unsharable nature of the command list implies that another architecture may be appropriate when the DMA controller is shared by loosely coupled multiprocessors.

Costs are reduced by eliminating the need to support status-queue structures. Status information is simply returned to prespecified locations within command descriptors, which are checked by the processor after an interrupt is received. The mandatory polling of multiple status locations implies that another architecture may be appropriate when large numbers of DMA controllers are shared by one processor.

### 5.3.1 Multiplexed Channels

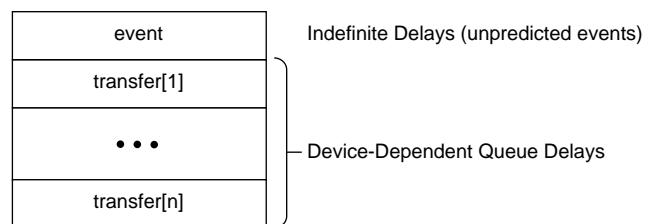
The DBDMA architecture assumes that each device is connected to its own data-transfer channel. I/O driver software is thus freed from the responsibility of dynamically assigning DMA channels to I/O devices. Two data-transfer channels would typically be assigned to a full-duplex device, such as Ethernet; one is sufficient for a half-duplex device such as a disk controller.

In most cases, the transfer of control information (such as a seek address) is performed before the data transfer is initiated. Similarly, status information (such as short-transfer counts) is returned after the data transfer finishes. In both cases, the control, data transfers, and control transfers are performed sequentially, so the same DMA channel resources can be shared.

Other forms of status information (such as a disk pack removal) are unexpected and cannot be associated with a data transfer channel. A special event channel may be used to report such asynchronous events.

Since the event channel is rarely used, it may be shared by all devices attached to a DBDMA controller. Hence the controller may contain one shared event channel and one or more device-connected data transfer channels, as shown in Figure 51.

Figure 51. Multiplexed channel types



The commands in the event list are expected to be Read commands that (on demand) return unexpected event status. In this case, the returned list-index value  $n$  is used by I/O driver software to identify the affected transfer list, transfer[ $n$ ]. Only one or two read commands are expected to be queued, since a queued event command is not preallocated to any list but can be used to return event status from any of the affiliated lists. Note that the commands may remain in the event list for an indefinite period of time, depending on the arrival rate of unexpected events.

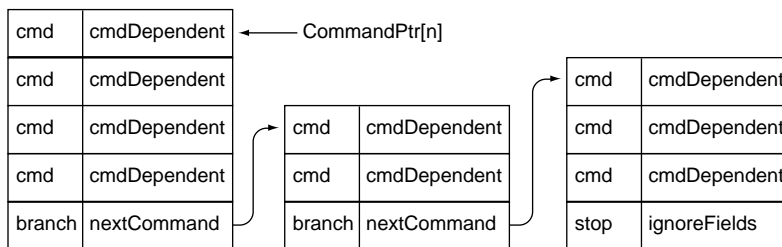
The commands in the transfer[1] through transfer[ $n$ ] lists are expected to be used individually (to support half-duplex traffic) or in pairs (to support full-duplex traffic). The commands in these lists are typically transient, in that they are constantly consumed as I/O operations are performed. However, some I/O operations (such as terminal read actions or flow-controlled write actions) may have an indefinite lifetime.

**Note:** The event channel is optional; it need not be implemented when the interrupt mechanism and device status registers are sufficient to report device-dependent asynchronous events.

### 5.3.2 Command List Structure

The command list space is allocated by the initiator, which fills a set of command-list entries before passing ownership of them to the target. The target fetches command entries and performs the command-entry-specified data-transfer operation, processing command entries until a STOP entry is reached. The structure of a typical command list is shown in Figure 52.

Figure 52. Command list structure



The command list may be structured in several different ways: to loop on itself (a circular queue), to link individual data-transfer operations (linked lists), or to link groups of data-transfer operations (a hybrid approach, as shown in Figure 52).



Note that branches may be embedded in most commands. Therefore, the branch shown in the example could be collapsed into the previous command or it could be encoded as a NOP command that specified a branch. The detailed structure of command lists is dependent on device driver software conventions and is not specified by the DBDMA architecture.

In the idle state, the target has a `CommandPtr[n]` value for each of the  $n$  multiplexed channels that are supported. When activated on channel  $n$ , a command entry is fetched from the physical address specified by `CommandPtr[n]`. Command fetching and execution normally continue until a `STOP` command is executed.

The autonomous fetching and execution of multiple commands provides a mechanism to execute different data-transfer operations (command chaining), provides several memory addresses for a single data-transfer operation (scattered memory addressing), or provides several target addresses for a single data-transfer operation (scattered device addressing).

During the execution of a command entry, the `CommandPtr[n]` value is either incremented by 16, replaced by the command-entry's `branchAddress` value, or left unchanged (for the `STOP` command value).

Additional commands may be dynamically appended to an existing command descriptor list while the previously initiated DMA operations are in progress. Command-list appending is performed by creating an additional command list and overwriting the `STOP` command at the end of the old command list with a `NOP` or a branch to the first command of the new list.

The `WAIT` field can be used to provide flow control within a channel. A channel can be programmed to wait until signaled by the processor or by another device.

## 5.4 Design Model

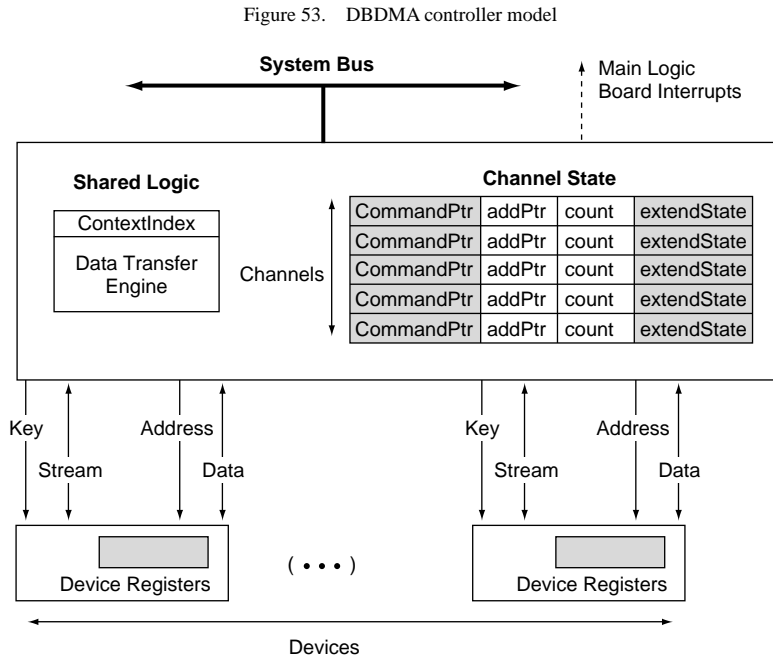
This section presents the DBDMA design model, which illustrates how the DBDMA components are structured.

### 5.4.1 Controller Components

A DBDMA controller is expected to connect multiple devices indirectly to a system bus that supports read and write transactions to system memory. A processor interrupt is either generated through special signal lines (for motherboard designs) or by sending a wakeup event (a 32-bit write to a prespecified register address).

The DBDMA controller may have shared states, as well as channel states that are replicated for each of several connected devices, as shown in

Figure 53. In Figure 53, the shaded elements support direct memory-mapped access and are optionally accessible through DBDMA.



Hardware is expected to use some form of the ContextIndex state to identify which of the DMA contexts is currently active, but this state is not directly or indirectly accessible to software. The data-transfer engine provides the facilities for performing the requested data transfers.

The DBDMA architecture provides a well-defined interface to attached I/O devices. An I/O device may be connected to a data-byte stream, where the controller handles the movement of unaddressed data bytes into or out of system memory. The controller provides a 3-bit key value. Four of the Command.key encodings are used to identify which byte-stream is being used. The standard data stream is typically STREAM0 and an optional control/status stream is typically STREAM1.

As an alternative option, a DBDMA controller may support the direct movement of data between device registers and memory. In this case, the controller provides a sequence of device register addresses while the data is being transferred. Although not needed for most devices, the device-address space may be as large as 64 bits. An additional Command.key encoding is used to identify transfers directed to device registers.

Each channel provides a few memory-mapped registers. These registers include the CommandPtr register (used to initialize the sequencer's pointer to the command list) and the ChannelStatus and ChannelControl registers. Writes to ChannelControl registers can reactivate idle command-list processing. Optional BranchSelect, InterruptSelect, and WaitSelect registers can be used to specify the manner in which the conditional execution of the branch, interrupt, and wait actions are selected.

## 5.4.2 System Bus Errors

Unrecoverable errors on system memory accesses includes parity and addressing errors. A busy-retry error is treated as a recoverable bus error, and the bus action is retried until it finishes successfully or the DBDMA controller's command processing is terminated.

An unrecoverable error sets the dead bit in the ChannelStatus register of the currently executing DMA channel. Software must explicitly reset the ChannelStatus.run bit to 0 and later set it back to 1 to return to an operational state.

## 5.4.3 Device Status

The device's control and status registers are mapped into system memory space, so the processor can access them directly using read and write transactions. For some applications, the processor is required to update these registers at the end of every data transfer, so there is no need for the channel program to return device status explicitly.

However, some devices have the intelligence to process multiple data transfer requests autonomously. For these applications, processor performance would be reduced unless the channel could autonomously return device status before initiating the next data transfer. The DBDMA architecture provides two mechanisms for autonomously returning device status:

- **Status stream.** An additional data-transfer command can be used to transfer a status-byte stream into memory. A unique Command.key value is used to distinguish the status stream from the normal data-transfer stream.
- **Status registers.** An additional data-transfer command can be used to transfer a range of device register addresses into memory. Another unique Command.key value is used to distinguish this memory-mapped transfer from the normal data-transfer stream.

**Note:** Simple device status information may also be returned in the Command.xferStatus field described in Section 5.6.1.9.

### 5.4.4 Conditional Actions

The INPUT, OUTPUT, and NOP commands provide `Command.b` and `Command.branchAddress` fields that specify the condition and address for optional branches in command sequencing. A similar mechanism is used to specify conditional interrupt generation and suspension of command processing.

## 5.5 Controller Registers

**Requirement 5-3:** The DBDMA controller's register interface must conform to the specifications of Section 5.5.

### 5.5.1 Register Organization

The DBDMA controller architecture uses two components that are directly visible to the processor:

- channel registers: a linear array of channel registers
- device registers: a data structure mapping directly to the supported device's registers

The base addresses of these data structures are defined by the affiliated bus standard, or within a ROM data structure whose format is specified by the affiliated bus standard, or in the chip or system specification (in the case of a motherboard implementation).

Each channel contains a set of registers that can be accessed directly through the system address space as well as indirectly through `STORE_QUAD` commands with a `Command.key` value of `KEY_REGS`. Several of these registers are optional, and serve as extensions to the functionality provided in the basic channel commands.

Table 73 shows the offset for each of the channel registers. The offset value appears literally in the `Command.address` field when the register is accessed indirectly (`Command.key = KEY_REGS`). When accessed directly by the host, or by a channel command (`Command.key = KEY_SYSTEM`) the register's address is the sum of the system-defined channel base address and the register's offset value.

Table 73. Channel registers

Offset	Register	Required/Optional
0	ChannelControl	Required
4	ChannelStatus	Required
8	CommandPtrHi	Optional
12	CommandPtrLo	Required
16	InterruptSelect	Optional
20	BranchSelect	Optional
24	WaitSelect	Optional
28	TransferMode	Optional
32	Data2PtrHi	Optional
36	Data2PtrLo	Optional
40	Reserved	Optional
44	AddressHi	Optional
48	BranchAddressHi	Optional
52–60	Reserved	Optional
64–124	Implementation-dependent	Optional

The functions of the mandatory channel control registers are as follows:

- ChannelControl and ChannelStatus are used to control and observe the activity of the channel.
- CommandPtr is used to indicate where the channel's command list is located in memory.

The initial values, read values, and write effects for the channel registers are shown in Table 74.

Table 74. DBDMA register summary

Register Name	Initial Value	Read Value	Write Effect
ChannelControl	n.a.	0x00000000	Update selected bits
ChannelStatus	0x00000000	ChannelStatus	Ignored
CommandPtrHi	Undefined	CommandPtrHi	Store if idle
CommandPtrLo	Undefined	CommandPtrLo	Store if idle
InterruptSelect	Undefined	InterruptSelect	Store
BranchSelect	Undefined	BranchSelect	Store
WaitSelect	Undefined	WaitSelect	Store
Data2PtrHi	Undefined	Data2PtrHi	Store
Data2PtrLo	Undefined	Data2PtrLo	Store
TransferMode	Undefined	TransferMode	Store
AddressHi	Undefined	AddressHi	Store
BranchAddressHi	Undefined	BranchAddressHi	Store

The next sections provide detailed descriptions of the DBDMA registers.

## 5.5.2 ChannelControl Register

The format of the ChannelControl register is shown in Figure 54. The ChannelControl register serves as the writing port for bits that are presented in the ChannelStatus register.

Figure 54. ChannelControl register format



### 5.5.2.1 ChannelControl.mask Field

The ChannelControl.mask field selects which of the lower 16 bits are to be modified by a write action. Bits in the lower half of the ChannelControl register are written only if the corresponding bits in ChannelControl.mask are set.

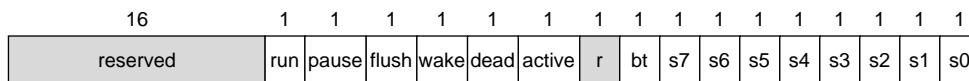
### 5.5.2.2 ChannelControl.data Field

The value in the ChannelControl.data field is conditionally written to the ChannelStatus register, based on the ChannelControl.mask field. Note that certain bits of the ChannelStatus register are not writeable or should be written only to a set state or to a cleared state. See the next section for full details.

## 5.5.3 ChannelStatus Register

The read-only ChannelStatus register provides access to the channel's internal status bits. The ChannelControl register bits are also visible through the ChannelStatus register. A write to the ChannelStatus register is ignored. The format of the ChannelStatus register is shown in Figure 55.

Figure 55. ChannelStatus register format



The run and pause bits are control bits that are set and cleared by software. The flush and wake bits are command bits that are set by software and are cleared by hardware when a given action has been performed. The dead, active, and bt bits are hardware status bits. The bits s7..s0 can be used for general purpose status and control. Their meaning is channel-specific, and they can be controlled by either hardware or software.

### 5.5.3.1 ChannelStatus.run Bit

ChannelStatus.run is set to 1 by software to start execution by the channel. This should be done only after the CommandPtr registers are initialized; otherwise, the operation of the channel is undefined.

Software can clear this bit to 0 to abort the operation of the channel. When channel operation is aborted data transfers are terminated, status is returned, and an interrupt is generated (if requested in the Command.i field of the command descriptor). Data stored temporarily in channel buffers may be lost.

### 5.5.3.2 ChannelStatus.pause Bit

ChannelStatus.pause is set to 1 by software to suspend command processing. Hardware responds by suspending data transfers and command execution and then clearing the ChannelStatus.active bit.

Software must reset ChannelStatus.pause to 0 to allow command processing to resume.

### 5.5.3.3 ChannelStatus.flush Bit

ChannelStatus.flush can be set to 1 by software, to force a channel that is executing an INPUT\_MORE or INPUT\_LAST command to update memory with any data that has been received from the device but has not yet been written to memory.

When the memory update is complete, hardware updates the xferStatus and resCount fields in the current memory resident channel command, then clears the flush bit. Thus, a partial status update is characterized by 1 in the flush bit of the xferStatus field and a final status update is characterized by 0.

### 5.5.3.4 ChannelStatus.wake Bit

ChannelStatus.wake is set to 1 by software to cause a channel that has gone idle to wake up, refetch the command pointed to by CommandPtr, and continue processing commands. The channel becomes idle after executing a STOP command. The STOP command does not increment the CommandPtr register.

ChannelStatus.wake is reset to 0 by hardware immediately after each command fetch.

### 5.5.3.5 ChannelStatus.dead Bit

ChannelStatus.dead is set to 1 by hardware when the channel halts execution due to a catastrophic event such as a bus or device error. The current command is terminated and hardware attempts to write status back to memory. Further commands are not executed. If a hardwired interrupt signal is implemented, the controller generates an unconditional interrupt. When hardware sets the ChannelStatus.dead bit, the ChannelStatus.active bit is simultaneously deasserted.

Hardware resets ChannelStatus.dead to 0 when the ChannelStatus.run bit is cleared by software.



### 5.5.3.6 ChannelStatus.active Bit

ChannelStatus.active is set to 1 by hardware in response to software setting the ChannelStatus.run bit to 1.

ChannelStatus.active is reset to 0 by hardware in response to software clearing the ChannelStatus.run bit or setting the ChannelStatus.pause bit. It is also reset to 0 after a STOP command is executed or when hardware sets the ChannelStatus.dead bit to 1.

### 5.5.3.7 ChannelStatus.bt Bit

ChannelStatus.bt is set by hardware at the completion of the NOP, INPUT, and OUTPUT commands to indicate whether a branch has been taken. Branching is governed by the Command.b field in the command descriptor, the ChannelStatus.s7..s0 bits, and the values in the mask and value fields of the BranchSelect register. The presence of this bit in the Command.xferStatus field allows software to follow the actual channel program flow with minimal overhead.

### 5.5.3.8 ChannelStatus.s7..s0 Bits

Each DMA channel has up to eight general purpose state bits (ChannelControl.s7..ChannelControl.s0) that can be written through the ChannelControl register and read through the ChannelStatus register. In some channel implementations, it may be desirable to provide an additional method for setting and clearing these bits directly through hardwired connections—for example, when errors occur or to indicate the completion of a logical record. These bits can be tested at the completion of each command to determine if certain actions should be taken. These actions may include generation of a hardwired interrupt signal, suspension of further command processing, and branching to a new location for further command fetches.

Where applicable, the following conventions for the use of the general-purpose status bits should be followed:

- In implementations where some of these bits are controlled by software and some bits are controlled by hardware, the least significant bits should be those controlled by software and the most significant bits should be those controlled by hardware.
- In implementations that define a hardware-controlled bit to be an error flag, it should be the most significant bit.
- In implementations that define a hardware-controlled bit to be an end-of-record flag, it should be the second most significant bit.

- In implementations where one of the software-controlled bits is used as the predominant method for controlling the wait function, the least significant bit should be used.

### 5.5.4 CommandPtr Registers

The CommandPtrHi and CommandPtrLo registers specify the address of the next command entry to be fetched. Since all channel commands are 16-byte aligned, the least-significant bits of the CommandPtrLo register must always be written with zeros. If they are written with a nonzero value, the operation of the channel is indeterminate and the value it returns is undefined.

The CommandPtr registers can be read at any time, but writes to the CommandPtr registers are ignored unless the ChannelStatus.run and ChannelStatus.active bits are both 0.

**Note:** The CommandPtrHi register is optional; it is part of the 64-bit DBDMA address extensions.

### 5.5.5 InterruptSelect Register

The InterruptSelect register is used to generate the interrupt condition bit that is tested at the completion of each command to determine if a hardwired interrupt signal should be asserted. Its format is shown in Figure 56.

Figure 56. InterruptSelect register



The two 8-bit res fields are reserved. Software writes a 0 value; hardware ignores the data value that is written. When the register is read, the values returned in these fields are undefined.

The 8-bit mask and value fields affect the generation of interrupts when command descriptors are processed.

An interrupt condition signal, which can be tested by each channel command, is generated according to the following algorithm:

```
c = (ChannelStatus.s7..s0 & InterruptSelect.mask)
  == (InterruptSelect.value & InterruptSelect.mask)
```

**Note:** The InterruptSelect register is optional. Depending on the needs of the specific channel, from 0 to 8 bits may be implemented in both the mask and value fields. If this register is not implemented, the channel returns 0 as the interrupt condition bit and channel commands cannot specify conditional assertion of hardwired interrupts.

### 5.5.6 BranchSelect Register

The BranchSelect register is used to generate the branch condition bit that is tested at the completion of each command to determine if a branch should be performed. The BranchSelect register has the same format as the InterruptSelect register.

The branch condition signal is generated in accordance with the following algorithm:

```
c = (ChannelStatus.s7..s0 & BranchSelect.mask)
   == (BranchSelect.value & BranchSelect.mask)
```

**Note:** The BranchSelect register is optional. Depending on the needs of the specific channel, from 0 to 8 bits may be implemented in both the mask and value fields. If this register is not implemented, the channel returns 0 as the branch condition bit and channel commands cannot specify conditional branching. Conditional branching may be particularly useful if the channel defines one of the general purpose status bits to be an error indication.

### 5.5.7 WaitSelect Register

The WaitSelect register is used to generate the wait condition bit that is tested at the completion of each command to determine if command execution should be suspended. The WaitSelect register has the same format as the InterruptSelect register.

The wait condition signal is generated according to the following algorithm:

```
c = (ChannelStatus.s7..s0 & WaitSelect.mask)
   == (WaitSelect.value & WaitSelect.mask)
```

**Note:** The WaitSelect register is optional. Depending on the needs of the specific channel, from 0 to 8 bits may be implemented in both the mask and value fields. If this register is not implemented, the channel returns 0 as the wait condition bit and channel commands cannot specify conditional suspension of command execution.

### 5.5.8 Data2Ptr Registers

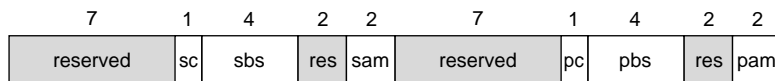
The Data2PtrHi and Data2PtrLo registers specify the secondary 64-bit base address for data transfers from device register space (Command.key value of KEY\_DEVICE), from a second system memory space (Command.key value of KEY\_SYSTEM), or from channel registers (Command.key value of KEY\_REGS).

**Note:** The Data2Ptr registers are optional. Only a subset of the Data2PtrLo register bits need to be implemented if a small range of device register addresses is supported.

### 5.5.9 TransferMode Register

The TransferMode register specifies access mode parameters for both the primary address (Command.address) and the secondary address (Data2Ptr). The format is shown in Figure 57.

Figure 57. TransferMode register



**Note:** The TransferMode register is optional. It is reserved for specifying special data-transfer modes, to distinguish coherent from uncached access, or to restrict the block-transfer lengths. The values for any unimplemented fields of this register default to 0.

Table 75 lists the functions of the TransferMode register bits. These bits are described in the next sections.

Table 75. TransferMode register bits

Name	Address	Description
pc	Primary	Coherency
pbs	Primary	Data transfer block size
pam	Primary	Address mode

Table 75. TransferMode register bits (*continued*)

Name	Address	Description
sc	Secondary	Coherency
sbs	Secondary	Data transfer block size
sam	Secondary	Address mode

### 5.5.9.1 TransferMode.pam and TransferMode.sam Fields

The TransferMode.pam and TransferMode.sam fields specify whether the primary and secondary addresses increment, decrement, or remain fixed as a transfer proceeds, as specified in Table 76.

Table 76. Address mode encoding

Value	Name	Description
0	ADDR_INC	Address increments
1	ADDR_DEC	Address decrements
2	ADDR_FIX	Address remains fixed
3		Reserved

### 5.5.9.2 TransferMode.pbs and TransferMode.sbs Fields

The TransferMode.pbs and TransferMode.sbs fields specify the block size for transfers, as specified in Table 77.

Table 77. Block size encodings

Value	Name	Block Size
0	BLK_DEFAULT	System-dependent
1		Reserved
2		4
3		8

Table 77. Block size encodings (*continued*)

Value	Name	Block Size
4		16
5		32
6		64
7		128
8		256
9		512
10		1024
11		2048
12		4096
13		8192
14		16384
15		32768

The channel may not support transfer block sizes as large as those specified by software in the `TransferMode.pbs` or `TransferMode.sbs` fields. In these cases, the largest of the smaller supported block transfer sizes is used.

### 5.5.9.3 `TransferMode.pc` and `TransferMode.sc` Fields

The `TransferMode.pc` and `TransferMode.sc` fields specify the coherency mode for DBDMA transfers. A value of 0 specifies that coherent transfers are to be performed. A value of 1 permits non-coherent transfers. With a zero value of `TransferMode.pc`, the `TransferMode.pbs` field is ignored. With a zero value of `TransferMode.sc`, the `TransferMode.sbs` field is ignored.

## 5.5.10 `AddressHi` Register

The `AddressHi` register is used to extend the 32-bit `Command.address` field to 64 bits. In 64-bit implementations it should be loaded through a `STORE_QUAD` command (`Command.key = KEY_REGS`, `Command.address = ADDRESS_HI`) preceding all data transfer commands. On the initial execu-

tion phase of a data transfer command (OUTPUT\_MORE, OUTPUT\_LAST, INPUT\_MORE, INPUT\_LAST, and STORE\_QUAD or LOAD\_QUAD when Command.key = KEY\_SYSTEM or KEY\_DEVICE) the value of AddressHi is loaded into the upper 32 bits of the channel's data buffer address register (DataPtrHi).

**Note:** The AddressHi register is optional; it is part of the 64-bit address extensions.

### 5.5.11 BranchAddressHi Register

The BranchAddressHi register is used to extend the 32-bit Command.branchAddress field to 64 bits. In 64-bit implementations it should be loaded through a STORE\_QUAD command (Command.key = KEY\_REGS, Command.address = ADDRESS\_HI) preceding all commands that specify a branch. When the branch is taken, the value of BranchAddressHi is loaded into CommandPtrHi.

**Note:** The BranchAddressHi register is optional; it is part of the DBDMA 64-bit address extensions.

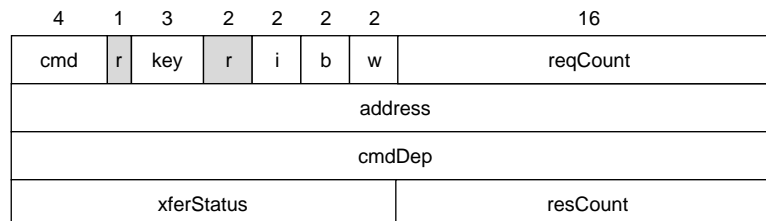
## 5.6 Commands

**Requirement 5-4:** The format and operation of of DBDMA command entries must conform to Section 5.6.

### 5.6.1 Command Formats

The general format of a DBDMA command descriptor contains 16 bits of opcode fields (including a 4-bit cmd field), a 16-bit reqCount field, a 32-bit address parameter, and a 32-bit cmdDep parameter, as shown in Figure 58.

Figure 58. Command entry format



The `xferStatus` and `resCount` fields are used by the channel to report status after a command finishes. Although the location and size of the `cmdDep` field is standardized for all commands, its interpretation is dependent on the `Command.cmd` field value.

The effects of command execution are described in the following sections.

### 5.6.1.1 Command.cmd Field

The `Command.cmd` field specifies which type of data transfer is performed, as defined in Table 78 and described in the following sections.

Table 78. `Command.cmd` field values

Value	Name	Description
0	OUTPUT_MORE	Transfer more memory to stream
1	OUTPUT_LAST	Transfer last memory to stream
2	INPUT_MORE	Transfer more stream to memory
3	INPUT_LAST	Transfer last stream to memory
4	STORE_QUAD	Store immediate 4-byte value
5	LOAD_QUAD	Load immediate 4-byte value
6	NOP	No data transfer
7	STOP	Suspend command processing
8–15		Reserved

### 5.6.1.2 Command.key Field

The `Command.key` field is used to select which of the device access ports is used, as specified in Table 79.

Table 79. Effects of `Command.key` field

Value	Name	Description	Usage
0	KEY_STREAM0	Default device stream	Data
1	KEY_STREAM1	Device-dependent stream	Control and status



Table 79. Effects of Command.key field (*continued*)

Value	Name	Description	Usage
2	KEY_STREAM2	Device-dependent stream	
3	KEY_STREAM3	Device-dependent stream	
4		Reserved	
5	KEY_REGS	Channel-state register space	
6	KEY_SYSTEM	System memory-mapped space	
7	KEY_DEVICE	Device memory-mapped space	

With INPUT and OUTPUT commands, the key field may specify alternate device streams (for example, data and status) as well as alternate secondary address spaces when two-address transfers are to be performed (in conjunction with Data2Ptr).

With LOAD\_QUAD and STORE\_QUAD commands, the key field specifies to which address space the immediate data is to be transferred.

Table 80 summarizes the operation of the key field in combination with each of the data transfer commands.

Table 80. Data transfer operations

Command	Key	Source	Destination
OUTPUT	KEY_STREAM0..3	SysMem(address)	Stream 0..3
	KEY_SYSTEM	SysMem(address)	SysMem(Data2Ptr)
	KEY_DEVICE	SysMem(address)	Device(Data2Ptr)
	KEY_REGS	SysMem(address)	ChannelRegs(Data2Ptr)
INPUT	KEY_STREAM0..3	Stream 0..3	SysMem(address)
	KEY_SYSTEM	SysMem(Data2Ptr)	SysMem(address)
	KEY_DEVICE	Device(Data2Ptr)	SysMem(address)
	KEY_REGS	ChannelRegs(Data2Ptr)	SysMem(address)

Table 80. Data transfer operations (*continued*)

Command	Key	Source	Destination
STORE_QUAD	KEY_STREAM0..3	Illegal; see note below	See note below
	KEY_SYSTEM	data32	SysMem(address)
	KEY_DEVICE	data32	Device(address)
	KEY_REGS	data32	ChannelRegs(address)
LOAD_QUAD	KEY_STREAM0..3	Illegal; see note below	See note below
	KEY_SYSTEM	SysMem(address)	data32
	KEY_DEVICE	Device(address)	data32
	KEY_REGS	ChannelRegs(address)	data32

**Note:** A key field value of KEYSTREAM0 through KEYSTREAM3 combined with a LOAD\_QUAD or STORE\_QUAD command is illegal and generates undefined results.

### 5.6.1.3 Command.i Field

Upon completion of each command, a hardwired interrupt may optionally be generated. Interrupt generation is controlled by the 2-bit Command.i field in conjunction with an internal interrupt condition bit generated by the channel. The channel generates the interrupt condition based on the current values of the ChannelStatus.s7..s0 bits along with mask and data values found in the InterruptSelect register of each channel. The suggested algorithm for generating the interrupt condition is as follows:

```
c = (ChannelStatus.s7..s0 & InterruptSelect.mask)
   == (InterruptSelect.value & InterruptSelect.mask)
```

In many implementations there is no need to allow all of the general purpose status bits to generate interrupts. In these cases, the generation of the interrupt condition may be as simple as tying it to a single status bit or tying it to 0. The foregoing algorithm should be used only in those cases where it is desired to allow interrupts to be generated based on the values of multiple status bits.

Table 81 indicates how the interrupt condition bit is used in conjunction with the Command.i field to determine whether or not to generate an interrupt.

Table 81. Effects of Command.i field

Value	Effect
0	Never interrupt
1	Interrupt if the interrupt condition bit is set
2	Interrupt if the interrupt condition bit is cleared
3	Always interrupt

**Note:** In implementations that have no hard-wired interrupt signal (such as a DBD-MA controller used with certain I/O expansion buses), the Command.i field value is ignored.

#### 5.6.1.4 Command.b Field

Upon completion of each data transfer command, the CommandPtr is updated to point to either the next command or the target of a branch. Branching is controlled by the Command.b field in conjunction with an internal branch condition bit that is generated by the channel. The channel generates the branch condition based on the current values of the ChannelStatus.s7..s0 bits along with a mask and a data value found in the BranchSelect register of each channel. The algorithm for generating the branch condition is as follows:

```
c = (ChannelStatus.s7..s0 & BranchSelect.mask)
   == (BranchSelect.value & BranchSelect.mask)
```

Table 82 indicates how the branch condition bit is used in conjunction with the Command.b field to determine whether or not to take the branch.

Table 82. Effects of Command.b field

Value	Effect
0	Never branch
1	Branch if the branch condition bit is set
2	Branch if the branch condition bit is cleared
3	Always branch

### 5.6.1.5 Command.w Field

Upon completion of each command, channel operation may optionally be suspended. This action is controlled by the Command.w field in conjunction with an internal wait condition bit that is generated by the channel interface. The channel interface generates the wait condition based on the current values of the ChannelStatus.s7..s0 bits along with mask and data values found in the WaitSelect register of each channel. The suggested algorithm for generating the wait condition is as follows:

```
c = (ChannelStatus.s7..s0 & WaitSelect.mask)
   == (WaitSelect.value & WaitSelect.mask)
```

Table 83 indicates how the wait condition bit is used in conjunction with the Command.w field to determine whether or not to suspend command execution.

Table 83. Effects of Command.w field

Value	Effect
0	Never wait
1	Branch if the wait condition bit is set
2	Branch if the wait condition bit is cleared
3	Always wait

If a wait action is invoked, it occurs before the normal command completion sequence (branch determination, status update, interrupt generation, or next command fetch).

### 5.6.1.6 Command.reqCount Field

The Command.reqCount field is used to specify the number of bytes to be transferred by data transfer commands.

### 5.6.1.7 Command.address Field

The Command.address field is used to specify the primary 32-bit address (or the least-significant portion of a 64-bit address) used in data transfers. With INPUT and OUTPUT commands, the primary address is always a system memory address and points to a data buffer. With LOAD\_QUAD and

STORE\_QUAD commands, the primary address is the address to or from which the immediate data is to be moved. In the latter case, it may point to system memory space, device memory space, or channel registers, as controlled by the Command.key field.

### 5.6.1.8 Command.cmdDep Field

The meaning and format of the 32-bit Command.cmdDep field is dependent on the Command.cmd values. With STORE\_QUAD and LOAD\_QUAD commands, it is the data value to be written or read. With input and output commands, it specifies a 32-bit conditional branch target (or the least-significant portion of a 64-bit branch target). See the descriptions of individual commands in the next sections.

### 5.6.1.9 Command.xferStatus Field

Upon completion of a command, the current content of the channel's Channel-Status register is written to the Command.xferStatus field. (The format of the ChannelStatus register is defined in Section 5.5.3.) When Command.xferStatus is written to memory, the bit corresponding to ChannelStatus.active is always set to 1. This serves as an indication that the corresponding command has been executed, and both the xferStatus and resCount fields have been updated. To make use of this feature, host software should clear the active bit in the Command.xferStatus field to 0 when the command descriptor is initialized.

The ChannelControl.flush bit provides a mechanism for software to force buffered data to be written to memory and for status to be updated before the completion of the command. Partial status reports are characterized by the flush bit being set to 1 in the xferStatus field.

Some devices may wish to provide intermediate status autonomously, identifying how many data bytes have been transferred, before completing the transfers specified by the command descriptor. This may be accomplished by providing the device with the ability to set the flush bit in the ChannelControl register. Intermediate status reports may be useful in cases such as terminal input, where the host needs to respond quickly to input data but would like to avoid interrupts on every byte.

The intermediate status capability is optional. When it is implemented, the conditions under which partial status is returned are device-dependent; for example, status could be returned periodically, when signaled through device-dependent state bits, when a minimum number of data bytes are available, or in response to a combination of these conditions.

### 5.6.1.10 Command.resCount Field

Upon completion of a command, the 16-bit residual byte count value is written to the Command.resCount field. This value is normally 0, but may be more when the device prematurely terminates the data transfer.

**Note:** The Command.resCount and Command.xferStatus fields must be updated in an indivisible operation.

## 5.6.2 INPUT and OUTPUT Commands

The INPUT\_MORE, INPUT\_LAST, OUTPUT\_MORE, and OUTPUT\_LAST commands transfer data between system memory and the device stream. The OUTPUT\_MORE and OUTPUT\_LAST commands transfer data from memory (which typically goes into the attached device). The INPUT\_MORE and INPUT\_LAST commands transfer data (which typically comes from the attached device) into memory. Data chaining is accomplished through the use of the MORE and LAST versions of these commands. The MORE commands indicate that the current buffer is not expected to complete a logical record (such as a network packet). The LAST commands indicate that the buffer is expected to complete a logical record. The reqCount field specifies the number of bytes to be transferred, the address field specifies a starting system memory address, and the branchAddress field specifies the address from which to fetch the next command if the branch test is successful. The Command.key field selects which of the device access ports is used. These fields are shown in Figure 59.

Figure 59. INPUT and OUTPUT command format

4	1	3	2	2	2	2	16
cmd	r	key	r	i	b	w	reqCount
address							
branchAddress							
xferStatus						resCount	

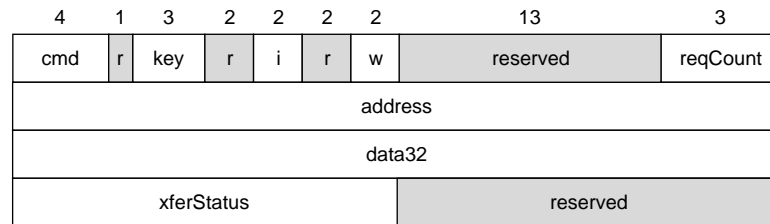
**Note:** An OUTPUT or INPUT command that performs a conventional transfer between system memory and an unaddressed device stream should specify a Command.key value of KEY\_STREAM0. Additional stream identifiers allow access to other streams that may be provided by the device, such as control or status information streams. Command.key values of KEY\_SYSTEM and KEY\_DEVICE are used with an (optional) Data2Ptr register to perform two-address transfers between system memory and the specified space.

In 64-bit implementations, the upper 32 bits of the buffer address are loaded with the value of the AddressHi register. The INPUT or OUTPUT command should always be preceded with a STORE\_QUAD command that sets AddressHi to the appropriate value.

### 5.6.3 STORE\_QUAD Command

The STORE\_QUAD command stores a 32-bit immediate value into memory. The 32-bit data32 field specifies the data value, and the address field (plus the key field) specifies the destination address, as shown in Figure 60.

Figure 60. STORE\_QUAD command format



The field normally used for specifying whether a branch is to be taken (Command.b) is reserved in this command. Software must write it with a 0 value. The operation of the channel is undefined when a nonzero value is written to this field.

The only valid values for the reqCount field are 1, 2, and 4. Only aligned transfers are supported. For example, if the count is 4, the low-order two bits of the address are assumed to be 0 and are ignored. Likewise, if the count is 2, the low-order bit of the address is assumed to be 0. When the count is less than 4 bytes, the least significant bytes of data32 are transferred.

STORE\_QUAD maps illegal reqCount values into legal ones as shown in Table 84. In Table 84, *x* represents a bit that is ignored.

Table 84. STORE\_QUAD reqCount values

reqCount	Effective Value
0b00 <i>x</i>	1
0b01 <i>x</i>	2
0b1 <i>xx</i>	4

During the execution of a STORE\_QUAD command, the Command.key value specifies the destination address for the immediate data. Only key values of KEY\_REGS, KEY\_SYSTEM, and KEY\_DEVICE are allowed. The operation of the channel is undefined when other key values are used.

### 5.6.4 LOAD\_QUAD Command

The LOAD\_QUAD command loads a 32-bit immediate value from memory. The 32-bit data32 field is the destination, and the address field (in combination with the key field) specifies the source address, as shown in Figure 61.

Figure 61. LOAD\_QUAD command format



The field normally used for specifying whether a branch is to be taken (Command.b) is reserved in this command. Software must write it with a 0 value. The operation of the channel is undefined when a nonzero value is written to this field.

LOAD\_QUAD handles the size and alignment of data transfers in the same way the STORE\_QUAD command does.

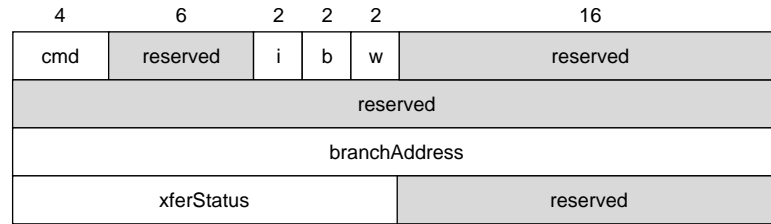
During the execution of a LOAD\_QUAD command, the Command.key value specifies the source address for the immediate data. Only key values of KEY\_REGS, KEY\_SYSTEM, and KEY\_DEVICE are allowed. The operation of the channel is undefined when other key values are used.

### 5.6.5 NOP Command

The NOP command performs no data transfers. However, it can use standard mechanisms to specify that interrupt, branch, or wait actions be performed. The NOP command is shown in Figure 62.



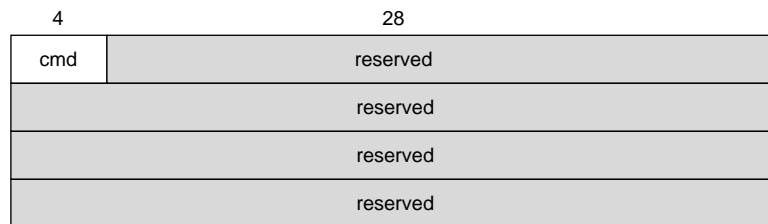
Figure 62. NOP command format



## 5.6.6 STOP Command

The STOP command deactivates channel processing. It is placed at the end of a command list to indicate that there are currently no further commands to be processed. The only effect of the STOP command is that the channel goes idle and clears the ChannelStatus.active bit. The format of the STOP command is shown in Figure 63.

Figure 63. STOP command format



To add additional commands to an active channel program, the following sequence should be performed:

1. Append the new commands to the existing channel program in memory.
2. Overwrite the STOP command with a NOP command (or the first new channel command).
3. Set the wake bit in the ChannelControl register to 1. This lets the channel process the new commands whether or not it had already completed the old commands and gone idle.

## 5.7 Asynchronous Event Packet Formats

The Macintosh DBDMA architecture defines a special channel to allow interfaces to report low frequency asynchronous events. This shared resource alleviates the need for an additional channel to be allocated to each interface. The reporting of modem signal changes (such as DCD or CTS) is one example of a situation in which this facility might be used.

**Note:** This feature is optional. In the presence of appropriate interrupt mechanisms and device status registers, it may not be needed.

The asynchronous event channel's command list is composed of a series of INPUT\_LAST commands with a count of 4 bytes. When an asynchronous event occurs, the DBDMA controller executes the current INPUT\_LAST command by writing a 4-byte event packet to the address indicated by Command.address. This packet consists of a 16-bit eventSource field and a 16-bit eventType field, as shown in Figure 64.

Figure 64. Event packet format



The eventSource field identifies the source of the asynchronous event. It should contain a value that identifies the channel affiliated with the event. The eventType field identifies the nature of the event that was generated by the source. The assignment of values within this field is implementation-specific.

## 5.8 Summary of DBDMA Options

The optional features of the DBDMA architecture, described in preceding sections, are summarized in this section.

**Requirement 5-5:** DBDMA implementations may omit the features listed in Section 5.8. If any of these features are included, they must conform to the appropriate sections of Chapter 5.

## 5.8.1 Hardwired Interrupts

For implementations that have no motherboard interrupt signal (for example, a DBDMA controller on an I/O extension bus), the Command.i bit value is ignored.

## 5.8.2 Asynchronous Event Channel

The event channel is optional and need not be implemented when the interrupt mechanism and device status registers are sufficient to report device-dependent asynchronous events.

## 5.8.3 Optional Registers

The CommandPtrHi, AddressHi, TransferMode, Data2PtrHi, Data2PtrLo, InterruptSelect, BranchSelect, and WaitSelect registers are optional. The CommandPtrHi, AddressHi, and BranchAddressHi registers are part of the 64-bit address extensions.

The TransferMode register is reserved for specifying special data transfer modes, to distinguish coherent from uncached access, or to restrict block transfer lengths.

The Data2PtrHi and Data2PtrLo registers specify secondary addresses for two-address transfers. Only a part of the Data2PtrLo register need be implemented if a small range of device register addresses are supported.

The InterruptSelect, BranchSelect, and WaitSelect registers are used when conditional generation of the interrupt, branch, and wait actions is desired. Only a subset of these registers need be implemented. It is strongly recommended that enough of the WaitSelect register be implemented to allow software to control the execution of the channel program.

## 5.9 Examples

This section presents examples of DBDMA architecture implementations.

### 5.9.1 Command Queuing

This section describes how the DBDMA architecture might be used to support a circular queue structure for commands—in this example, for Ethernet packet transmission.

Table 85 illustrates a sample channel program. In this example, three commands are required for each packet: one to transmit the header, one to transmit the data, and a final one to receive the transmit status information. An interrupt is generated when the last command of the sequence finishes.

Table 85. Queued transmission commands

No.	Command	Key	i	b	w	reqCount	Address	cmdDep
0	OUTPUT_MORE	STREAM0	Never	Never	Never	100	&Hdr1	n.a.
1	OUTPUT_LAST	STREAM0	Never	Never	Never	1000	&Buf1	n.a.
2	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status1	n.a.
3	OUTPUT_MORE	STREAM0	Never	Never	Never	100	&Hdr2	n.a.
4	OUTPUT_LAST	STREAM0	Never	Never	Never	1000	&Buf2	n.a.
5	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status2	n.a.
6	STOP->NOP	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
7	OUTPUT_MORE	STREAM0	Never	Never	Never	100	&Hdr2	n.a.
8	OUTPUT_LAST	STREAM0	Never	Never	Never	1000	&Buf2	n.a.
9	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status2	n.a.
10	STOP	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
11	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-
13	NOP (Branch)	n.a.	Never	Always	Never	n.a.	n.a.	&Cmd0

In the example, commands 0 through 5 represent a request to transmit two packets. Initially these commands are followed by a STOP command because there is no further work to be done. When a new request for a packet transmission is received from a client, it can be added to the queue without waiting for completion of the first two packets. This is accomplished by appending the new commands to the list, changing the STOP to a NOP, and setting the wake bit in the ChannelControl register to 1.

Adding commands to the queue is illustrated by commands 6 through 10 in the example. Note that there are several alternatives to the insertion of the NOP command. For example, the STOP command could be overwritten with the new OUTPUT\_MORE command after the remaining commands and the new

STOP command are written. Software could also replace the stop with a NOP that branches to a new command group, terminated with a STOP command.

Command groups may continue to be added in this manner until the allocated descriptor blocks are exhausted. At that point, a branch back to the first descriptor can be executed, and new command groups can overwrite those that have been executed and released by hardware. (Hardware releases descriptors by writing to the Command.xferStatus field.) The branch can either take the form of a static, standalone command, as illustrated here, or can be embedded in the last data transfer command.

## 5.9.2 Ethernet Reception

This section illustrates one possible way in which the DBDMA architecture can be used to support Ethernet packet reception and mini-buffer allocation. As shown in Table 86, the command list is structured as a number of channel command pairs.

Table 86. Ethernet receive channel commands

No.	Command	Key	i	b	w	reqCount	Address	cmdDep
0	STORE_QUAD	SYSTEM	Never	Never	Never	4	&BrSelReg	00400040 (EOF = 1)
1	INPUT_MORE	STREAM0	Never	!Cond	Never	256	&Buffer0	&Cmd3
2	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status0	n.a.
3	INPUT_MORE	STREAM0	Never	!Cond	Never	256	&Buffer1	&Cmd5
4	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status1	n.a.
5	INPUT_MORE	STREAM0	Never	!Cond	Never	256	&Buffer2	&Cmd7
6	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status2	n.a.
7	INPUT_MORE	STREAM0	Never	!Cond	Never	256	&Buffer3	&Cmd2
8	INPUT_LAST	STREAM1	Always	Never	Never	4	&Status3	n.a.
9	STOP	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
10	NOP (BRANCH)	n.a.	Never	Always	Never	n.a.	n.a.	&Cmd1

In this example, memory utilization is optimized by allocating buffers that are substantially smaller than the maximum packet size. Thus, in the typical sit-

uation where the majority of packets received are quite small, less physical memory must be allocated in order to buffer a given number of packets.

After each input buffer has been filled, the channel checks for packet completion. If the packet is complete, a command is executed that fetches the receive status from the device's status stream and stores it in system memory. An interrupt is generated when this command finishes. Note that the first command in the command list is a `STORE_QUAD`. This initializes the `BranchSelect` register to the value required for the following conditional branches by specifying that the end-of-frame bit be tested.

Only four input buffers are used in this example; in practice many more receive buffers would typically be allocated. The command list is structured as a circular queue of buffers. A `STOP` command must always be inserted after the last buffer/status command pair. This delineates the buffers that are owned by the controller from the returned buffers that are owned by the host, and prevents the controller from overwriting any filled buffers that have not been released by the host.

As buffers are filled and returned by the controller, new receive buffers are added to the queue in their place, and the position of the `STOP` command is advanced. This progression of the command list structure is shown in Table 87. Note that this example assumes that the first two received packets each fit within a single mini-buffer and that the third packet requires two mini-buffers.

Table 87. Ethernet receive command sequence

No.	Initial Command List	After Buffer 0 Is Filled	After Buffer 1 Is Filled	After Buffers 2 and 3 Are Filled
0	<code>STORE_QUAD</code>	<code>STORE_QUAD</code>	<code>STORE_QUAD</code>	<code>STORE_QUAD</code>
1 2	Buffer0 Status0	<code>STOP</code> <code>NOP</code>	Buffer5 Status5	Buffer5 Status5
3 4	Buffer 1 Status 1	Buffer 1 Status 1	<code>STOP</code> <code>NOP</code>	Buffer 6 Status 6
5 6	Buffer 2 Status 2	Buffer 2 Status 2	Buffer 2 Status 2	Buffer 7 Status 7
7 8	Buffer 3 Status 3	Buffer 3 Status 3	Buffer 3 Status 3	<code>STOP</code> <code>NOP</code>
9 10	<code>STOP</code> <code>NOP</code> (w. Branch)	Buffer 4 Status 4 (w. Branch)	Buffer 4 Status 4 (w. Branch)	Buffer 4 Status 4 (w. Branch)

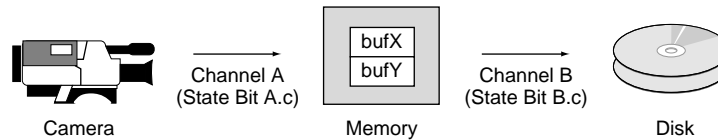
### 5.9.3 Full Handshake Flow Control

The DBDMA architecture also allows large device-to-device transfers to be performed autonomously, using small memory-resident buffers for intermediate data storage.

As an example, consider the live capture of the most-recent video images on a disk device, as illustrated in Figure 65. In the most general case, the transient data transfer rates of the two devices may be arbitrarily different—the disk would have a higher transient data transfer rate, but the data transfer rate would drop to 0 during seek operations.

For illustration purposes, assume that a small video sample is being saved in a circular buffer on disk.

Figure 65. Flow-controlled device model

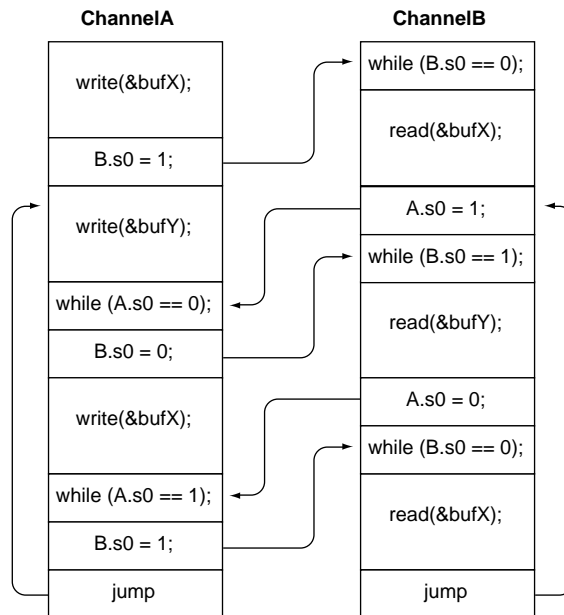


The DBDMA channels contain state bits that can influence the conditional execution of WAIT actions. Thus, channel programs can use this capability to spin-wait on externally generated events. For the data camera, flow control involves waiting for buffer-use bits to be set by the disk when a data buffer has been read from memory. For the disk, flow control involves waiting for buffer-use bits to be set by the data camera when a new data buffer has been written into memory.

The channelA program uses a STORE\_QUAD command to set a channelB state bit (B.s0) after each data buffer has been written; channelA waits for its own state bit (A.s0) to be set by channelB after each data buffer has been read.

The structure of these two channel programs is illustrated in Figure 66. In the diagram, dotted lines are used to illustrate how the STORE\_QUAD command entries in one program are coupled to the WAIT actions in the other.

Figure 66. Synchronizing flow-controlled channel programs







# Macintosh Serial Port

# 6

## 6.1 Overview

This chapter defines the serial port that Macintosh computers use to connect RS-422 serial data devices such as printers, modems, local-area networks, and telephone interface pods.

First- and second-generation Power Macintosh computers have two serial ports on the back panel; Power Macintosh computers that comply with the CHRP specification will have either one or two. For information about including Macintosh serial ports in the CHRP architecture, see *PowerPC Microprocessor Common Hardware Reference Platform: I/O Reference*. This document is listed in the bibliography.

**Note:** An SCC controller capable of supporting two Macintosh serial ports is included in the Mac I/O chip, as described in 2.11.3, “SCC Support,” on page 63.

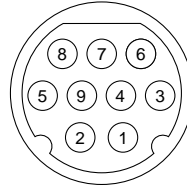
Macintosh serial ports support a range of communication products, including Apple GeoPort and LocalTalk devices, that make it easy for customers to connect their computers to data networks and telephone lines. For more information about Macintosh serial ports and compatible Macintosh peripheral devices, see *Guide to the Macintosh Family Hardware*, second edition. This book is listed in the bibliography.

**Requirement 6-1:** All Macintosh-compatible serial ports must comply with Sections 6.2 and 6.3 and must provide a signal interface compatible with an 85C30 serial communications chip.

## 6.2 Connector

The Macintosh-compatible serial port connector is a 9-pin mini-DIN socket, as shown in Figure 67. It accepts either 8-pin or 9-pin plugs.

Figure 67. Macintosh serial port connector



## 6.3 Pin Assignments

Table 88 gives the pin assignments for the Macintosh serial port.

Table 88. Serial port pin assignments

Pin	Name	Description
1	HSKo	Handshake output
2	HSKi	Handshake input or external clock (up to 920 Kbit/sec.)
3	TxD-	Transmit data -
4	GND	Ground
5	RxD-	Receive data -
6	TxD+	Transmit data +
7	GPI	General-purpose input (wake up CPU or perform DMA handshake)
8	RxD+	Receive data +
9	+5 V	Power to external device (350 mA maximum)

Pin 9 on the serial connector provides +5 V power from the power supply that supports the Apple Desktop Bus. An external device should draw no more than 100 mA from that pin. The total current available for all devices connected to the +5 V supply for the ADB and all serial ports must be at least 500 mA. Excessive current drain causes a circuit breaker to interrupt the +5 V

supply; the breaker automatically resets when the load returns to normal.

The serial port includes the GPi (general-purpose input) signal on pin 7. The GPi signal for the port connects to the corresponding data carrier detect input on the 85C30 hardware interface.

## 6.4 Serial Communications Controller

Macintosh serial ports are supported by an 85C30-compatible controller. In Macintosh computers, all ports can be independently programmed for asynchronous or synchronous communication formats including AppleTalk and the full range of Apple GeoPort protocols. With external adapters connected to the serial ports, the computer can communicate with a variety of ISDN and other telephone transmission facilities.

For a typical 85C30-compatible implementation, see Section 2.11.3, “SCC Support,” on page 63.

**Requirement 6-2:** A Macintosh serial port implementation must provide substantially the same functionality as described in Section 2.11.3.



# Macintosh Toolbox ROM

# 7

## 7.1 Overview

The Mac OS uses the Macintosh Toolbox ROM to obtain routines used by Macintosh applications and system software. The ROM is supplied by Apple and by third-party vendors.

This chapter specifies how the Macintosh ROM is connected to a computer system that complies with the CHRP specification.

**Requirement 7-1:** Implementations of the Macintosh ROM must comply with Sections 7.2 and 7.3.

## 7.2 Physical Mounting

The Macintosh ROM consists of four separate chips, each containing 512 K by 16 bits, for a total of 4 MB of data. Each chip contributes 16 data lines to the PowerPC processor bus.

Each chip typically requires a 0.01  $\mu$ F capacitor between +5 V and ground for noise suppression.

## 7.3 Electrical Connection

The four ROM chips are connected as shown in Table 89.

Table 89. ROM pin assignments

ROM Pin	All Chips	Chip 1	Chip 2	Chip 3	Chip 4
1	NC				
2	A18 (ROMAdr10)				
3	A17 (ROMAdr11)				
4	A7 (ROMAdr21)				
5	A6 (ROMAdr22)				
6	A5 (ROMAdr23)				
7	A4 (ROMAdr24)				
8	A3 (ROMAdr25)				
9	A2 (ROMAdr26)				
10	A1 (ROMAdr27)				
11	A0 (ROMAdr28)				
12	CE				
13	GND0				
14	OE				
15	D0	MemDat15	MemDat31	MemDat47	MemDat63
16	D8	MemDat7	MemDat23	MemDat39	MemDat55
17	D1	MemDat14	MemDat30	MemDat46	MemDat62
18	D9	MemDat6	MemDat22	MemDat38	MemDat54
19	D2	MemDat13	MemDat29	MemDat45	MemDat61
20	D10	MemDat5	MemDat21	MemDat37	MemDat53
21	D3	MemDat12	MemDat28	MemDat44	MemDat60
22	D11	MemDat4	MemDat20	MemDat36	MemDat52
23	V <sub>DD</sub> (+5 V)				
24	D4	MemDat11	MemDat27	MemDat43	MemDat59
25	D12	MemDat3	MemDat19	MemDat35	MemDat51

Table 89. ROM pin assignments (*continued*)

ROM Pin	All Chips	Chip 1	Chip 2	Chip 3	Chip 4
26	D5	MemDat10	MemDat26	MemDat42	MemDat58
27	D13	MemDat2	MemDat18	MemDat34	MemDat50
28	D6	MemDat9	MemDat25	MemDat41	MemDat57
29	D14	MemDat1	MemDat17	MemDat33	MemDat49
30	D7	MemDat30	MemDat24	MemDat40	MemDat56
31	D15	MemDat0	MemDat16	MemDat32	MemDat48
32	GND1				
33	BYTE (+5 V)				
34	A16 (ROMAdr12)				
35	A15 (ROMAdr13)				
36	A14 (ROMAdr14)				
37	A13 (ROMAdr15)				
38	A12 (ROMAdr16)				
39	A11 (ROMAdr17)				
40	A10 (ROMAdr18)				
41	A9 (ROMAdr19)				
42	A8 (ROMAdr20)				

## 7.4 Timing

The optimum timing for reading from 120/60 ns burst ROMs is shown in Table 90. The table specifies the ideal number of PowerPC processor bus clock cycles at bus rates of 25, 33, 40, and 50 MHz.



Table 90. ROM timing

Interval	Default	25 MHz	33 MHz	40 MHz	50 MHz
FirstWD: number of bus clock cycles between the assertion of TS and the first assertion of TA	8	3	4	5	6
BurstRate: number of bus clock cycles between successive TA assertions	4	2	3	3	4

# Open Firmware Requirements



## 8.1 Overview

The Mac OS is designed to boot in an Open Firmware environment compliant with IEEE 1275-1994 *Standard for Boot (Initialization, Configuration) Firmware*. This standard is listed in the bibliography.

This chapter describes the interactions between the Mac OS and Open Firmware and defines the specific information that the Mac OS must obtain from firmware that is built into the host computer or supplied with plug-in cards or other peripheral devices. For further details, see *Designing PCI Cards and Drivers for Power Macintosh Computers*, listed in the bibliography.

**Requirement 8-1:** Open Firmware compatible with the Mac OS must supply the device configuration tree properties listed in Table 91.

## 8.2 Name Registry

When the Mac OS boots on a computer that complies with the CHRP specification, it builds a data structure called the Name Registry from the device configuration tree (DCT) created by the computer's Open Firmware.

### 8.2.1 Description

The Name Registry provides Macintosh-compatible device drivers and system software with a way to store names. The information stored is determined by

the creator of the name entry and may include such data as the physical location of a code entity, technical descriptions of it, and logical addresses. The Macintosh Name Registry is similar to the name services used in some other computing environments. In concept it resembles the X.500 or BIND (named) network name services. However, the present implementation of the Mac OS Name Registry is less general; it is optimized for the specific needs of hardware and device driver configuration.

Name entries are created in the Name Registry by expert software that is a part of the Mac OS. Each expert owns specific entries and is responsible for removing them when they are no longer needed. Clients (such as device drivers) search for entries the expert has placed in the Name Registry, making the Name Registry a rendezvous point for clients and experts. The Name Registry does not provide general communication between clients and experts; it only helps clients and experts find each other. After clients and experts find each other, driver and other software helps them communicate directly.

## 8.2.2 Constructing the Name Registry

When the Mac OS is launched, it extracts device information from the DCT in the following steps:

1. Search for devices.
2. For each device, add a name entry and a set of properties to the Name Registry.
3. Find a driver for each device.
4. Initialize the driver.

Connections between name entries are formed when the entries are added to the Name Registry. The connections have direction and point from an existing entry to the new one. The software entity that creates a name entry owns it, whether it is an expert or a device driver. Only the owner should remove a name entry. Since most device drivers do not create entries in the Name Registry, most drivers never remove them.

The Mac OS places most of the name entries in the Name Registry during system startup. However, some hardware provides standard ways to probe for devices and return information describing them. In this case, the low-level expert responsible for that variety of hardware finds the devices and adds their names to the Name Registry. The low-level expert attaches descriptive information for each device to the name entry as properties. In a few cases, drivers may enter names and properties in the Name Registry directly.

### 8.2.3 Name Registry Properties

The Mac OS creates the properties shown in Table 91 from information in the DCT. Some of the properties listed are described in the next sections.

Table 91. Name Registry properties

Name	Description
<b>Standard properties specified by Open Firmware</b>	
address	Defines large virtual address regions
compatible	Defines alternate name property values
device-type	The implemented interface
fcode-rom-offset	Location of node's FCode in the expansion ROM
interrupts	Defines the interrupts used
model	Defines a manufacturer's model
name	Name of the name entry (nameString)
reg	The package's physical address space request
status	Indicates the device's operations status
<b>Standard properties specified by PCI binding to Open Firmware</b>	
alternate-reg	Alternate access paths for addressable regions
assigned-addresses	Assigned physical addresses
class-code	Value from corresponding PCI configuration register
device-id	Value from corresponding PCI configuration register
devsel-speed	Value from corresponding PCI configuration register
driver, <i>xxx,yyy,zzz</i>	Driver code for <i>xxx,yyy,zzz</i> platform
driver-reg, <i>xxx,yyy,zzz</i>	Not supported by Mac OS
fast-back-to-back	Value from corresponding PCI configuration register
max-latency	Value from corresponding PCI configuration register
min-grant	Value from corresponding PCI configuration register
power-consumption	Function's power requirements

Table 91. Name Registry properties (*continued*)

Name	Description
revision-id	Value from corresponding PCI configuration register
vendor-id	Value from corresponding PCI configuration register
<b>Properties specific to the Mac OS</b>	
AAPL,address	Vector to logical address pointers
AAPL,interrupts	Internal interrupt number
AAPL,slot-name	Physical slot identifier
depth	Color depth of each pixel (display device node only)
driver,AAPL,MacOS,PowerPC	Driver code for Mac OS
driver-description	Property that contains the driver description structure
driver-ist	IST member and set value, used to install interrupts
driver-ptr	Memory address of driver code
driver-ref	Reference to driver controlling a specific name entry
height	Height in pixels (for display device node only)
linebytes	Number of bytes in each line (display device node only)
width	Width in pixels (for display device node only)

### 8.2.3.1 Name Property

Manufacturers of computers and PCI expansion cards should use their United States stock symbol (if they are a publicly traded company) as the hardware manufacturer's ID in the name property. Otherwise, they can ask the IEEE to assign a 24-bit ID number by contacting

IEEE Registration Authority Committee  
 445 Hoes Lane  
 Piscataway, NJ 08855-1331  
 Telephone 809-562-3812

### 8.2.3.2 Driver Property

Mac OS run-time drivers written in PowerPC (native) code should use the following value for their driver property:

```
driver,AAPL,MacOS,PowerPC
```

### 8.2.3.3 Assigned Addresses

A Name Registry property that is important to drivers written in PowerPC code is the assigned-addresses property defined in *PCI Bus Binding to IEEE 1275-1994*. The assigned-addresses property tells the driver where a card's relocatable address locations have been placed in physical memory. Device driver code running with the Mac OS must resolve assigned-addresses values to AAPL,address values before using them.

### 8.2.3.4 ID Properties

Drivers can use the vendor-id, device-id, class-code, and revision-id properties to distinguish one device from another. However, these values typically refer to the controller instead of the device. For example, software will be unable to use these properties to distinguish between two video devices that use the same model of controller chip. Open Firmware can make the devices distinct by giving different names to them in the DCT.

### 8.2.3.5 ROM Offset

The fcode-rom-offset property contains the location in system ROM or in a PCI card's expansion ROM at which the FCode that produces the DCT node is found. Software can use the value of this property to determine the values of other properties, such as driver properties. If a card's expansion ROM contains no FCode, the fcode-rom-offset property will be absent from the card's Name Registry entry.

### 8.2.3.6 Driver-Ref Pointer

The driver-ref pointer can be important. This property is created by the Mac OS when a device driver is installed; it is the driver reference as defined by *Inside Macintosh: Devices*. The property is removed when the driver is removed. The presence of this property can be used to determine whether a particular device is open.

### 8.2.3.7 Driver Description

The driver-description property is a Name Registry data structure taken from the driver header; it defines various characteristics of the device. These characteristics include the driver name and version, run-time information that determines how a driver is handled when the Mac OS finds it, the driver's ability to support concurrent requests, and the driver's supported programming interfaces. For further information about the driver description data structure, see *Designing PCI Cards and Drivers for Power Macintosh Computers*, listed in the bibliography.

### 8.2.3.8 Apple Address Property

The PCI property assigned-addresses provides vector entries that represent the physical addresses of devices on expansion cards (see Section 8.2.3.3). Apple has added another property—AAPL,address—that provides a vector of 32-bit logical address values, where the nth value corresponds to the nth assigned-addresses vector entry. When accessing device functions located in memory space, software should use the corresponding AAPL,address property as the device's base. The same technique is recommended when software accesses high-performance device functions in PCI I/O space.

Using the AAPL,address property, a driver can find the logical address of an I/O resource. Accessing the logical address generates an I/O cycle on the PCI bus. Using the logical base address, a driver can generate a PCI I/O cycle in the same way it accesses a PCI device in memory space. This provides the fastest possible interface to PCI I/O space.

### 8.2.3.9 Apple Interrupts Property

The AAPL,interrupts property is an internal interrupt number that the Open Firmware startup process creates before any FCode is read from the card.

### 8.2.3.10 Apple Slot Name Property

The AAPL,slot-name property is an identifier for the hardware slot in which the card is plugged. This property is created by the Open Firmware startup process before any FCode is read from the card.

### 8.2.3.11 Graphic Display Properties

The height, width, linebytes, and depth properties are attached to the Name Registry entries of graphic display devices to provide a definition of each display's characteristics.

### 8.2.3.12 Driver Code Access

The property driver, *xxx,yyy,zzz* provides access to driver code. The value of *xxx* specifies the hardware manufacturer (AAPL for Apple Computer, Inc.), *yyy* specifies the operating system (MacOS), and *zzz* specifies the instruction set architecture (PowerPC). The value of driver, *xxx,yyy,zzz* is the driver code itself, which can be quite large; there is no defined upper limit to the size of a property's data.

## 8.2.4 Translation From the DCT to the Name Registry

The Name Registry generally takes the form of a global tree structure with a large branch equal to the original Open Firmware DCT plus and minus a few properties. When translating the DCT to the Mac OS Name Registry, the only pruning of the original tree is to delete drivers for other operating systems that may be stored there. All drivers with a driver,AAPL,MacOS,PowerPC property are brought into the Name Registry.

A typical DCT for a second-generation PCI-based Power Macintosh computer (the Power Macintosh 9500) is the following.

```
/bandit@F2000000
  /gc@10
    /53c94@10000
      /sd@0,0
    /mace@11000
    /esc@13020
    /esc@13000
    /awacs@14000
    /swim3@15000
    /via-cuda@16000
      /adb@0,0
        /keyboard@0,0
        /mouse@1,0
```



```
    /pram@0,0
    /rtc@0,0
    /power-mgt@0,0
/mesh@18000
    /sd@0,0
/bandit@B
    /AAPL,8250@E
/bandit@F4000000
    /bandit@B
    /ATY,mach64@E
/hammerhead@F8000000
```

For information about the Power Macintosh 9500 computer and the chip names shown in the foregoing listing, see *Macintosh Developer Note Number 12*, listed in the bibliography.

# Mac OS NVRAM Requirements

# 9

## 9.1 Overview

The Mac OS uses nonvolatile RAM (NVRAM) to store information when the hardware platform is not running. This chapter defines that usage.

**Requirement 9-1:** Platform hardware must provide at least 2 KB of NVRAM for exclusive use by the Mac OS.

## 9.2 NVRAM Allocation

Apple typically reserves 2048 bytes of NVRAM space for use by Macintosh firmware and system software, as shown in Table 92. Part of this allocation constitutes the 256 bytes of parameter RAM (PRAM) that Power Macintosh computers traditionally provide for use by the Mac OS.

Table 92. Typical NVRAM space allocations

Length (Bytes)	Description
768	Diagnostics
256	Parameter RAM
1024	Name Registry properties

The allocations shown in Table 92 provide permanent configuration data storage, both for the Mac OS and for PCI expansion cards. Expansion card firmware and application software can access some of the Macintosh PRAM space by using the Macintosh Toolbox routines described in *Inside Macintosh: Operating System Utilities*.

### 9.3 Storing Name Registry Data in NVRAM

Hardware NVRAM can be used to store device properties permanently. However, such storage is necessary only for devices used during the Mac OS startup process, because other devices can store an unlimited amount of permanent information on disk in the Macintosh System Preferences folder.

Firmware in the Macintosh ROM provides routines that can store the contents of a Name Registry property entry in NVRAM. During the Mac OS startup process, the firmware retrieves the entry value from NVRAM and places it in the Name Registry (see Section 8.2). The Macintosh device location algorithm encodes only five levels of PCI-to-PCI bridges. Devices located more than five levels from the PCI host bridge cannot store properties in NVRAM.

The Mac OS Name Registry properties stored in NVRAM are available to boot devices before the devices have been installed. For example, properties stored in NVRAM can be used to configure a primary display or define the net address of a network boot device. In both cases, the device driver can access user-changeable information before disk storage services are available.

To provide facilities for multiple boot devices, each entry in the Name Registry can store a single, small property in NVRAM. The Name Registry uses the following format to store the properties:

- **Device location** (6 bytes). An absolute location within the PCI system hardware universe. The format of this value is not public, and its value is not visible to drivers.
- **Property name** (4 bytes). A 1-byte to 4-byte string that is a creator ID assigned by Apple Developer Technical Support. Creator IDs are assigned on a first-come, first-served basis and form unique labels for products such as applications and driver files. Developers can use the C/F Registration Requests tool to register a creator ID with Apple Developer Technical Support.
- **Property value** (8 bytes maximum). A value that is stored and is retrieved by firmware in the Macintosh ROM.

# Macintosh Power Controls

# 10

## 10.1 Overview

The Mac OS and its compatible drivers expect to be able to exercise a certain amount of power control over the hardware on which they run. The extent of this control is defined in this chapter.

## 10.2 Determining Device Power Levels

The Mac OS supports a function called Driver Gestalt, by which it is able to query device drivers about the power requirements of the devices they control and about their ability to vary those requirements. The Driver Gestalt calls are listed in Table 93.

Table 93. Driver Gestalt selectors for power information

Selector	Description
'lpr'	True if driver supports power switching
'pmin3'	Minimum power consumption at 3.3 V
'pmin5'	Minimum power consumption at 5 V
'pmax3'	Maximum power consumption at 3.3 V
'pmax5'	Maximum power consumption at 5 V

For further information about using Driver Gestalt, see *Designing PCI Cards and Drivers for Power Macintosh Computers*, listed in the bibliography.

## 10.3 Setting Device Power Levels

The control and status calls that the Mac OS makes to drivers are shown in Table 94. They can ascertain the device's current power mode and set it high or low. Calls from the Mac OS to device drivers are described in *Designing PCI Cards and Drivers for Power Macintosh Computers*, listed in the bibliography.

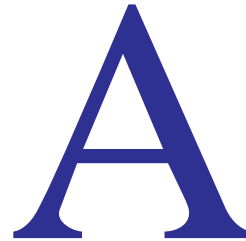
Table 94. Power setting controls

Selector	Description
kcsGetPowerMode	Status call to return current power mode
kcsSetPowerMode	Control call to set power mode

**Note:** The Mac I/O chip, described in Chapter 2, has a power-saving sleep mode, described in Section 2.7.4.

**Requirement 10-1:** Device drivers and other controllers for hardware that may consume more than 3 watts of power must support the high and low power mode controls listed in Tables 93 and 94.

# List of Requirements



## A.1 Apple Desktop Bus Controller

- 3-1:** The ADB implementation must conform to Apple specifications 062-0267-F and 062-2012-A.
- 3-2:** The ADB controller must support the signal lines listed in Table 47.
- 3-3:** The ADB controller must support the communication protocols described in Section 3.4.
- 3-4:** The ADB controller must recognize the error conditions listed in Table 48 and must handle them as described in Section 3.5.
- 3-5:** The ADB controller must perform the functions described in Section 3.6.
- 3-6:** The ADB controller must perform the special functions described in Sections 3.7.1 through 3.7.3.
- 3-7:** ADB bit cell timing must occur as shown in Figure 21 and must fall within the limits shown in Table 49.
- 3-8:** The ADB controller must determine bit cell values as specified in Section 3.8.1.
- 3-9:** The ADB controller must handle reset commands as described in Section 3.9.
- 3-10:** The ADB controller must implement the registers listed in Table 50.
- 3-11:** The ADB controller must implement the bit actions described in Sections 3.10.1.1 through 3.10.1.19.
- 3-12:** The ADB controller must implement timing registers as specified in Table 51.

## A.2 SCSI Support

- 4-1: The MESH SCSI controller must implement the signals specified in Table 52.
- 4-2: The electrical representation of the I/O types shown in the second column of Table 52 must conform to the limits shown in Table 53.
- 4-3: The MESH SCSI controller must communicate through 16 one-byte registers as specified in Table 54.
- 4-4: The registers listed in Table 54 must operate as describes in Sections 4.4.1 through 4.4.14.
- 4-5: The signal interface to the MESH SCSI controller must conform to Section 4.5.
- 4-6: The timing characteristics of the MESH SCSI controller must be as described in Section 4.6.
- 4-7: The external SCSI connection must be a 25-pin D type connector with pin assignments as shown in Table 71.

## A.3 Descriptor-Based DMA

- 5-1: DBDMA implementations must support both big-endian and little-endian byte ordering.
- 5-2: Every DBDMA reserved field must be zero when written and must be ignored when read. This must be true whether the reserved field is accessed by the DMA hardware or by device driver software.
- 5-3: The DBDMA controller's register interface must conform to the specifications of Section 5.5.
- 5-4: The format and operation of of DBDMA command entries must conform to Section 5.6.
- 5-5: DBDMA implementations may omit the features listed in Section 5.8. If any of these features are included, they must conform to the appropriate sections of Chapter 5.

## A.4 Macintosh Serial Port

- 6-1: All Macintosh-compatible serial ports must comply with Sections 6.2 and 6.3 and must provide a signal interface compatible with an 85C30 serial communications chip.
- 6-2: A Macintosh serial port implementation must provide substantially the same functionality as described in Section 2.11.3.

## A.5 Macintosh Toolbox ROM

- 7-1:** Implementations of the Macintosh ROM must comply with Sections 7.2 and 7.3.

## A.6 Open Firmware

- 8-1:** Open Firmware compatible with the Mac OS must supply the device configuration tree properties listed in Table 91.

## A.7 NVRAM

- 9-1:** Platform hardware must provide at least 2 KB of NVRAM for exclusive use by the Mac OS.

## A.8 Power Controls

- 10-1:** Device drivers and other controllers for hardware that may consume more than 3 watts of power must support the high and low power mode controls listed in Tables 93 and 94.





# Developer Services and Programs

# B

## B.1 Overview

Apple Computer's developer services and programs provide a broad mix of industry-leading technical information, products, training, and services. Several distinct programs are customized around the different types and levels of support that different developers need.

Apple also offers developer training classes, plus a variety of products and services through APDA, Apple Computer's sales channel for Apple and third-party development tools and technical resources. APDA is described in the bibliography.

## B.2 How to Get Information

Contact the Apple Developer Support Center for current information about training opportunities or to apply for membership in one of the Apple Developer Programs:

Apple Computer, Inc.  
Developer Support Center  
1 Infinite Loop, M/S 303-2T  
Cupertino, California 95014

Telephone: (408) 974-4897 (24 hour FAX on demand)

AppleLink: DEVSUPPORT

Internet: DEVSUPPORT@applelink.apple.com

Developers outside the U.S. and Canada should also contact their local Apple office or distributor for information about local programs.

## **B.3 Developer Resources**

Developers joining any of the Apple Developer Programs gain access to technical and marketing information, are informed about Apple developer events, and enjoy a communication channel to Apple, fellow developers, and customers. Apple delivers essential resources to all developers on CD-ROM, in printed form, and through on-line postings.

Apple developer services provide a variety of forms of assistance to developers of hardware and software compatible with Macintosh computers and computers that comply with the CHRP specification. Some of these services are described in the next sections.

### **B.3.1 Orientation Kit**

New members receive an orientation kit specific to the program or programs they have joined, to help them get started as Apple developers. The kit includes online communication information, development hardware order forms, a current developer price list, and information about Apple's Developer Program policies and procedures. Other materials include information about technical resources and marketing opportunities.

### **B.3.2 Developer University**

Training courses in Apple technologies and development tools are available through Apple Developer University. A broad curriculum of classes is offered, ranging from basic Macintosh programming skills through advanced courses in new technology. Developer University also publishes a number of self-paced training courses through APDA.

### **B.3.3 Worldwide Developers Conference**

The annual Worldwide Developers Conference is a technical meeting and showcase for developers in all areas of Apple technology. The conference provides a unique opportunity to meet fellow developers and hear details about Apple Computer's business and technology directions. There is a separate fee to attend the conference.

### **B.3.4 Hardware Purchasing Privileges**

Apple Computer's development hardware purchasing privileges give developer program members access to limited quantities of vintage, current, and forthcoming Apple products at a discount.

### **B.3.5 Technology Seeding**

Macintosh technology seeding gives developers an opportunity to work with new Macintosh products, including CHRP technology products, before the products are announced or introduced in the marketplace.

### **B.3.6 Market Research**

*Apple Directions*, an informative market research monthly, encourages technology adoption and increases developer effectiveness by communicating Apple's strategic, business, and technical directions.

### **B.3.7 Technical Journal**

Apple publishes *develop*, a quarterly technical journal that provides developers with an in-depth look at code and techniques that have been reviewed for robustness by Apple engineers. Each issue comes with a CD-ROM that contains the source code for that issue, all back issues, and much more.

### **B.3.8 Developer CD**

Developers can subscribe to the monthly Macintosh Developer CD, which includes online copies of the Macintosh System Software, *New Inside Macintosh*, *Macintosh Technical Notes*, *Human Interface Guidelines*, and sample code and snippets.

### **B.3.9 Software Development Kit**

Developers have access to the Mac OS Software Development Kit CD, containing the latest header files and other development aids for Macintosh software development.

### **B.3.10 Compatibility Testing Lab**

Apple Computer's Third-Party Compatibility Testing Lab, located in Cupertino, California, is available to members of Apple Developer Programs on an appointment basis. The lab provides program members access to a wide range of Apple equipment for compatibility testing. Participating developers are required to share test data with Apple on a confidential basis.

# Bibliography

## Apple / IBM / Motorola

*PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*, Morgan Kaufmann, San Francisco, CA, is the primary standard for designing computers to the CHRP architecture.

*PowerPC Microprocessor Common Hardware Reference Platform: I/O Reference* presents architectural specifications for CHRP input and output facilities. It is available from IBM.

## Apple Computer, Inc.

Apple Developer Press publishes a variety of books and technical notes designed to help third-party developers design hardware and software products compatible with Apple computers.

*Inside Macintosh* is a collection of books, organized by topic, that describe the system software of Macintosh computers. Together, these books provide the essential reference for programmers, software designers, and engineers. They include the following titles:

*Inside Macintosh: AOCE Application Interfaces*

*Inside Macintosh: AOCE Service Access Modules*

*Inside Macintosh: Devices*

*Inside Macintosh: Files*

*Inside Macintosh: Imaging With QuickDraw*

*Inside Macintosh: Interapplication Communication*

*Inside Macintosh: Macintosh Toolbox Essentials*

*Inside Macintosh: Memory*

*Inside Macintosh: More Macintosh Toolbox*

*Inside Macintosh: Networking*

*Inside Macintosh: Operating System Utilities*

*Inside Macintosh: Overview*

*Inside Macintosh: PowerPC Numerics*

*Inside Macintosh: PowerPC System Software*

*Inside Macintosh: Processes*

*Inside Macintosh: QuickDraw GX Environment and Utilities*

*Inside Macintosh: QuickDraw GX Graphics*

*Inside Macintosh: QuickDraw GX Objects*

*Inside Macintosh: QuickDraw GX Printing*

*Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*

*Inside Macintosh: QuickDraw GX Environment and Utilities*

*Inside Macintosh: QuickTime*

*Inside Macintosh: QuickTime Components*

*Inside Macintosh: Sound*

*Inside Macintosh: Text*

*Inside Macintosh: Devices* documents the last version of the Device Manager before its enhancements to support PowerPC native drivers.

*Inside Macintosh: PowerPC System Software* covers in detail the changes and extensions to Macintosh system software version 7.1 for Power Macintosh computers, including new Macintosh Toolbox managers and the run-time architecture that supports the PowerPC microprocessor.

*Building Programs for Macintosh With PowerPC* is a general discussion for developers of the development and building of application software for PowerPC microprocessor-based systems that run the Mac OS, including computers that comply with the CHRP specification.

*Technical Introduction to the Macintosh Family*, second edition, surveys the complete Macintosh family of computers from the developer's point of view.

*Macintosh Human Interface Guidelines* provides authoritative information on the theory behind the Macintosh "look and feel" and Apple Computer's standard ways of using individual interface components. A companion CD-ROM disk, *Making It Macintosh*, illustrates the Macintosh human interface guidelines through interactive, animated examples.

*Designing PCI Cards and Drivers for Power Macintosh Computers* is a complete hardware and software guide to the PCI bus implementation in first- and second-generation Power Macintosh computers. It also contains design guidelines and descriptions of Mac OS support for both boot and run-time PCI device drivers.

*Macintosh Developer Note Number 12* contains hardware details of the Power Macintosh 9500 computer, the first model of the PCI-based second generation of Power Macintosh computers.

Most of the Apple publications just listed are available from APDA. APDA is Apple Computer's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. APDA offers convenient payment and shipping options, including site licensing. To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

#### APDA

Apple Computer, Inc.

P.O. Box 319

Buffalo, NY 14207-0319

Telephone United States 1-800-282-2732

Canada 1-800-637-0029

International 716-871-6555

Fax 716-871-6511

eWorld APDA

AppleLink APDA

America Online APDAorder

CompuServe 76666,2405

Internet APDA@eworld.com

Apple has published two specifications for the Apple Desktop Bus:

- 062-0267-F Specification, Apple Desktop Bus
- 062-2012-A Engineering Specification, Macintosh Transceiver Interface, ADB

In addition to the specifications listed above, Macintosh Technical Note 144 (*Macintosh Color Monitor Connections*), Macintosh Technical Note 326 (*M.HW.SenseLines*), and *Macintosh New Technical Note HW-30* provide technical details of the interfaces to various Apple and third-party monitors. You can obtain these specifications and technical notes through the Apple Developer Support channels described in Appendix B.

## IBM

*The PowerPC Architecture: A Specification For a New Family of RISC Processors*, Second Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, ISBN 1-55860-316-6

*Open PIC Multiprocessor Interrupt Controller Register Interface Specification*, Revision 1.2



## American National Standards Institute

ANSI has prepared a standard called *ANSI/IEEE X3.215-199x Programming Languages—Forth*. It is a useful reference for the Forth language used in the Open Firmware process. ANSI also publishes the *ANSI X3T9.2 SCSI-3 SCSI Parallel Interface (SPI) Specification*, which defines the SCSI bus.

You can contact ANSI at

American National Standards Institute  
11 West 42nd Street  
New York, NY 10036  
Phone 212-642-4900  
Fax 212-302-1286

## FirmWorks

FirmWorks has issued a book, *Writing FCode Programs for PCI*, that provides essential information for programmers designing Open Firmware code. This book is published by FirmWorks and is available by writing to

FirmWorks  
480 San Antonio Road, Suite 230  
Mountain View, CA 94040-1218  
Email info@firmworks.com  
Phone 415-917-0100  
Fax 415-917-6990

## Institute of Electrical and Electronic Engineers

The essential IEEE document for designers of CHRP specification computers is *1275-1994 Standard for Boot (Initialization, Configuration) Firmware*, IEEE part number DS02683. It is referred to in this book as IEEE Standard 1275.

The IEEE also publishes documents that provide valuable background for understanding the Macintosh DBDMA architecture. They include:

*Standard 1596-1992, Scalable Coherent Interface.2*  
*Standard 1212-1991, Standard for Control and Status Register (CSR) Architecture for Microcomputer Buses*  
*Standard P1394, Serial Bus*

You can order IEEE documents from

IEEE Standards Department  
445 Hoes Lane, P.O. Box 1331  
Piscataway, NJ 08855-1331  
Phone 800-678-4333 (U.S.)  
908-562-5432 (International)

The P1275 Working Group continues to work on new PCI bus and processor bindings, as well as extensions to *IEEE Standard 1275*. Current documents, including *PCI Bus Binding to IEEE 1275-1994*, are available on an anonymous FTP site, donated by Sun Microsystems, at [playground.sun.com/pub/p1275](http://playground.sun.com/pub/p1275).

## PCI Special Interest Group

The essential PCI standard document for designers of CHRP specification computers is *PCI Local Bus Specification*, Revision 2.0. It is available from

PCI Special Interest Group  
P. O. Box 14070  
Portland, OR 97214  
Phone 800-433-5177 (U.S.)  
503-797-4207 (International)  
Fax 503-234-6762

The PCI SIG also publishes *PCI Multimedia Design Guide*.

## SunSoft Press

SunSoft Press has issued a book, *Writing FCode Programs*, that provides useful background information about the FCode used in the Open Firmware startup process. Its ISBN number is 0-13-107236-6. This book is published by PTR Prentice Hall and is available at most computer bookstores.



# Glossary

**address invariance** A feature of a data bridge (such as a PCI bridge) by which the address of any byte transferred across the bridge remains the same on both sides of the bridge.

**APDA** Apple Computer's worldwide direct distribution channel for Apple and third-party development tools and documentation products.

**aperture** A logical view of the data in a frame buffer, organized in a specific way and mapped to a separate area of memory. For example, a frame buffer may have a big-endian aperture and a little-endian aperture, providing instant access to the buffer in either addressing mode.

**Apple GeoPort interface** An RS-422 serial I/O interface through which Macintosh computers can communicate with a variety of ISDN and other telephone transmission facilities by using external pods.

**application programming interface (API)** A set of services in the Mac OS that supports application software. See system programming interface.

**big-endian** Used to describe data formatting in which each field is addressed by referring to its most significant byte. See also little-endian.

**boot driver** A device driver that is used during the Open Firmware startup process. It must be written in FCode and is usually loaded from system ROM or from the expansion ROM on a PCI card.

**bus transaction** A data exchange between two nodes, consisting of a request and a response. The request action transfers a read or write code from a requester to a responder; the response action returns a completion code from the

responder. Data is transferred in the request for write transactions and in the response for read transactions.

**byte lane** An 8-bit channel of a data bridge that passes individual data bytes.

**byte swapping** A technique of changing the order of byte lanes as they pass through a data bridge (such as a PCI bridge) that produces address invariance in a mixed-endian system.

**channel** In DBDMA, a DMA facility used to transport data bytes between system memory and attached device hardware.

**channel program** In DBDMA, a sequence of commands intended to perform one or more data transfers. Channel programs may incorporate conditional branches and waits, and may specify the generation of processor interrupts.

**coherency** See **memory coherency**.

**command descriptor** In DBDMA, a linked array containing instructions for transferring data by direct memory access.

**command list** In DBDMA, a sequence of command descriptors.

**concurrent drivers** Drivers that can process more than one request at a time.

**configuration** The process of modifying the software of a computer so it can communicate with various hardware components.

**DBDMA (descriptor-based direct memory access)** A means of transferring data rapidly into or out of RAM without passing it through the microprocessor. DBDMA uses array structures called command descriptors.

**device configuration tree (DCT)** A software structure, generated during the Open Firmware startup process, that assigns nodes to all PCI devices available to the system. The Mac OS extracts information from the device tree to construct the device parts of the Macintosh Name Registry.

**driver** The code that controls a physical device such as an I/O port or expansion card.

**driver gestalt call** A status call to a device driver that returns information such as the driver's revision level or the device's power consumption.

**dynamic random-access memory (DRAM)** Random-access memory in which each storage address must be periodically accessed ("refreshed") to maintain its value.

**expansion ROM** A ROM on a plug-in accessory card that supplies the computer with information about the card and any associated peripheral devices during the configuration process. Also called a *declaration ROM* or a *configuration ROM*.

**expert** Code that connects a family of devices to the I/O facilities of the Mac OS.

**family** A collection of devices that provide the same kind of functionality, such as a set of display devices.

**family expert** An expert that uses the Name Registry to find device entries of its family service type.

**FCode** A tokenized version of the Forth programming language, used in the Open Firmware startup process. The elements of FCode are all one or two bytes in length.

**frame buffer** Memory that stores one or more frames of video information until they are displayed on a screen.

**hardware interrupt** A physical device's method for requesting attention from a computer.

**IEEE** Institute of Electrical and Electronics Engineers.

**initiator** An entity that generates bus commands, allocates data buffers for data transfers, and sends the wakeup signal to the target to initiate command processing. The initiator typically receives status from the target upon command completion.

**little-endian** Used to describe data formatting in which each field is addressed by referring to its least significant byte. See also **big-endian**.

**low-level expert** An expert that places information about low-level code into the Name Registry.

**Mac I/O chip** An Apple-designed ASIC that implements most of the Macintosh technology specified in this book.

**Mac OS** Apple Computer's operating system software for Macintosh and Macintosh-compatible computers. Also called *Macintosh system software*.

**memory coherency** The property of a range or kind of memory by which all parts of the computing system access the same values. Memory coherency ensures that data being moved into or out of memory does not appear to have different values when accessed by the processor and by PCI host bridges.

**mixed-endian** The ability of a computer system, such as Power Macintosh, to support both big-endian and little-endian data formats.

**name entry** An element of the Name Registry. Name entries are connected hierarchically to other name entries and have properties.

**Name Registry** A high-level system service in the Mac OS that stores the names of software objects and the relations among the names. The Name Registry extracts device information from the device configuration tree and makes it available to Macintosh run-time drivers.

**native driver** A driver that is written in PowerPC code.

**nonvolatile RAM (NVRAM)** Memory, in either flash ROM or battery-powered RAM, that retains data between system startups.

**Open Firmware driver** A driver for an I/O device that is used during the Open Firmware startup process, before an operating system has taken control of the computer.

**Open Firmware startup process** The startup process, conforming to an IEEE standard, by which CHRP specification computers recognize and configure peripheral devices.

**PCI** Abbreviation for *Peripheral Component Interconnect*.

**PCI host bridge** A chip that communicates between the computer's microprocessor and a PCI local bus.

**PCI local bus** A bus architecture for connecting ASICs and plug-in expansion cards to a computer's main processor and memory. It is defined by the PCI specification.

**PCI specification** *PCI Local Bus Specification*, Revision 2.0, a document issued and maintained by the PCI Special Interest Group.

**phase expecting** A performance-enhancing technique used by a low-level SCSI host interface, in which it anticipates the phases a SCSI bus must go through to complete a transaction.

**physical device** A piece of computer hardware that performs an I/O function and is controlled by a driver.

**pixel** A single dot on a screen display.

**PowerPC** A family of RISC microprocessors. PowerPC 601, 603, and 604 microprocessors are currently used in Macintosh computers.

**property** A piece of descriptive information associated with a node in the device configuration tree or with a name entry in the Name Registry.

**read transaction** A bus transaction that returns data from a responder to a requester.

**reduced instruction set computing (RISC)** A technology of microprocessor design in which all machine instructions are uniformly formatted and are processed through the same steps.

**requester** The entity that initiates a bus transaction and transfers an address and command to the responder.

**responder** The entity that completes a bus transaction and returns status to the requester.

**run-time driver** A device driver that is used by an operating system after the Open Firmware startup process has finished. It may be supplied by the operating system or contained in the expansion ROM on a PCI card.

**scatter-gather** A technique for storing data in several discontinuous ranges of memory.

**SPI** See **system programming interface**.

**startup firmware** Code in the computer's system ROM that implements the Open Firmware startup process.

**system programming interface (SPI)** A set of services in the Mac OS that supports hardware-related code such as drivers. See **application programming interface**.

**target** The entity addressed by an initiator in a bus transaction.

**write transaction** A bus transaction that transfers data from a requester to a responder.





# Index

## A

- AAPL.address property (Open Firmware) 192, 194
- abbreviations xxvi
- active device address registers 102
- ActNeg bit 111
- ADB Reset (ADB\_RST) bit 103
- ADB. *See* Apple Desktop Bus
- address decoding on PCI bus 30
- address field (DBDMA commands) 167
- address invariance 142
- address phase of PCI transaction 28, 30
- address ranges xxvi
- AddressHi register (DBDMA) 161
- AIX operating system 4
- American National Standards Institute 212
- APDA 211
- Apple Desktop Bus (ADB) 11, 62, 81
  - connector 82
  - data transfers 99
  - documentation for 211
  - error handling 86
  - keyboard events 90
  - registers 94–104
  - resetting 103
  - timing controls 103
  - timing of 92
- Apple Developer Support 205
- Apple Directions* 207
- Apple GeoPort 4, 10, 181
- Apple technology licensing ii
- Arbitrate command 112
- arbitration on PCI bus 41
- ArbLost bit 116

- ARBus (Apple RISC bus) 2
- assigned-addresses property (Open Firmware) 193
- asynchronous events (in DBDMA) 173
- Atn bit 111
- Autopoll Data (APD) bit 86, 101
- Autopoll Enable (APE) bit 102
- autopolling (in ADB) 88

## B

- base register for the MIO chip 40
- big-endian addressing 3, 143
- BIST register 40
- bit cell timing 92
- branch field (DBDMA commands) 166
- BranchAddressHi register (DBDMA) 162
- BranchSelect register (DBDMA) 158
- BranchSelect register (MIO) 46
- burst rate for ROM accesses 188
- bus FIFO count register (MESH) 115
- bus FIFO register (MESH) 110
- bus status registers (MESH) 115
- BusFree command 114
- bytes lanes 32

## C

- Cache Line Size register 40
- cache memory 4, 25
- CachePD register 25
- ChannelControl register (DBDMA) 153
- ChannelControl register (MIO) 43
- ChannelStatus register (DBDMA) 154
- ChannelStatus register (MIO) 43
- Clock Fail Warning signal 27
- Cmd field (DBDMA commands) 163
- cmdDep field (DBDMA commands) 168
- CmdDone bit 119
- coherency of memory 3
- command list (in DBDMA) 147
- command queuing in DBDMA 174
- Command Response Expected (CRE) bit 102
- Command.b field 51
- Command.cmd field 48
- Command.cmdDep field 52
- Command.i field 50
- Command.key field 49
- Command.resCount field 53
- Command.w field 52

Command.xferStatus 52  
 Command/Data register file (CDRF) 83, 97  
 CommandPtr register (MIO) 45  
 CommandPtr registers (DBDMA) 157  
 Common Hardware Reference Platform  
   and Macintosh computers 3–5  
   specification 209  
 compatibility testing 208  
 configuration cycle on PCI bus 28, 36  
 connectors  
   ADB 82  
   SCSI 135  
   serial port 182  
 CPUID register 26  
 CpuInt signal 57

**D**

Data From Bus (DFB) bit 84, 98  
 Data Lost Error (DLE) 100  
 Data To Bus (DTB) bit 85, 99  
 Data2Ptr registers (DBDMA) 159  
 DBDMA 139  
   channel usage 146  
   channels (MIO) 15, 42  
   command descriptors 162–172  
   commands (MIO) 48  
   controller model 148  
   descriptor structure 139  
   endian variants 143  
   input and output commands 169  
   in MESH 130  
   in the MIO chip 10, 12, 42  
   optional registers 173  
   registers 151–162  
   reserved fields in 144  
 DestinationID register (MESH) 119  
 DetectAB register 65  
*develop* magazine 207  
 developer support 205  
 device configuration tree (DCT) 189, 195  
 device drivers 3  
 device ID for the MIO chip 26, 38  
 device power levels 199  
 device registers in the MIO chip 15  
 DFB Interrupt Source Enable (DFB\_IS\_EN) bit 98  
 diagnostic code 197  
 DisParChk command 114  
 DisReSel command 114  
 DMA bit (MESH) 111  
 DOS operating system 3  
 Driver Gestalt 199

driver,AAPL,MacOS,PowerPC property (Open  
   Firmware) 192, 193  
 driver-description property (Open Firmware) 194  
 driver-ref pointer (Open Firmware) 193  
 driver-ref property (Open Firmware) 192  
 driver-reg property (Open Firmware) 191

**E**

85C30 chip 10, 183  
 8259 interrupt controller 13, 57  
 endian addressing modes 3  
 EnParChk command 114  
 EnReSel command 114  
 Error register (MESH) 117  
 errors  
   in ADB 100  
   in DBDMA 150  
   in MESH 118  
 Ethernet 4, 176  
 Exception bit (MESH) 118  
 Exception register (MESH) 115  
 expert software in Mac OS 190

**F**

FCode 212, 213  
 fcode-rom-offset property (Open Firmware) 193  
 Feature Control register 26  
 FIFO Count register (MESH) 115  
 53C96 chip 69, 138  
 FirmWorks 212  
 Floppy disk support 4  
 Forth language 212

**G**

GeoPort serial port 4, 10  
 GPi (general purpose input) signal 183  
 graphic display properties (Open Firmware) 195

**H**

hard disk drives 4  
 hexadecimal notation xxvi  
 How Many Bytes (HMB) field 101  
 human interface guidelines 210

**I, J**

I/O cycles on PCI bus 28  
 ID properties (Open Firmware) 193  
 IDE bus 3, 4  
 IEEE Standard 1275 213  
 information transfer commands (MESH) 113

- Initial Recovery Count register 63
- initiator (in DBDMA) 147
- INPUT command (MIO) 53
- Inside Macintosh* 209
- Institute of Electrical and Electronic Engineers 212
- Intel processors 2
- interrupt field (DBDMA commands) 165
- Interrupt Mask register 118
- Interrupt register (MESH) 118
- Interrupt Request (IRQ) bit 100
- interrupts
  - in MESH 131
  - in the MIO chip 12, 57
- interrupts property (Open Firmware) 194
- InterruptSelect register (DBDMA) 157
- InterruptSelect register (MIO) 46
- ISA bus 5
- ISDN network interface 183
- K, L**
- key field (DBDMA commands) 163
- key field (MIO) 49
- Latency Timer register 40
- licensing iii
- little-endian addressing 3, 143
- LOAD\_QUAD command (DBDMA) 171
- LOAD\_QUAD command (MIO) 55
- LocalTalk 65, 181
- LTPC logic module 65–66
- M**
- Mac I/O chip 7
  - features 9
  - interrupts 57
  - PCI bus interface 28–42
  - registers 12–25
  - signals and pins 72
  - testing 71
- Mac OS operating system 2, 4
- Mac OS Software Development Kit 207
- Macintosh Technical Notes 207, 211
- Macintosh technology licensing ii
- Macintosh Toolbox ROM 185
- Media Access for Ethernet (MACE) 16
- memory allocations for the MIO chip 13
- memory cycles on PCI bus 28
- MESH SCSI controller 105
  - channel programs 70
  - electrical interface 108
  - and 53C96 chip 138
- ID register 121
  - in the MIO chip 10, 69
  - registers 109
  - SCSI bus interface 123
  - signals 106, 121
  - timing 126–135
- MIO. *See* Mac I/O chip
- modem port 183
- monitors 211
- Monostable Reset Enable (MR\_EN) bit 103
- N**
- name property (Open Firmware) 192
- Name Registry 189, 198
  - and device configuration tree 195
  - driver code in 195
  - and NVRAM 198
- NetWare operating system 4
- No Response Error (NRE) 101
- nonmaskable interrupt (NMI) key sequence 90
- NOP command (DBDMA) 171
- NOP command (MIO) 55
- NuBus 3, 5
- NVRAM 4, 197
- O**
- O'Hare I/O controller 7
- Open Firmware 3, 189
  - documentation for 212
- Open PIC 11
  - disabling 11, 27, 60
- OS/2 operating system 4
- OUTPUT command (MIO) 53
- P, Q**
- parity on PCI bus 32
- ParityErr bits 117
- patent licensing ii
- PCI bus
  - binding to Open Firmware 213
  - documentation for 210
  - levels of 198
  - in the MIO chip 9, 28–42
- PCI Special Interest Group 213
- phase expecting (MESH) 125
- Phase Mismatch bit 116
- power control 90, 199
- Power Macintosh computers 2
  - documentation for 210
  - networking in 181

PowerPC microprocessor 2

  programming for 210

PRAM (Macintosh NVRAM) 4, 197

processor bus coherency 3

properties (Open Firmware) 191

P1284 parallel port 4

## R

RAM storage 4

  read transactions on PCI bus 34

  reqCount field (DBDMA commands) 167

  requirements in this specification xxv, 201

  resCount field (DBDMA commands) 169

  Reselected bit (MESH) 116

  Reset key sequence 90

RISC architecture 2

ROM for Mac OS 185

ROM offset (Open Firmware) 193

RS-232 serial port 4

RS-422 serial port 181

RstMESH command 114

## S

SCC clock (MIO) 26

SCC. *See* Serial Communications Controller

SCSI bus 4

  arbitration 134

  connector for 135

  documentation for 212

  operating phases 124

SCSI clock (MIO) 26

SCSI controller. *See* MESH SCSI controller

SCSIRst bit (MESH) 117

  seeding 207

Selected bit (MESH) 116

Selection command 113

Selection Timeout register 121

SelTO bit 116

SelWatn bit 116

Seq bits 112

SeqErr bit (MESH) 117

Sequence register 111

Serial Communications Controller (SCC) 10, 63

  serial port 182

    modem power from 183

    pin assignments 182

  signal nomenclature xxvi

  68000 processor architecture 2

  sleep mode 27

  Solaris operating system 4

Sound Blaster 4

SourceID register (MESH) 119

  specification requirements xxv

SRQ autopolling (in ADB) 86–88

Start registers 65

  status information on PCI bus 39

STOP command (DBDMA) 172

STOP command (MIO) 56

STORE\_QUAD command (DBDMA) 170

STORE\_QUAD command (MIO) 54

SunSoft Press 213

synchronous data transfers (MESH) 120

SyncOffset field 119

SyncParms register 119

SyncPeriod field 120

## T, U

TAG Interrupt Source Enable (TAG\_IS\_EN) bit 98

target (of DBDMA) 145

technology licensing ii

technology seeding 207

termination of SCSI bus 136

timing of ROM accesses 187

TMode bit 111

Toolbox ROM 185

transaction termination on PCI bus 33

Transfer Access Grant (TAG) bit 98

Transfer Access Request (TAR) bit 84, 97

Transfer Count registers 109

TransferMode register (DBDMA) 159

turnaround cycle 31

typographical conventions xxvi

UnExpDisc bit (MESH) 117

## V

vendor ID for the MIO chip 38

Versatile Interface Adapter (VIA) 11, 61

video capture using DBDMA 178

## W

wait field (DBDMA commands) 167

WaitSelect register (DBDMA) 158

WaitSelect register (MIO) 47

Windows software 3, 4

Worldwide Developers Conference 206

write transactions on PCI bus 36

## X, Y, Z

x86 technology 2

xferStatus field (DBDMA commands) 168

THIS APPLE MANUAL was written, edited, and composed on a desktop publishing system using Apple Power Macintosh computers and FrameMaker software. Final page proofs were created on an Apple LaserWriter Pro printer.

