Parts Not Suitable for New Designs

For Additional Information

End-Of-Life Product Change Notice

MOTOROLA

MC68307

Integrated Multiple-Bus Processor User's Manual

68K FAX-IT

Documentation Comments

FAX 512-891-8593—Documentation Comments Only

The Motorola High-Performance Embedded Systems Technical Communications Department provides a fax number for you to submit any questions or comments about this document or how to order other documents. We welcome your suggestions for improving our documentation. Please do not fax technical questions.

Please provide the part number and revision number (located in upper right-hand corner of the cover) and the title of the document. When referring to items in the manual, please reference by the page number, paragraph number, figure number, table number, and line number if needed.

When sending a fax, please provide your name, company, fax number, and phone number including area code.

Applications and Technical Information

For questions or comments pertaining to technical information, questions, and applications, please contact one of the following sales offices nearest you.

— Sales Offices —

Field Applications Engineering Available Through All Sales Offices

UNITED STATES

ALABAMA, Huntsville	(205) 464-6800
ARIZONA, Tempe	(602) 897-5056
CALIFORNIA, Agoura Hills	(818) 706-1929
CALIFORNIA, Los Angeles	(310) 417-8848
CALIFORNIA, Irvine	(714) 753-7360
CALIFORNIA, Rosevllle	(916) 922-7152
CALIFORNIA, San Diego	(619) 541-2163
CALIFORNIA, Sunnyvale	(408) 749-0510
COLORADO, Colorado Springs	(719) 599-7497
COLORADO, Denver	(303) 337-3434
CONNECTICUT, Wallingford	(203) 949-4100
FLORIDA, Maitland	(407) 628-2636
FLORIDA, Pompano Beach/ Fort Lauderdale	(305) 486-9776
FLORIDA, Clearwater	(813) 538-7750
GEORGIA, Atlanta	(404) 729-7100
IDAHO, Boise	(208) 323-9413
ILLINOIS, Chicago/Hoffman Estates	(708) 490-9500
INDIANA, Fort Wayne	(219) 436-5818
INDIANA, Indianapolis	(317) 571-0400
INDIANA, Kokomo	(317) 457-6634
IOWA, Cedar Rapids	(319) 373-1328
KANSAS, Kansas City/Mission	(913) 451-8555
MARYLAND, Columbia	(410) 381-1570
MASSACHUSETTS, Marborough	(508) 481-8100
MASSACHUSETTS, Woburn	(617) 932-9700
MICHIGAN, Detroit	(313) 347-6800
MINNESOTA, Minnetonka	(612) 932-1500
MISSOURI, St. Louis	(314) 275-7380
NEW JERSEY, Fairfield	(201) 808-2400
NEW YORK, Fairport	(716) 425-4000
NEW YORK, Hauppauge	(516) 361-7000
NEW YORK, Poughkeepsie/Fishkill	(914) 473-8102
NORTH CAROLINA, Raleigh	(919) 870-4355
OHIO, Cleveland	(216) 349-3100
OHIO, Columbus/Worthington	(614) 431-8492
OHIO, Dayton	(513) 495-6800
OKLAHOMA, Tulsa	(800) 544-9496
OREGON, Portland	(503) 641-3681
PENNSYLVANIA, Colmar	(215) 997-1020
Philadelphia/Horsham	(215) 957-4100
TENNESSEE, Knoxville	(615) 690-5593
TEXAS, Austin	(512) 873-2000
TEXAS, Houston	(800) 343-2692
TEXAS, Plano	(214) 516-5100
VIRGINIA, Richmond	(804) 285-2100
WASHINGTON, Bellevue	(206) 454-4160
Seattle Access	(206) 622-9960
WISCONSIN, Milwaukee/Brookfield	(414) 792-0122

CANADA

BRITISH COLUMBIA, Vancouver	(604) 293-7605
ONTARIO, Toronto	(416) 497-8181
ONTARIO, Ottawa	(613) 226-3491
QUEBEC, Montreal	(514) 731-6881

INTERNATIONAL

AUSTRALIA, Melbourne	(61-3)887-0711
AUSTRALIA, Sydney	(61(2)906-3855
BRAZIL, Sao Paulo	55(11)815-4200
CHINA, Beijing	86 505-2180
FINLAND, Helsinki	358-0-35161191
Car Phone	358(49)211501
FRANCE, Paris/Vanves	33(1)40 955 900
GERMANY, Langenhagen/ Hanover	49(511)789911
GERMANY, Munich	49 89 92103-0
GERMANY, Nuremberg	49 911 64-3044
GERMANY, Sindelfingen	49 7031 69 910
GERMANY, Wiesbaden	49 611 761921
HONG KONG, Kwai Fong	852-4808333
Tai Po	852-6668333
INDIA, Bangalore	(91-812)627094
ISRAEL, Tel Aviv	972(3)753-8222
ITALY, Milan	39(2)82201
JAPAN, Aizu	81(241)272231
JAPAN, Atsugi	81(0462)23-0761
JAPAN, Kumagaya	81(0485)26-2600
JAPAN, Kyushu	81(092)771-4212
JAPAN, Mito	81(0292)26-2340
JAPAN, Nagoya	81(052)232-1621
JAPAN, Osaka	81(06)305-1801
JAPAN, Sendai	81(22)268-4333
JAPAN, Tachikawa	81(0425)23-6700
JAPAN, Tokyo	81(03)3440-3311
JAPAN, Yokohama	81(045)472-2751
KOREA, Pusan	82(51)4635-035
KOREA, Seoul	82(2)554-5188
MALAYSIA, Penang	60(4)374514
MEXICO, Mexico City	52(5)282-2864
MEXICO, Guadalajara	52(36)21-8977
Marketing	52(36)21-9023
Customer Service	52(36)669-9160
NETHERLANDS, Best	(31)49988 612 11
PUERTO RICO, San Juan	(809)793-2170
SINGAPORE	(65)2945438
SPAIN, Madrid	34(1)457-8204
or	34(1)457-8254
SWEDEN, Solna	46(8)734-8800
SWITZERLAND, Geneva	41(22)7991111
SWITZERLAND, Zurich	41(1)730 4074
TAIWAN, Taipei	886(2)717-7089
THAILAND, Bangkok	(66-2)254-4910
UNITED KINGDOM, Aylesbury	44(296)395-252

FULL LINE REPRESENTATIVES

COLORADO, Grand Junction	
Cheryl Lee Whltely	(303) 243-9658
KANSAS, Wichita	
Melinda Shores/Kelly Greiving	(316) 838 0190
NEVADA, Reno	
Galena Technology Group	(702) 746 0642
NEW MEXICO, Albuquerque	
S&S Technologies, Inc.	(505) 298-7177
UTAH, Salt Lake City	
Utah Component Sales, Inc.	(801) 561-5099
WASHINGTON, Spokane	
Doug Kenley	(509) 924-2322
ARGENTINA, Buenos Aires	
Argonics, S.A.	(541) 343-1787

HYBRID COMPONENTS RESELLERS

Elmo Semiconductor	(818) 768-7400
Minco Technology Labs Inc.	(512) 834-2022
Semi Dice Inc.	(310) 594-4631

PREFACE

The *MC68307 User's Manual* describes the programming, capabilities, and operation of the MC68307 and the *MC68000 Family Programmer's Reference Manual* provides instruction details for the MC68307.

The organization of this manual is as follows:

Section 1	Introduction
Section 2	Signal Description
Section 3	Bus Operation
Section 4	EC000 Core Processor
Section 5	System Integration Module
Section 6	Dual Timer Module
Section 7	M-Bus Interface Module
Section 8	Serial Module
Section 9	IEEE 1149.1 Test Access Port
Section 10	Applications Information
Section 11	Electrical Characteristics
Section 12	Ordering Information and Mechanical Data

TABLE OF CONTENTS

Section 1
Introduction

1.1 M68300 Family 1-3
1.1.1 Organization 1-3
1.1.2 Advantages 1-3
1.2 MC68307 Architecture 1-4
1.2.1 EC000 Core Processor 1-4
1.2.2 System Integration Module (SIM07) 1-4
1.2.2.1 External Bus Interface 1-5
1.2.2.2 Chip Select And Wait State Generation 1-5
1.2.2.3 System Configuration and Protection 1-5
1.2.2.4 Parallel Input/Output Ports 1-5
1.2.2.5 Interrupt Controller 1-6
1.2.3 Timer Module 1-6
1.2.4 UART Module 1-6
1.2.5 M-Bus Module 1-6
1.2.6 Test Access Port 1-7

Section 2
Signal Description

2.1 Bus Signals 2-5
2.1.1 Address Bus (A23–A0) 2-5
2.1.1.1 Address Bus (A23–A8) 2-5
2.1.1.2 Address Bus (AD7–AD0) 2-5
2.1.2 Data Bus (D15–D0) 2-6
2.2 Chip Selects 2-6
2.2.1 Chip Select 0 ($\overline{CS0}$) 2-6
2.2.2 Chip Select 1 ($\overline{CS1}$) 2-6
2.2.3 Chip Select 2 ($\overline{CS2}$, $\overline{CS2B}$, $\overline{CS2C}$, $\overline{CS2D}$) 2-6
2.2.4 Chip Select 3 ($\overline{CS3}$) 2-7
2.3 Bus Control Signals 2-7
2.3.1 Data Transfer Acknowledge ($\overline{DTACK}$) 2-7
2.3.2 Address Strobe ($\overline{AS}$) 2-8
2.3.3 Read/Write (R/$\overline{W}$) 2-8
2.3.4 Data Strobes, Upper and Lower ($\overline{UDS}$, $\overline{LDS}$) 2-8
2.3.5 8051 Address Latch Enable (ALE) 2-9
2.3.6 8051-Compatible Bus Read ($\overline{RD}$) 2-9
2.3.7 8051-Compatible Bus Write ($\overline{WR}$) 2-9
2.3.8 Bus Width Select for $\overline{CS0}$ (BUSW0) 2-9
2.4 Exception Control Signals 2-9

2.4.1 Reset (RESET) ... 2-9
2.4.2 Power-On Reset (RSTIN) ... 2-10
2.4.3 Halt (HALT) ... 2-10
2.4.4 Bus Request (BR/PA5) ... 2-10
2.4.5 Bus Grant (BG/PA6) ... 2-10
2.4.6 Bus Grant Acknowledge (BGACK /PA7) ... 2-10
2.5 Clock Signals ... 2-10
2.5.1 Crystal Oscillator (EXTAL, XTAL) ... 2-10
2.5.2 Clock Output (CLKOUT) ... 2-11
2.6 Test Signals ... 2-11
2.6.1 Test Clock (TCK) ... 2-11
2.6.2 Test Mode Select (TMS) ... 2-11
2.6.3 Test Data In (TDI) ... 2-11
2.6.4 Test Data Out (TDO) ... 2-11
2.7 M-Bus I/O Signals ... 2-11
2.7.1 Serial Clock (SCL/PB0) ... 2-11
2.7.2 Serial Data (SDA/PB1) ... 2-12
2.8 UART I/O Signals ... 2-12
2.8.1 Transmit Data (TxD/PB2) ... 2-12
2.8.2 Receive Data (RxD/PB3) ... 2-12
2.8.3 Request-To-Send (RTS/PB4) ... 2-12
2.8.4 Clear-To-Send (CTS/PB5) ... 2-12
2.9 Timer I/O Signals ... 2-12
2.9.1 Timer 1 Input (TIN1/PB6) ... 2-12
2.9.2 Timer 2 Input (TIN2/PB7) ... 2-13
2.9.3 Timer 1 Output (TOUT1/PA3) ... 2-13
2.9.4 Timer 2 Output (TOUT2/PA4) ... 2-13
2.10 Interrupt Request Inputs ... 2-13
2.10.1 Interrupt Inputs (INT1–INT8/PB8–PB15) ... 2-13
2.10.2 Non-Maskable Interrupt Input (IRQ7) ... 2-13
2.11 Use of Pullup Resistors ... 2-13
2.12 Signal Index ... 2-14

Section 3
Bus Operation

3.1 Data Transfer Operations ... 3-1
3.1.1 16-Bit M68000 Bus Operation ... 3-1
3.1.2 16-Bit M68000 Bus Read Cycle ... 3-2
3.1.3 16-Bit M68000 Bus Write Cycle ... 3-5
3.1.4 Read-Modify-Write Cycle ... 3-8
3.1.5 CPU Space Cycle ... 3-11
3.1.6 8-Bit M68000 Dynamically-Sized Bus ... 3-11
3.1.7 8051-Bus Operation ... 3-13
3.2 Bus Arbitration ... 3-15
3.2.1 Requesting the Bus ... 3-17

3.2.2 Receiving the Bus Grant 3-18
3.2.3 Acknowledgment of Mastership (Three-Wire Bus Arbitration Only) 3-18
3.3 Bus Arbitration Control 3-19
3.4 Bus Error And Halt Operation 3-27
3.4.1 Bus Error Operation 3-27
3.4.2 Retrying the Bus Cycle 3-29
3.4.3 Halt Operation 3-30
3.4.4 Double Bus Fault 3-31
3.5 Reset Operation 3-31
3.6 Asynchronous Operation 3-32
3.7 Synchronous Operation 3-35

Section 4
EC000 Core Processor

4.1 Features 4-1
4.2 Processing States 4-1
4.3 Programming Model 4-2
4.3.1 Data Format Summary 4-3
4.3.2 Addressing Capabilities Summary 4-3
4.3.3 Notation Conventions 4-4
4.4 EC000 Core Instruction Set Overview 4-6
4.5 Exception Processing 4-9
4.5.1 Exception Vectors 4-12
4.6 Processing of Specific Exceptions 4-12
4.6.1 Reset Exception 4-14
4.6.2 Interrupt Exceptions 4-14
4.6.3 Uninitialized Interrupt Exception 4-15
4.6.4 Spurious Interrupt Exception 4-15
4.6.5 Instruction Traps 4-16
4.6.6 Illegal and Unimplemented Instructions 4-16
4.6.7 Privilege Violations 4-17
4.6.8 Tracing 4-17
4.6.9 Bus Error 4-18
4.6.10 Address Error 4-18
4.6.11 Multiple Exceptions 4-19

Section 5
System Integration Module

5.1 Module Operation 5-2
5.1.1 MC68307 System Configuration 5-2
5.1.1.1 Module Base Address Register Operation 5-2
5.1.1.2 System Control Register Functions 5-4
5.1.1.3 System Protection Functions 5-5
5.1.2 Chip Select and Wait-State Logic 5-5
5.1.2.1 Programmable Data-Bus Size 5-6

5.1.2.2 Peripheral Chip Selects 5-7
5.1.2.3 8051-Compatible Bus Chip Select 5-8
5.1.2.4 Global Chip Select Operation (Reset Defaults) 5-8
5.1.2.5 Overlap in Chip Select Ranges 5-8
5.1.3 External Bus Interface Logic 5-9
5.1.3.1 M68000 Bus Interface 5-9
5.1.3.2 8051-Compatible Bus Interface 5-10
5.1.3.3 Port A, Port B General-Purpose I/O Ports 5-10
5.1.4 Interrupt Processing 5-13
5.1.4.1 Interrupt Controller Logic 5-14
5.1.4.2 Interrupt Vector Generation 5-15
5.1.4.3 $\overline{\text{IRQ7}}$ Non-Maskable Interrupt 5-17
5.1.4.4 General-Purpose Interrupt Inputs 5-17
5.1.4.5 Peripheral Interrupt Handling 5-18
5.1.5 Low-Power Sleep Logic 5-19
5.2 Programming Model 5-20
5.2.1 System Configuration and Protection Registers 5-22
5.2.1.1 Module Base Address Register (MBAR) 5-22
5.2.1.2 System Control Register (SCR) 5-23
5.2.1.3 System Status Register Bits Description 5-23
5.2.1.4 System Control Register Bits Description. 5-25
5.2.2 Chip Select Registers 5-30
5.2.2.1 Base Registers (BR3–BR0) 5-30
5.2.2.2 Option Registers (OR3–OR0) 5-32
5.2.3 External Bus Interface Control Registers 5-34
5.2.3.1 Port A Control Register (PACNT) 5-34
5.2.3.2 Port A Data Direction Register (PADDR) 5-35
5.2.3.3 Port A Data Register (PADAT) 5-35
5.2.3.4 Port B Control Register (PBCNT) 5-36
5.2.3.5 Port B Data Direction Register (PBDDR) 5-36
5.2.3.6 Port B Data Register (PBDAT) 5-37
5.2.4 Interrupt Control Registers 5-38
5.2.4.1 Latched Interrupt Control Registers 1,2 (LICR1,LICR2) 5-38
5.2.4.2 Peripheral Interrupt Control Register (PICR) 5-39
5.2.4.3 Programmable Interrupt Vector Register (PIVR) 5-40
5.3 MC68307 Initialization Procedure 5-41
5.3.1 Startup—Cold Reset 5-41
5.3.2 SIM Configuration 5-41

Section 6
Dual Timer Module

6.1 Overview 6-1
6.2 Module Operation 6-1
6.2.1 General-Purpose Timer Units 6-1
6.2.2 Software Watchdog Timer 6-3

6.3 Programming Model 6-4
6.3.1 General Purpose Timer Units 6-4
6.3.1.1 Timer Mode Register (TMR1, TMR2) 6-4
6.3.1.2 Timer Reference Registers (TRR1, TRR2) 6-5
6.3.1.3 Timer Capture Registers (TCR1, TCR2) 6-5
6.3.1.4 Timer Counter (TCN1, TCN2) 6-5
6.3.1.5 Timer Event Registers (TER1, TER2) 6-6
6.3.2 Software Watchdog Timer 6-7
6.3.2.1 Watchdog Reference Register (WRR) 6-7
6.3.2.2 Watchdog Counter Register (WCR) 6-7
6.4 Timer Programming Examples 6-8
6.4.1 Initialization and Reference Compare Function 6-8
6.4.2 Event Counting Function and Interrupts 6-9
6.4.3 Input Capture Function 6-9
6.4.4 Watchdog Usage Example 6-10

Section 7
M-Bus Interface Module

7.1 M-Bus System Configuration 7-2
7.2 M-Bus Protocol 7-2
7.2.1 START Signal 7-3
7.2.2 Slave Address Transmission 7-3
7.2.3 Data Transfer 7-3
7.2.4 Repeated START Signal 7-4
7.2.5 STOP Signal 7-4
7.2.6 Arbitration Procedure 7-4
7.2.7 Clock Synchronization 7-4
7.2.8 Handshaking 7-5
7.2.9 Clock Stretching 7-5
7.3 Programming Model 7-5
7.3.1 M-Bus Address Register (MADR) 7-6
7.3.2 M-Bus Frequency Divider Register (MFDR) 7-6
7.3.3 M-Bus Control Register (MBCR) 7-7
7.3.4 M-Bus Status Register (MBSR) 7-9
7.3.5 M-Bus Data I/O Register (MBDR) 7-10
7.4 M-Bus Programming Examples 7-10
7.4.1 Initialization Sequence 7-10
7.4.2 Generation of START 7-11
7.4.3 Post-Transfer Software Response 7-11
7.4.4 Generation of STOP 7-12
7.4.5 Generation of Repeated START 7-13
7.4.6 Slave Mode 7-13
7.4.7 Arbitration Lost 7-13

Section 8
Serial Module

8.1 Module Overview 8-2
8.1.1 Serial Communication Channel 8-2
8.1.2 Baud Rate Generator Logic 8-3
8.1.3 Baud Rate Generator/Timer 8-3
8.1.4 Interrupt Control Logic 8-3
8.1.5 Comparison of Serial Module to MC68681 8-3
8.2 Serial Module Signal Definitions 8-3
8.2.1 Transmitter Serial Data Output (TxD) 8-4
8.2.2 Receiver Serial Data Input (RxD) 8-4
8.2.3 Request-To-Send ($\overline{\text{RTS}}$) 8-4
8.2.4 Clear-To-Send ($\overline{\text{CTS}}$) 8-5
8.3 Operation 8-5
8.3.1 Baud Rate Generator/Timer 8-5
8.3.2 Transmitter and Receiver Operating Modes 8-5
8.3.2.1 Transmitter 8-6
8.3.2.2 Receiver 8-8
8.3.2.3 FIFO Stack 8-8
8.3.3 Looping Modes 8-10
8.3.3.1 Automatic Echo Mode 8-10
8.3.3.2 Local Loopback Mode 8-11
8.3.3.3 Remote Loopback Mode 8-11
8.3.4 Multidrop Mode 8-12
8.3.5 Bus Operation 8-14
8.3.5.1 Read Cycles 8-14
8.3.5.2 Write Cycles 8-14
8.3.5.3 Interrupt Acknowledge Cycles 8-14
8.4 Register Description and Programming 8-14
8.4.1 Register Description 8-14
8.4.1.1 Mode Register 1 (UMR1) 8-15
8.4.1.2 Mode Register 2 (UMR2) 8-17
8.4.1.3 Status Register (USR) 8-19
8.4.1.4 Clock-select Register (UCSR) 8-21
8.4.1.5 Command Register (UCR) 8-23
8.4.1.6 Receiver Buffer (URB) 8-25
8.4.1.7 Transmitter Buffer (UTB) 8-25
8.4.1.8 Input Port Change Register (UIPCR) 8-26
8.4.1.9 Auxiliary Control Register (UACR) 8-26
8.4.1.10 Interrupt Status Register (UISR) 8-27
8.4.1.11 Interrupt Mask Register (UIMR) 8-28
8.4.1.12 Timer Upper Preload Register (UBG1) 8-29
8.4.1.13 Timer Upper Preload Register (UBG2) 8-29
8.4.1.14 Interrupt Vector Register (UIVR) 8-29
8.4.1.15 Input Port Register (UIP) 8-29

8.4.1.16 Output Port Data Registers (UOP1, UOP0) 8-30
8.4.2 Programming 8-30
8.4.2.1 Serial Module Initialization 8-31
8.4.2.2 I/O Driver Example 8-31
8.4.2.3 Interrupt Handling 8-31
8.5 Serial Module Initialization Sequence 8-31

Section 9
IEEE 1149.1 Test Access Port

9.1 Overview 9-1
9.2 Tap Controller 9-3
9.3 Boundary Scan Register 9-4
9.4 Instruction Register 9-9
9.4.1 EXTEST (0000) 9-10
9.4.2 SAMPLE/PRELOAD (0010) 9-10
9.4.3 BYPASS (1111) 9-10
9.4.4 CLAMP (1100) 9-11
9.5 MC68307 Restrictions 9-11
9.6 Non-IEEE 1149.1 Operation 9-11

Section 10
Applications Information

10.1 MC68307 Minimum Stand-Alone System Hardware 10-1
10.1.1 MC68307 Signal Configuration 10-1
10.1.2 EPROM Memory Interface 10-5
10.1.3 RAM Memory Interface 10-5
10.1.4 RS232 UART Port 10-5
10.1.5 EPROM Timing 10-6
10.1.6 RAM Timing 10-6
10.2 Power Management 10-7
10.2.1 Fully Static Operation 10-7
10.2.2 Prescalable CPU Clock 10-8
10.2.3 Wake-Up 10-8
10.2.4 Low-Power Sleep Mode 10-9
10.2.5 Low-Power Stop Mode 10-9
10.3 Using M-Bus Software to Communicate Between Processor Systems 10-10
10.3.1 Overview of M-Bus Software Transfer Mechanism 10-11
10.3.2 M-Bus Master Mode Operation 10-12
10.3.3 M-Bus Slave Mode Operation 10-12
10.3.4 Description of Setup 10-13
10.3.5 Software Flow 10-13
10.3.6 Transfer Blocks 10-14
10.3.7 Software Implementation 10-14
10.3.7.1 Software Listing 1—M-Bus Master Software 10-17
10.3.7.2 Software Listing 2—M-Bus Slave Software 10-21

10.4 MC68307 UART Driver Examples 10-24
10.4.1 Software Listing 3 10-24
10.5 Swapping ROM and RAM Mapping on the MC68307 10-27
10.5.1 Software Implementation 10-27
10.5.1.1 Software Listing 4 10-28

Section 11
Electrical Characteristics

11.1 Maximum Ratings 11-1
11.2 Thermal Characteristics 11-1
11.3 Power Considerations 11-2
11.4 AC Electrical Specification Definitions 11-2
11.5 DC Electrical Specifications 11-4
11.6 AC Electrical Specifications—Clock Timing 11-4
11.7 AC Electrical Specifications—Read and Write Cycles 11-5
11.8 AC Electrical Specifications—Bus Arbitration 11-9
11.9 AC Electrical Specifications—8051 Bus Interface Module 11-11
11.10 Timer Module Electrical Characteristics 11-13
11.11 UART Electrical Characteristics 11-14
11.12 AC Electrical Characteristics—M-Bus Input Signal Timing 11-15
11.13 AC Electrical Characteristics—M-Bus Output Signal Timing 11-15
11.14 AC Electrical Characteristics—Port Timing 11-16
11.15 IEEE 1149.1 Electrical Characteristics 11-17

Section 12
Ordering Information and Mechanical Data

12.1 Standard Ordering Information 12-1
12.2 100-Pin PQFP Pin Assignments 12-1
12.3 100-Pin PQFP Package Dimensions 12-2
12.4 100-Pin TQFP Pin Assignments 12-3
12.5 100-Pin TQFP Package Dimensions 12-4

Index

LIST OF ILLUSTRATIONS

1-1 MC68307 Block Diagram 1-1
2-1 MC68307 Detailed Block Diagram 2-2
3-1 Word Read Cycle Flowchart (16-Bit Bus) 3-2
3-2 Byte Read Cycle Flowchart (16-Bit Bus) 3-3
3-3 Read and Write Cycle Timing Diagram (16-Bit Bus) 3-3
3-4 Word and Byte Read Cycle Timing Diagram (16-Bit Bus) 3-4
3-5 Word Write Cycle Flowchart (16-Bit Bus) 3-5
3-6 Byte Write Cycle Flowchart (16-Bit Bus) 3-6
3-7 Word and Byte Write Cycle Timing Diagram 3-6
3-8 Read-Modify-Write Cycle Flowchart 3-8
3-9 Read-Modify-Write Cycle Timing Diagram 3-9
3-10 Interrupt Acknowledge Cycle – Address Bus 3-11
3-11 Interrupt Acknowledge Cycle Timing Diagram 3-12
3-12 8051-Compatible Read Cycle Signals 3-14
3-13 8051-Compatible Write Cycle Signals 3-14
3-14 Three-Wire Bus Arbitration Cycle Flowchart 3-15
3-15 Two-Wire Bus Arbitration Cycle Flowchart 3-16
3-16 Three-Wire Bus Arbitration Timing Diagram 3-17
3-17 Two-Wire Bus Arbitration Timing Diagram 3-17
3-18 External Asynchronous Signal Synchronization 3-19
3-19 Bus Arbitration Unit State Diagrams 3-20
3-20 Three-Wire Bus Arbitration Timing Diagram—Processor Active 3-21
3-21 Three-Wire Bus Arbitration Timing Diagram—Bus Inactive 3-22
3-22 Three-Wire Bus Arbitration Timing Diagram—Special Case 3-23
3-23 Two-Wire Bus Arbitration Timing Diagram—Processor Active 3-24
3-24 Two-Wire Bus Arbitration Timing Diagram—Bus Inactive 3-25
3-25 Two-Wire Bus Arbitration Timing Diagram—Special Case 3-26
3-26 Bus Error Timing Diagram 3-28
3-27 Retry Bus Cycle Timing Diagram 3-29
3-28 Halt Operation Timing Diagram 3-30
3-29 Reset Operation Timing Diagram 3-32
3-30 Fully Asynchronous Read Cycle 3-33
3-31 Fully Asynchronous Write Cycle 3-33
3-32 Pseudo-Asynchronous Write Cycle 3-34
3-33 Pseudo-Asynchronous Read Cycle 3-34
3-34 Synchronous Read Cycle 3-37
3-35 Synchronous Write Cycle 3-37
4-1 Programming Model 4-2
4-2 Status Register 4-3

4-3 General Form of Exception Stack Frame 4-10
4-4 General Exception Processing Flowchart 4-11
4-5 Exception Vector Format 4-12
4-6 Address Translated from 8-Bit Vector Number 4-12
4-7 Supervisor Stack Order for Bus or Address Error Exception 4-19
5-1 Module Base Address, Decode Logic 5-3
5-2 Chip-Select Block Diagram 5-6
5-3 External Bus Interface Logic 5-9
5-4 Interrupt Controller Logic Block Diagram 5-15
6-1 Timer Block Diagram 6-2
7-1 M-Bus Interface Block Diagram 7-2
7-2 M-Bus Transmission Signals 7-3
7-3 M-Bus Clock Synchronization 7-5
7-4 Flow-Chart of Typical M-Bus Interrupt Routine 7-14
8-1 Simplified Block Diagram 8-1
8-2 External and Internal Interface Signals 8-4
8-3 Baud Rate Generator Block Diagram 8-5
8-4 Transmitter and Receiver Functional Diagram 8-6
8-5 Transmitter Timing Diagram 8-7
8-6 Receiver Timing Diagram 8-9
8-7 Looping Modes Functional Diagram 8-11
8-8 Multidrop Mode Timing Diagram 8-13
8-9 Serial Mode Programming Flowchart 8-32
9-1 Test Access Port Block Diagram 9-2
9-2 TAP Controller State Machine 9-3
9-3 Output Cell (O.Cell) 9-6
9-4 Input Cell (I.Cell) 9-7
9-5 Output Control Cell (En.Cell) 9-7
9-6 Bidirectional Cell (IO.Cell) 9-8
9-7 Bidirectional Cell (IOx0.Cell) 9-8
9-8 General Arrangement for Bidirectional Pins 9-9
9-9 Bypass Register 9-10
10-1 MC68307 Minimum System Configuration 10-3
10-2 Hardware Setup 10-13
10-3 Master/Slave Responsibilities for the Master Transmit Block 10-15
10-4 Summary of M-Bus Activity for the Master Transmit Block 10-15
10-5 Master/Slave Responsibilities for the Master Receive Block 10-16
10-6 Summary of M-Bus Activity for the Master Receive Block 10-16
10-7 Memory Map after Swap Complete 10-28
11-1 Drive Levels and Test Points for AC Specifications 11-3
11-2 Clock Timing 11-5
11-3 Read Cycle Timing Diagram 11-7
11-4 Write Cycle Timing Diagram 11-8
11-5 Three-Wire Bus Arbitration Diagram 11-9
11-6 Two-Wire Bus Arbitration Timing Diagram 11-10

11-7 External 8051 Bus Read Cycle 11-12
11-8 External 8051 Bus Write Cycle 11-12
11-9 Timer Module Timing Diagram 11-13
11-10 Transmitter Timing 11-14
11-11 Receiver Timing 11-14
11-12 M-Bus Interface Input/Output Signal Timing 11-15
11-13 Port Timing 11-16
11-14 Test Clock Input Timing Diagram 11-17
11-15 Boundary Scan Timing Diagram 11-18
11-16 Test Access Port Timing Diagram 11-18

LIST OF TABLES

2-1 68000 Bus Signal Summary 2-3
2-2 8051 Bus Signal Summary 2-3
2-3 Chip Select Signal Summary 2-3
2-4 Interrupt Port Signal Summary 2-4
2-5 Clock and Mode Control Signal Summary 2-4
2-6 Serial Module Signal Summary 2-4
2-7 JTAG Signal Summary 2-4
2-8 Timer Module Signal Summary 2-5
2-9 M-Bus Module Signal Summary 2-5
2-10 Data Strobe Control of Data Bus 2-8
2-11 Signal Index 2-14
4-1 Processor Data Formats 4-3
4-2 Effective Addressing Modes 4-4
4-3 Notation Conventions 4-4
4-4 EC000 Core Instruction Set Summary 4-6
4-5 Exception Vector Assignments 4-13
4-6 Exception Grouping and Priority 4-19
5-1 Address Block Selection in Peripheral Chip Select Mode 5-7
5-2 Port A Pin Functions 5-12
5-3 Port B Pin Functions 5-12
5-4 Interrupt Vector Response 5-16
5-5 MC68307 Configuration Memory Map 5-20
5-6 $\overline{\text{DTACK}}$ Field Encoding 5-32
7-1 M-Bus Prescaler Values 7-7
8-1 Serial Module Programming Model 8-15
8-2 PMx and PT Control Bits 8-16
8-3 B/Cx Control Bits 8-17
8-4 CMx Control Bits 8-17
8-5 SBx Control Bits 8-18
8-6 RCSx Control Bits 8-21
8-7 TCSx Control Bits 8-22
8-8 MISCx Control Bits 8-23
8-9 TCx Control Bits 8-24
8-10 RCx Control Bits 8-24
8-11 Timer Mode and Source Select Bits 8-27
9-1 Boundary Scan Control Bits 9-4
9-2 Boundary Scan Bit Definitions 9-5
9-3 Instructions 9-9
10-1 Power Contribution from Modules 10-7

SECTION 1
INTRODUCTION

The MC68307 is an integrated processor combining a static EC000 processor with multiple interchip bus interfaces. The MC68307 is designed to provide optimal integration and performance for applications such as digital cordless telephones, portable measuring equipment, and point-of-sale terminals. By providing 3.3 V, static operation in a small package, the MC68307 delivers cost-effective performance to handheld, battery-powered applications.

The MC68307 (see Figure 1-1) contains a static EC000 core processor, multiple bus interfaces, a serial channel, two timers, and common system glue logic. The multiple bus interfaces include: dynamic 68000 bus, 8051-compatible bus, and Motorola bus (M-bus) or I^2C bus[1].

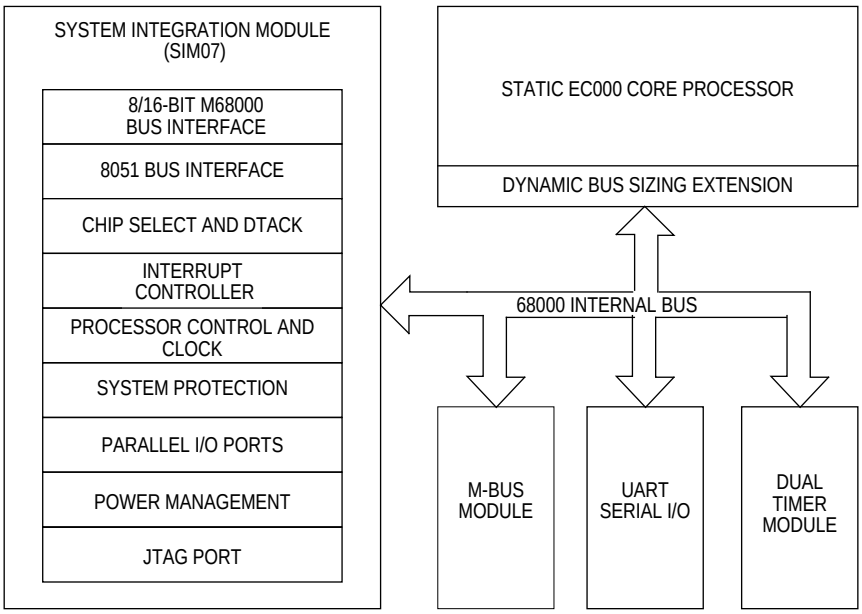


Figure 1-1. MC68307 Block Diagram

The dynamically-sized 68000 bus allows 16-bit performance using static random access memory (SRAM) while providing a low-cost interface to an 8-bit read-only memory (ROM). The 8051-compatible bus interfaces gluelessly to 8051-type devices and allows the reuse

1. I^2C bus is a proprietary Philips interface bus.

of application-specific integrated circuits (ASICs) designed for this industry standard bus. The M-bus is an industry-standard 2-wire interface that provides efficient communications with peripherals such as EEPROM, analog/digital (A/D) converters, and liquid crystal display (LCD) drivers. Thus, the MC68307 interfaces gluelessly to boot ROM, SRAM, 8051 devices, M-bus devices, and memory-mapped peripherals. The MC68307 also incorporates a slave mode which allows the EC000 core to be disabled, providing a 3.3-V or 5-V static, low-power multifunction peripheral for higher performance M68000 family processors.

The main features of the MC68307 include:

- Static EC000 Core Processor
  - Full Compatibility with M68000 and EC000
  - 24-Bit Address Bus, for 16-Mbyte Off-Chip Address Space
  - 16-Bit On-Chip Data Bus for M68000 Bus Operations
  - 2.7 MIPS Performance at 16.67 MHz Processor Clock
  - Processor Disable Mmode for Use as a Peripheral to an External Processor
  - Emulation Mode for Use with In-Circuit Emulator
- External M68000 Bus Interface with Dynamic Bus Sizing for 8-Bit and 16-Bit Data Ports
- External 8051-Compatible 8-Bit Data Bus Interface
- Power Management
  - Fully Static Operation with Processor Shutdown and Wake-Up Modes for Substatial Power Savings
  - Very Rapid Response to Interrupts from the Power-Down State
  - Operates from DC to 16.67 MHz System Clock
  - Clock Enable/Disable for Each Peripheral
- M-Bus Module
  - Provides Interchip Bus Interface for EEPROMs, LCD Controllers, A/D Converters, etc.
  - Compatible with Industry-Standard I^2C Bus
  - Master or Slave Operation Modes, Supports Multiple Masters
  - Automatic interrupt Generation with Programmable Level
  - Software-Programmable Clock Frequency
  - Data Rates from 4–100 Kbit/s above 3.0 MHz System Clock
- Universal Asynchronous Receiver/Transmitter (UART) Module
  - Flexible Baud Rate Generator
  - Based on MC68681 Dual Universal Asynchronous Receiver/Transmitter (DUART) Programming Model
  - 5 Mbits/s Maximum Transfer Rate at 16.67 MHz System Clock
  - Automatic Interrupt Generation with Programmable Level
  - Modem Control Signals Available ($\overline{\text{CTS}}$, $\overline{\text{RTS}}$)
- Timer Module
  - Dual Channel 16-Bit General-Purpose Counter/Timer
  - Multimode Operation, Independent Capture/Compare Registers
  - Automatic Interrupt Generation with Programmable Level
  - 60-ns Resolution at 16.67 MHz System Clock
  - Separate Input and Output Pins for Each Timer

- System Integration Module (SIM07), Incorporating Many Functions Typically Relegated to External Programmable Array Logic (PALs), Transistor-Transistor Logic (TTL), and ASICs, Such as:
  —System Configuration, Programmable Address Mapping
  —System Protection by Hardware Watchdog Logic and Software Watchdog Timer
  —Power-Down Mode Control, Programmable Processor Clock Driver
  —Four Programmable Chip Selects with Wait State Generation Logic
  —Three Simple Peripheral Chip Selects
  —Parallel Input/Output Ports, Some with Interrupt Capability
  —Programmed Interrupt Vector Response for On-Chip Peripheral Modules
  —IEEE 1149.1 Boundary Scan Test Access Port (JTAG)
- Operating Voltages of 3.3V ± 0.3V and 5V ± 0.5V
- 0 to 70°C (Standard part); –40 to +85°C (Extended Temperature Part)
- Compact 100-Lead Quad Flat Pack (QFP) and 100-Lead Thin Quad Flat Pack (TQFP) Packages

1.1 M68300 FAMILY

The MC68307 is one of a series of components in Motorola's M68300 family. Other members of the family include the MC68302, MC68306, MC68322, MC68330, MC68331, MC68332, MC68F333, MC68334, MC68340, MC68341, MC68349, MC68356, and MC68360.

1.1.1 Organization

The M68300 family of integrated processors and controllers is built on an M68000 core processor and a selection of intelligent peripherals appropriate for a set of applications. Common system glue logic such as address decoding, wait state insertion, interrupt prioritization, and watchdog timing is also included.

Each member of the M68300 family is distinguished by its selection of on-chip peripherals. Peripherals are chosen to address specific applications, but are often useful in a wide variety of applications. The peripherals may be highly sophisticated timing or protocol engines that have their own processors, or they may be more traditional peripheral functions, such as UARTs and timers.

1.1.2 Advantages

By incorporating so many major features into a single M68300 family chip, a system designer can realize significant savings in design time, power consumption, cost, board space, pin count, and programming. The equivalent functionality can easily require 20 separate components. Each component might have 16–64 pins, totalling over 350 connections. Most of these connections require interconnects or are duplications. Each connection is a candidate for a bad solder joint or misrouted trace. Each component is another part to qualify, purchase, inventory, and maintain. Each component requires a share of the printed circuit board. Each component draws power, which is often used to drive large buffers to get the signal to another chip. The cumulative power consumption of all the components must be available from the power supply. The signals between the central processing unit (CPU)

and a peripheral might not be compatible nor run from the same clock, requiring time delays or other special design considerations.

In an M68300 family component, the major functions and glue logic are all properly connected internally, timed with the same clock, fully tested, and uniformly documented. Only essential signals are brought out to pins. The primary package is the surface-mount plastic QFP for the smallest possible footprint.

1.2 MC68307 ARCHITECTURE

To improve total system throughput and reduce part count, board size and cost of system implementation, the MC68307 integrates a powerful processor, intelligent peripheral modules, and typical system interface logic. These functions include the SIM07, timers, UART, M-bus interface, and 8051-compatible bus interface.

The EC000 core processor communicates with these modules via an internal bus, providing the opportunity for fully synchronized communication between all modules and allowing interrupts to be handled in parallel with data transfers, greatly improving system performance.

1.2.1 EC000 Core Processor

The EC000 is a core implementation of the M68000 32-bit microprocessor architecture. The programmer can use any of the eight 32-bit data registers for fast manipulation of data and any of the eight 32-bit address registers for indexing data in memory. Flexible instructions support data movement, arithmetic functions, logical operations, shifts and rotates, bit set and clear, conditional and unconditional program branches, and overall system control.

The EC000 core can operate on data types of single bits, binary-coded decimal (BCD) digits, and 8, 16, and 32 bits. The integrated chip selects allow peripherals and data in memory to reside anywhere in the 16-Mbyte linear address space. A supervisor operating mode protects system-level resources from the more restricted user mode, allowing a true virtual environment to be developed. Many addressing modes complement these instructions, including predecrement and postincrement, which allow simple stack and queue maintenance and scaled indexing for efficient table accesses. Data types and addressing modes are supported orthogonally by all data operations and with all appropriate addressing modes. Position-independent code is easily written.

Like all M68000 family processors, the EC000 core recognizes interrupts of seven different priority levels and, in conjunction with the integrated interrupt controller, allows a programmed vector to direct the processor to the desired service routine. Internal trap exceptions ensure proper instruction execution with good addresses and data, allow operating system intervention in special situations, and permit instruction tracing. The hardware timeout can terminate bad memory accesses before instructions process data incorrectly. The EC000 core provides 2.7 millions of bits per second (MIPS) at 16.67 MHz.

1.2.2 System Integration Module (SIM07)

The SIM07 provides the external bus interface for the EC000 core. It also eliminates much of the glue logic that typically supports a microprocessor and its interface with the peripheral

and memory system. The SIM07 provides programmable circuits to perform address-decoding and chip selects, wait-state insertion, interrupt handling, clock generation, discrete I/O, and power management features.

1.2.2.1 EXTERNAL BUS INTERFACE. The external bus interface (EBI) handles the transfer of information between the internal EC000 core and memory, peripherals, or other processing elements in the external address space. It consists of an M68000 bus interface and an 8051-compatible bus interface. The external M68000 bus provides up to 24 address lines and 16 data lines. Each bus access can appear externally either as an M68000 bus cycle (either 16-bit or 8-bit dynamic data bus width) or an 8-bit wide 8051-compatible bus cycle (multiplexing address and 8-bit data) with the respective sets of control signals. The default bus cycle is an M68000 16-bit-wide one, the 8051-compatible address space being programmable.

1.2.2.2 CHIP SELECT AND WAIT STATE GENERATION. Four programmable chip select outputs provide signals to enable external memory and peripheral circuits, providing all handshaking and timing signals for automatic wait-state insertion and data bus sizing.

Base memory address and block size are both programmable, with some restrictions, (e.g. a starting address must be on a boundary which is a multiple of the block size). Each chip-select is general-purpose. However one of the chip-selects can be programmed to select an addressing range which is decoded as an 8051-compatible bus access, and another can be used to enable one of four simple external peripherals. Data bus width (8-bit or 16-bit) is programmable on all four chip-selects and further decoding is available for protection from user mode access or read-only access.

1.2.2.3 SYSTEM CONFIGURATION AND PROTECTION. The SIM07 provides configuration registers that allow general system functions to be controlled and monitored. For example, all on-chip registers can be relocated as a block by programming a module base address, power-down modes can be selected, and the source of the most recent RESET or BERR can be checked. The hardware watchdog features can be enabled or disabled and the bus timeout times can be programmed.

The power-down mode allows software to disable the EC000 core during periods of inactivity. This feature works in conjunction with the interrupt control logic to allow any interrupt condition to cause a wake-up, which causes the EC000 core to resume processing without requiring any RESET or re-initialization. All register contents are preserved, and the interrupt which caused wake-up is serviced in the normal manner as soon as the clock restarts. To reduce power consumption further, the internal clocks to the on-chip peripheral modules can be disabled by software.

1.2.2.4 PARALLEL INPUT/OUTPUT PORTS. Two general-purpose ports (A and B) are provided for input/output, although the pins for these ports are shared with other functions. Some bits in port A are multiplexed with the peripheral chip-select lines and timer output signals, and port B is multiplexed with various peripheral I/O lines from the UART, M-Bus and Timer modules. Maximum flexibility is therefore available for differing hardware configurations. Eight of the 16 Port B lines are also latched inputs to the interrupt controller, with programmable interrupt priority level.

1.2.2.5 INTERRUPT CONTROLLER. The SIM07 coordinates all interrupt sources, both from on-chip peripherals (timer1, timer2, M-bus, and UART) and off-chip inputs ($\overline{IRQ7}$, and $\overline{INT1}$–$\overline{INT8}$). It provides interrupt requests to the EC000 core with programmable priority level, and responds to acknowledge cycles by providing programmed vectors unique to each source.

1.2.3 Timer Module

The timer module comprises two independent, identical general-purpose timers, and a software watchdog timer. Each general-purpose timer block contains a free-running 16-bit timer that can be used in various modes, to capture the timer value with an external event, to trigger an external event or interrupt when the timer reaches a set value, or to count external events. Each has an 8-bit prescaler to allow a programmable clock input frequency derived from the system clock or external count input. The output pins (one per timer) have a programmable mode.

A software watchdog timer is also provided for system protection. If required, this resets the EC000 core if it is not refreshed periodically by software.

1.2.4 UART Module

The MC68307 contains a full-duplex UART module, with an on-chip baud-generator providing both standard and nonstandard baud rates up to 5 Mb/s. The module is functionally equivalent to the MC68681 DUART, although only one serial channel is implemented. Data formats can be 5, 6, 7, or 8 bits with even, odd, or no parity and up to two stop bits in 1/16 increments. Four-byte receive buffers and two-byte transmit buffers minimize CPU service calls. A wide variety of error detection and maskable interrupt capability is provided on each channel. Full-duplex autoecho loopback, local loopback, and remote loopback modes can be selected. Multidrop applications are supported.

Clocking is provided by the MC68307 system clock, via a programmable prescaler allowing various baud rates to be chosen. Modem support is provided with request-to-send ($\overline{RTS}$) and clear-to-send ($\overline{CTS}$) lines available. The serial port can sustain data rates of 5 Mbps.

1.2.5 M-Bus Module

The M-bus interface module provides a two-wire, bidirectional serial bus which provides a simple, efficient way for data exchange between devices. It is compatible with the I^2C-bus standard. M-bus minimizes the interconnection between devices in the end-system. It is best suited for applications which need occasional bursts of fast communication over a short distance, among a number of devices. The maximum data rate is limited to 595 kbps at 16.67 MHz (100 kbps to be compatible with I^2C), the maximum communication length and number of devices which can be connected are limited physically by the maximum bus capacitance and logically by the number of unique addresses.

The M-bus system is a true multimaster bus including collision detection and arbitration to prevent data corruption if two or more masters intend to control the bus simultaneously. This feature provides the capability for complex applications with multiprocessor control. It may also be used for rapid testing and alignment of end products via external diagnostic connections.

1.2.6 Test Access Port

To aid in system diagnostics, the MC68307 includes dedicated user-accessible test logic that is compliant with the IEEE 1149.1 standard for boundary scan testability, often referred to as JTAG (Joint Test Action Group). This is described briefly in **Section 9 IEEE 1149.1 Test Access Port**. For further information refer to the IEEE 1149.1 standard.

SECTION 2
SIGNAL DESCRIPTION

This section contains a brief description of the input and output signals, with reference (if applicable) to other sections which give greater detail on its use. Figure 2-1 provides a detailed diagram showing the integrated peripherals and signals, and Table 2-2 to Table 2-9 provide a quick reference for determining a signal's name, mnemonic, its use as an input or output, active state, and type identification.

NOTE

The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing with a mixture of “active-low” and “active-high” signals. The term assert or assertion is used to indicate that a signal is active or true, independent of whether that level is represented by a high or low voltage. The term negate or negation is used to indicate that a signal is inactive or false.

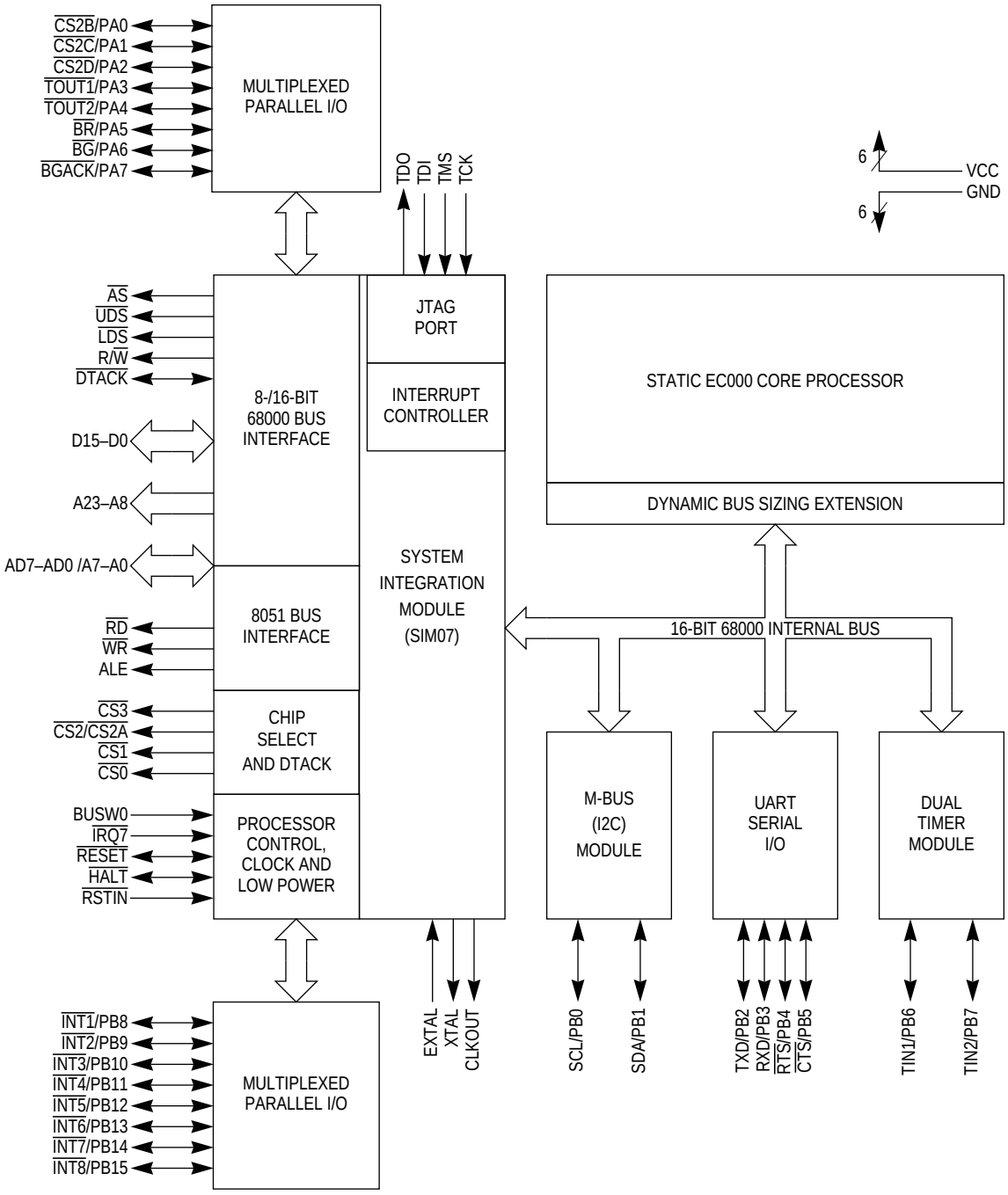


Figure 2-1. MC68307 Detailed Block Diagram

Table 2-1. 68000 Bus Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Address Signals	A23–A0	Output	Yes	(4)
Address Strobe	$\overline{AS}$	Output	Yes	(3)
Bus Grant	$\overline{BG}$/PA6	Output (2)	No	—
Bus Grant Acknowledge	$\overline{BGACK}$/PA7	Input (2)	—	(1)
Bus Request	$\overline{BR}$/PA5	Input (2)	—	(1)
Data Bus	D15–D0	I/O	Yes	—
Data Transfer Acknowledge	$\overline{DTACK}$	I/O	Yes	2.2 KΩ
Halt	$\overline{HALT}$	I/O	—	2.2 KΩ
Lower Data Strobe	$\overline{LDS}$	Output	Yes	(3)
Upper Data Strobe	$\overline{UDS}$	Output	Yes	(3)
Read/Write	R/$\overline{W}$	Output	Yes	(3)
Reset	$\overline{RESET}$	I/O	—	2.2 KΩ

Table 2-2. 8051 Bus Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Address/Data Bus	AD7–AD0	I/O	Yes	—
Address Latch Enable	ALE	Output	No	—
8051 Read Strobe	$\overline{RD}$	Output	No	—
8051 Write Strobe	$\overline{WR}$	Output	No	—

NOTES:
1. Pullup may be required (value depends on individual application). This pin must not be left floating.
2. These signals have dual functions as port A I/O lines. Their function is programmed in the port A control register.
3. A pull-up is required if the output is decoded, otherwise leave unconnected. (Value of pull-up resistor depends on individual application.
4. A pull-up is required on A23 if it is decoded. Pull-ups on whole bus minimize sleep mode power consumption.

Table 2-3. Chip Select Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Chip Select	$\overline{CS3}$–$\overline{CS0}$	Output	No	—
Peripheral Chip Select	$\overline{CS2D}$/PA0, $\overline{CS2C}$/ PA1, $\overline{CS2B}$/PA2	Output (1)	No	—

NOTE:
1. These signals have dual functions as port A I/O lines. Their function is programmed in the port A control register.

Table 2-4. Interrupt Port Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Interrupt Request Level 7	$\overline{\text{IRQ7}}$	Input	—	(1)
Latched Interrupt 1	$\overline{\text{INT1}}$/PB8	Input (2)	—	(1)
Latched Interrupt 2	$\overline{\text{INT2}}$/PB9	Input (2)	—	(1)
Latched Interrupt 3	$\overline{\text{INT3}}$/PB10	Input (2)	—	(1)
Latched Interrupt 4	$\overline{\text{INT4}}$/PB11	Input (2)	—	(1)
Latched Interrupt 5	$\overline{\text{INT5}}$/PB12	Input (2)	—	(1)
Latched Interrupt 6	$\overline{\text{INT6}}$/PB13	Input (2)	—	(1)
Latched Interrupt 7	$\overline{\text{INT7}}$/PB14	Input (2)	—	(1)
Latched Interrupt 8	$\overline{\text{INT8}}$/PB15	Input (2)	—	(1)

NOTES:
1. Pullup or pulldown may be required (value depends on individual application).
2. These signals have dual functions as port B I/O lines. Their function is programmed in the port B control register.

Table 2-5. Clock and Mode Control Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Crystal Oscillator or External Clock	EXTAL	Input	—	—
Crystal Oscillator	XTAL	Output	—	—
System Clock	CLKOUT	Output	No	—
Initial Bus Width for $\overline{\text{CS0}}$	BUSW0	Input	—	—
Power-On Reset	$\overline{\text{RSTIN}}$	Input	—	(1)

NOTE:
1. Pullup may be required (value depends on individual application). This pin must not be left floating.

Table 2-6. Serial Module Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
UART Transmitter Serial Data	TxD/PB2	Output (1)	No	—
UART Receiver Serial Data	RxD/PB3	Input (1)	—	—
UART Request-to-Send	$\overline{\text{RTSA}}$/$\overline{\text{PB4}}$	Output (1)	No	—
UART Clear-to-Send	$\overline{\text{CTSA}}$/$\overline{\text{PB5}}$	Input (1)	—	(2)

NOTES:
1. These signals have dual functions as port B I/O lines. Their function is programmed in the port B control register.
2. Pullup may be required (value depends on individual application). If used as $\overline{\text{CTSA}}$ this pin must not be left floating.

Table 2-7. JTAG Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Test Clock	TCK	Input	—	—
Test Data Input	TDI	Input	—	—
Test Data Output	TDO	Output	—	—
Test Mode Select	TMS	Input	—	—

Table 2-8. Timer Module Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
Timer Input 1	TIN1/PB6	Input (1)	—	(2)
Timer Input 2	TIN2/PB7	Input (1)	—	(2)
Timer Output 1	TOUT1/PA3	Output (1)	—	—
Timer Output 2	TOUT2/PA4	Output (1)	—	—

NOTES:
1. These signals have dual functions as port A and port B I/O lines. Their function is programmed in the port A or port B control register.
2. Pullup may be required (value depends on individual application). If used as timer inputs, these pins must not be left floating.

Table 2-9. M-Bus Module Signal Summary

Signal Name	Mnemonic	Input/ Output	Three-State During Bus Arbitration	Pullup Resistor Required
M-Bus clock	SCL/PB0	I/O (1)	—	2.2 KΩ
M-Bus data	SDA/PB1	I/O (1)	—	2.2 KΩ

NOTE:
1. These signals have dual functions as port B I/O lines. Their function is programmed in the port B control register.

2.1 BUS SIGNALS

The following signals are used for the MC68307 bus.

2.1.1 Address Bus (A23–A0)

The address bus signals are outputs that define the address of a byte (or the most significant byte) to be transferred during a bus cycle. The MC68307 places the address on the bus at the beginning of a bus cycle, it is valid while the address strobe output ($\overline{AS}$) is asserted. The complete address bus (A23–A0) is capable of addressing 16 Mbytes of data.

The address bus signals are three-stated when the MC68307 is arbitrated off the bus by an external bus master. They are also three-stated during reset of the EC000 core.

The address bus consists of the following groups of signals. Refer to **Section 3 Bus Operation** for information on the address bus and its relationship to bus operation.

2.1.1.1 ADDRESS BUS (A23–A8). These signals are always used as address output lines. They are valid when address strobe ($\overline{AS}$) is asserted. Chip-selected memory and peripherals, including 8051-compatible bus devices, need not decode the upper address bits, depending on the programmed block size for that chip select.

2.1.1.2 ADDRESS BUS (AD7–AD0). In addition to carrying eight address signals A7–A0, these low-order address lines also carry the 8-bit data bus during 8051-compatible bus cycles, as indicated by the relevant strobe signals ($\overline{RD}$ and $\overline{WR}$). At the start of each type of bus cycle (both M68000 and 8051-compatible), they carry the eight low-order address bits A7–A0. In M68000 bus cycles, they are valid addresses when address strobe ($\overline{AS}$) is asserted. Chip select 3 ($\overline{CS3}$) should be used to distinguish 8051 bus cycles if necessary.

Address line A0 is included for 8-bit data-bus interfaces only, i.e., the 8051-compatible bus, as described above, and also the dynamically-sized 68000 bus when programmed to use an 8-bit data-bus width. During 16-bit data-bus width cycles, the A0 output is meaningless, and should be ignored, as it may well hold a misleading value. The $\overline{UDS}$ and $\overline{LDS}$ signals should be used to further decode the even/odd byte access in this case.

2.1.2 Data Bus (D15–D0)

This 16-bit bidirectional parallel bus contains the data being transferred to or from the MC68307 during M68000 bus cycles. Its value should be ignored during 8051-compatible bus read and write cycles, it is not three-stated in either case. A read or write operation may transfer 8 or 16 bits of data (one or two bytes) in one bus cycle.

During an internal peripheral access, the data bus reflects the value read or written, for emulation and debug purposes. Care is required, therefore, if external buffers are needed.

The data bus has a programmable 8-bit bus size option for M68000 bus cycles, which is used in conjunction with the programmable chip-select dynamic bus sizing. If a chip select is configured for 8-bit port size, any 16-bit transfers appears externally as two 8-bit transfers. In this case, the 8-bit data uses the D15–D8 lines only.

2.2 CHIP SELECTS

The programmable chip select outputs allow system designers to interface the MC68307 directly to memory and peripheral devices without having to perform address decode requiring additional external logic. Although they can be programmed for many different configurations, each one has a particular usage to which it is tailored, giving added functionality. They are asserted coincident with the address strobe ($\overline{AS}$) output, and are all active-low. Refer to **Section 5 System Integration Module** for details of how the chip selects can be programmed.

2.2.1 Chip Select 0 ($\overline{CS0}$)

This signal is the chip select for a boot ROM containing the reset vectors and initialization program. From a cold reset, this chip select is asserted on every bus cycle in the first 8K bytes of address space until is is programmed otherwise. The BUSW0 pin specifies how this chip select behaves from cold reset, whether it is an 8-bit wide data bus access (BUSW0=0) or a 16-bit wide data bus access (BUSW0=1).

2.2.2 Chip Select 1 ($\overline{CS1}$)

This signal is primarily intended to be an enable for a RAM memory device. From a cold reset, this signal does not assert until it is programmed with a valid base address and address mask. The data bus width for memory devices selected by this chip select is also programmable, between 8- and 16-bit data, in the system control register.

2.2.3 Chip Select 2 ($\overline{CS2}$, $\overline{CS2B}$, $\overline{CS2C}$, $\overline{CS2D}$)

Although the $\overline{CS2}$/$\overline{CS2A}$ pin can be used as a general-purpose M68000 chip-select ($\overline{CS2}$), it can, together with the $\overline{CS2B}$, $\overline{CS2C}$, $\overline{CS2D}$ signals, be used to enable one of four miscellaneous peripherals. The $\overline{CS2}$ output address range can be relocated like any of the other

chip selects by programming, but it has the three extra enable outputs which are used to further divide this address space. If they are used, each of the four peripheral enable outputs decode fixed 16-Kbyte address ranges within the confines of the programmed range of $\overline{CS2}$ (which should therefore be a block 64K bytes in size). Refer to **Section 5.1.2.2 Peripheral Chip Selects** and **Section 5.2.2 Chip Select Registers** for details. The data bus width for memory devices selected by this chip select is programmable, between 8- and 16-bit data, in the system control register.

The $\overline{CS2}$/$\overline{CS2A}$ chip select signal has its own pin on the MC68307, but the three extra enable outputs $\overline{CS2B}$, $\overline{CS2C}$, $\overline{CS2D}$ are multiplexed with the port A input/output functions, and require to be programmed to use these pins at cold reset. This is done by setting bits in the port A control register (PACNT) as well as a bit in the system control register to enable the peripheral address decoder. As port A defaults to an input port on cold reset, these three additional peripheral chip select lines, if they are used, should have pullup resistors to ensure peripherals are not accidentally enabled during system initialization.

When programmed as general-purpose input/output port lines, $\overline{CS2B}$–$\overline{CS2D}$ function as PA0–PA2.

2.2.4 Chip Select 3 ($\overline{CS3}$)

This chip select output can be programmed to perform an 8051-compatible bus cycle rather than a M68000 bus cycle. If 8051-compatible bus access is not required in a design, then this signal can be programmed to be a general purpose chip-select output. The data bus width for M68000-bus memory devices selected by this chip select is programmable, between 8- and 16-bit data, in the system control register. If the 8051-compatible bus interface is enabled, then the data-bus width for this chip select should be programmed to be 8 bits. The 8051-compatible bus has an address space (typically 64 Kbytes long, but not restricted) which can be located anywhere in memory, as long as it does not overlap with other programmed chip select ranges.

2.3 BUS CONTROL SIGNALS

These are signals that may be decoded or provided by external logic, to control the various types of bus access which can occur.

The bus control signals are three-stated whenever the MC68307 is arbitrated off the bus by an external bus master.

2.3.1 Data Transfer Acknowledge ($\overline{DTACK}$)

This bidirectional, open-drain, active-low signal indicates that the data transfer has been completed. $\overline{DTACK}$ is an output when it is generated internally by the programmable wait-state generators in the chip-select logic (including $\overline{CS3}$ for 8051-compatible bus accesses), or during an access to internal peripheral registers. It is an input for all other M68000 bus cycles, i.e., when the MC68307 accesses an external device not within the range of the chip-select logic or when programmed to be generated externally for any particular chip-select. In this case, external logic must assert it in order to complete the bus cycle.

2.3.2 Address Strobe ($\overline{AS}$)

This three-state active-low output signal indicates that there is a valid address on the address bus during M68000 bus cycles. It should be ignored during 8051-compatible bus cycles as the eight low-order address lines also carry the data bus during such a cycle. During all types of bus cycle, it should be taken into account when other bus masters are arbitrating for the bus, as it indicates that the MC68307 is still using the bus.

When the bus is arbitrated to an external master, or during reset, $\overline{AS}$ is three-stated.

2.3.3 Read/Write (R/$\overline{W}$)

This three-state output signal defines the data-bus transfer as a read or a write cycle. The R/$\overline{W}$ signal relates to the data strobe signals described in the following paragraphs. When the R/$\overline{W}$ line is high, the processor reads from the data bus. When the R/$\overline{W}$ line is low, the processor drives the data bus. The processor also drives the data bus during a read from an internal location, to aid emulation and debug.

When the bus is arbitrated to an external master, or during reset, R/$\overline{W}$ is three-stated.

2.3.4 Data Strobes, Upper and Lower ($\overline{UDS}$, $\overline{LDS}$)

These three-state active-low output signals control the flow of data on the M68000 data bus. Table 2-10 lists the combinations of these signals and the corresponding data on the bus in 16-bit wide mode.

Table 2-10. Data Strobe Control of Data Bus

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D15–D8	D7–D0
High	High	—	No Valid Data	No Valid Data
Low	Low	High	Valid Data Bits 15–8	Valid Data Bits 7–0
High	Low	High	No Valid Data	Valid Data Bits 7–0
Low	High	High	Valid Data Bus 15–8	No Valid Data
Low	Low	Low	Valid Data Bits 15–8	Valid Data Bits 7–0
High	Low	Low	Valid Data Bits 7–0	Valid Data Bits 7–0
Low	High	Low	Valid Data Bits 15–8	Valid Data Bits 15–8

In 8-bit wide bus mode (programmed in conjunction with chip selects), all bus cycles appear as 8-bit reads or writes to the upper half of the data bus (D15–D8), and so $\overline{UDS}$ only is asserted. A0 should be used to determine even or odd byte being addressed in this case; it is valid whenever the external $\overline{AS}$ signal is asserted during such a cycle. There is no external indication of data-bus width other than the chip-select asserted. The user has the knowledge of what bus width that chip-select is programmed to provide, for any given configuration of the MC68307 in a system.

When the bus is arbitrated to an external master, or during reset, $\overline{UDS}$ and $\overline{LDS}$ are three-stated. D7–D0 should be regarded as invalid in 8-bit-wide bus mode.

2.3.5 8051 Address Latch Enable (ALE)

This output signal is used to latch the low byte of address (AD7–AD0 signals) during access to external 8051-compatible peripheral circuits. Its function is tied to $\overline{CS3}$ logic which can be programmed to locate the 8051-compatible bus address space anywhere in the memory map. For a discussion of the timing of the 8051-compatible bus signals, refer to **Section 3 Bus Operation**. ALE is not three-stated during external bus mastership or system reset.

2.3.6 8051-Compatible Bus Read ($\overline{RD}$)

This active-low output indicates that the bus cycle in progress is an 8051 read cycle, and that an addressed 8051 peripheral should provide data on the AD7–AD0 lines within the specified access time. $\overline{RD}$ is not three-stated during external bus mastership or system reset.

2.3.7 8051-Compatible Bus Write ($\overline{WR}$)

This active-low output indicates that the bus cycle in progress is an 8051 write cycle, and that an addressed 8051 peripheral should accept the valid data which is now on the AD7-AD0 lines. $\overline{WR}$ is not three-stated during external bus mastership or system reset.

2.3.8 Bus Width Select for $\overline{CS0}$ (BUSW0)

The state on this input pin is read at reset, and is used to choose the data bus width for memory accesses for $\overline{CS0}$ (Refer to **Section 5.2.1.2 System Control Register (SCR)**). Hold BUSW0 low for an 8-bit data bus, or high for a 16-bit data bus during bus cycles which trigger $\overline{CS0}$. BUSW0 does not choose the bus width for $\overline{CS1}$, $\overline{CS2}$ or $\overline{CS3}$; that is done by user initialization code. Internally, the MC68307 always has a 16-bit bus.

2.4 EXCEPTION CONTROL SIGNALS

The following paragraphs describe the exception control signals.

2.4.1 Reset ($\overline{RESET}$)

The external assertion of this bidirectional active-low signal simultaneously with the assertion of $\overline{HALT}$ starts a system initialization sequence by resetting the whole MC68307 (processor, SIM, and internal peripherals). This is called a cold reset or system reset. The processor assertion of $\overline{RESET}$ (from executing a RESET instruction) resets all external devices of a system and internal peripherals of the MC68307 without affecting the initial state of the processor, chip select logic, port configuration, or Interrupt configuration. This is called a software reset or peripheral reset. Refer to **Section 3 Bus Operation** for further information on reset operation.

During a cold reset, the address bus, data bus, and bus control pins ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$) are all three-stated. Chip select outputs, $\overline{CS3}$–$\overline{CS0}$, remain high. None of these signals are three-stated during a peripheral reset.

2.4.2 Power-On Reset ($\overline{\text{RSTIN}}$)

This active-low input signal causes the MC68307 (processor, SIM, and peripherals) to enter the reset state (cold reset). The assertion of $\overline{\text{RSTIN}}$ causes $\overline{\text{RESET}}$ to be asserted out to reset external circuitry; however, note that $\overline{\text{HALT}}$ is not asserted as a result of $\overline{\text{RSTIN}}$ assertion. Internal power-on reset circuitry provides a reset pulse of at least 32768 clocks width at power-on or when $\overline{\text{RSTIN}}$ is subsequently asserted. This reset pulse length can be extended by adding an external RC network, to ensure that $\overline{\text{RSTIN}}$ is held low for long enough to apply the reset pulse for 128 CPU clocks after V_{CC} and clock are stable. Refer to **Section 10.1 MC68307 Minimum Stand-Alone System Hardware** for an example circuit.

2.4.3 Halt ($\overline{\text{HALT}}$)

This active-low bidirectional signal can be asserted with $\overline{\text{RESET}}$ to cause a cold reset as above. If asserted alone, it causes the processor to stop after completion of the current bus cycle. As long as $\overline{\text{HALT}}$ is held asserted, all bus control signals go to their inactive state, except $\overline{\text{BGACK}}$, and all three-state bus signals (A23–A0, D15–D0 only) are placed in the high-impedance state. When the processor has stopped executing instructions (e.g., after a double bus fault), the MC68307 asserts this signal.

2.4.4 Bus Request ($\overline{\text{BR}}$/PA5)

This input signal indicates to the MC68307 that an external device desires to become the bus master on the MC68307 external bus. Refer to **Section 3 Bus Operation**for details of the bus arbitration features. When programmed as general-purpose input/output, this signal functions as bit 5 of port A.

2.4.5 Bus Grant ($\overline{\text{BG}}$/PA6)

This output signal indicates to all external bus master devices (if any) that the MC68307 releases bus control at the end of the current bus cycle to an external requesting bus master. During cold reset, $\overline{\text{BG}}$ reflects the value of $\overline{\text{BR}}$. When programmed as general-purpose input/output, this signal functions as bit 6 of port A.

2.4.6 Bus Grant Acknowledge ($\overline{\text{BGACK}}$ /PA7)

This input signal indicates that some other device besides the MC68307 has become the bus master. When programmed as general-purpose input/output, this signal functions as bit 7 of port A.

2.5 CLOCK SIGNALS

The following paragraphs describe the clock signals.

2.5.1 Crystal Oscillator (EXTAL, XTAL)

This input provides two clock generation options (crystal and external clock). EXTAL may be used with XTAL to connect an external crystal to the on-chip oscillator and clock generator. If an external clock is used, the clock source should be connected to EXTAL, and XTAL must be left unconnected. The oscillator uses an internal frequency equal to the external crystal frequency. The frequency of EXTAL may range from DC to 16.67 MHz, although if a

crystal is used it should be in the range 1 MHz to 16.67 MHz. When an external clock is used, it must provide a CMOS level at the required system clock frequency.

2.5.2 Clock Output (CLKOUT)

This output clock signal is derived from the on-chip clock oscillator. This clock is used by the processor and the internal peripherals. All MC68307 bus timings are referenced to the CLKOUT signal rather than the EXTAL input signal, as there is a skew between the two. The clock output is active from reset, but can be turned off by the software writing to the system control register, in order to save power or control external devices.

2.6 TEST SIGNALS

The following signals are used with the on-board test logic defined by the IEEE 1149.1 standard. Refer to **Section 9 IEEE 1149.1 Test Access Port** for more information on the use of these signals.

2.6.1 Test Clock (TCK)

This input provides a clock for on-board test logic defined by the IEEE1149.1 standard.

2.6.2 Test Mode Select (TMS)

This input controls test mode operations for on-board test logic defined by the IEEE 1149.1 standard. Connecting TMS to V_{CC} disables the test controller, making all JTAG circuits transparent to the system.

2.6.3 Test Data In (TDI)

This input is used for serial test instructions and test data for on-board test logic defined by the IEEE 1149.1 standard.

2.6.4 Test Data Out (TDO)

This output is used for serial test instructions and test data for on-chip test logic defined by the IEEE 1149.1 standard.

2.7 M-BUS I/O SIGNALS

The M-bus is an I^2C-bus-compatible serial interface on two wires. All devices connected to the bus must have open-drain or open-collector outputs. The logical AND function is exercised on both lines with pullup resistors being required. The pins are multiplexed with port B, individual pin function being programmable. Port B bits 0 and 1 are therefore always open-drain input/outputs. Refer to **Section 7 M-Bus Interface Module** for more information on these signals.

2.7.1 Serial Clock (SCL/PB0)

This bidirectional open-drain signal is the clock signal for the M-bus interface. Either it is driven by the M-bus module when the bus is in the master mode or it becomes the clock input when the M-bus is in the slave mode. When programmed as general-purpose input/output, this signal functions as bit 0 of port B.

2.7.2 Serial Data (SDA/PB1)

This bidirectional open-drain signal is the data input/output for the M-bus interface. When programmed as general-purpose input/output, this signal functions as bit 1 of port B.

2.8 UART I/O SIGNALS

The following signals are used by the UART serial I/O module for data and clock signals. Refer to **Section 8 Serial Module** for more information on these signals.

2.8.1 Transmit Data (TxD/PB2)

This bidirectional signal can be programmed as the transmitter serial data output for the UART module. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal at the falling edge of the serial clock source, with the least significant bit transmitted first. When programmed as a general-purpose input/output, this signal functions as bit 2 of the 16-bit secondary port, port B.

2.8.2 Receive Data (RxD/PB3)

This bidirectional signal can be programmed as the receiver serial data input for the UART module. Data received on this signal is sampled on the rising edge of the serial clock source, with the least significant bit received first. When the UART clock has been stopped for power-down mode, any transition on this pin can optionally restart it. When programmed as a general-purpose input/output, this signal functions as bit 3 of port B.

2.8.3 Request-To-Send ($\overline{\text{RTS}}$/PB4)

This bidirectional signal can be programmed as an active-low request-to-send output from the UART module. When programmed as a general-purpose input/output, this signal functions as bit 4 of port B.

2.8.4 Clear-To-Send ($\overline{\text{CTS}}$/PB5)

This bidirectional input signal can be programmed as the active-low clear-to-send input for the UART module. When programmed as general-purpose input/output, this signal functions as bit 5 of port B.

2.9 TIMER I/O SIGNALS

The following external signals are used by the timer module. Refer to **Section 6 Dual Timer Module** for additional information on these signals.

2.9.1 Timer 1 Input (TIN1/PB6)

This bidirectional signal can be programmed as a clock input that causes events to occur in timer/counter channel 1, either causing a clock to the event counter or providing a trigger to the timer value capture logic. When programmed as a general-purpose input/output, this signal functions as bit 6 of port B.

2.9.2 Timer 2 Input (TIN2/PB7)

This bidirectional signal can be programmed as a clock input that causes events to occur in timer/counter channel 2, either causing a clock to the event counter or providing a trigger to the timer value capture logic. When programmed as general-purpose input/output, this signal functions as bit 7 of port B.

2.9.3 Timer 1 Output ($\overline{\text{TOUT1}}$/PA3)

This bidirectional signal can be programmed to toggle or pulse low for one system clock duration when timer/counter channel 1 reaches a reference value. When programmed as general-purpose I/O, this signal functions as bit 3 of port A.

2.9.4 Timer 2 Output ($\overline{\text{TOUT2}}$/PA4)

This bidirectional signal can be programmed to toggle or pulse low for one system clock duration when timer/counter channel 2 reaches a reference value. When programmed as general-purpose I/O, this signal functions as bit 4 of port A.

2.10 INTERRUPT REQUEST INPUTS

These pins can be programmed to be either prioritized interrupt request lines or port B general-purpose input/output lines.

2.10.1 Interrupt Inputs ($\overline{\text{INT1}}$–$\overline{\text{INT8}}$/PB8–PB15)

These eight bidirectional signals may be configured as general-purpose parallel I/O ports with interrupt capability. Each of the pins can be configured either as an input or an output. When configured as an input, each pin can generate a separate, maskable interrupt on a high-to-low transition, which is latched internally. The interrupt request level (IPL) can be programmed individually for each pin, and each causes a unique vector to be fetched during the subsequent interrupt processing.

2.10.2 Non-Maskable Interrupt Input ($\overline{\text{IRQ7}}$)

$\overline{\text{IRQ7}}$ is the highest priority interrupt available to the EC000 core, it is a nonmaskable interrupt (IPL=7) and cannot be superseded by any other interrupt request. This is an active-low input.

2.11 USE OF PULLUP RESISTORS

In general, pins that are input-only or output-only do not require external pullup resistors. However, see the notes on low power consumption in the following paragraphs.

The open-drain bidirectional signals ($\overline{\text{HALT}}$, $\overline{\text{RESET}}$, $\overline{\text{DTACK}}$, SCL/PB0, SDA/PB1) always require pullup resistors.

The bus control outputs ($\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$) three-state at reset during bus arbitration and low-power sleep mode. They need pullups only if they are decoded for use in the user's design or if minimum power consumption is required, otherwise they can be left as unconnected outputs.

The address and data bus also three-state at reset and during bus arbitration and power-down mode. In this case the designer must decide whether a short period of floating values on the bus is a concern. The internal pullups on the bidirectional data bus pins are not sufficient to drive external levels. Again, adding pullups will help minimize power consumption in low-power sleep mode.

Bidirectional I/O pins default to inputs on reset. If they are to be used as outputs later, or their function changed to an alternate output function for the pin then a pullup or pulldown may be required. One example is the $\overline{CS2B}$, $\overline{CS2C}$, and $\overline{CS2D}$ signals, multiplexed with port A lines. If these chip-selects are used they should have pullups otherwise they will float from reset until they are setup.

The $\overline{RSTIN}$ input can either be left floating or pulled up if the default power-on reset time is required, or can have an external RC network to stretch reset time. Refer to **Section 3.5 Reset Operation** and **Section 10.1 MC68307 Minimum Stand-Alone System Hardware** for details of use of this pin.

For maximum reliability, unused inputs should not be left floating. If they are input-only, they may be pulled to V_{CC} or ground. Unused outputs may be left unconnected. Unused I/O pins may be configured as outputs after reset, and left unconnected.

If the MC68307 is to be held in reset for extended periods of time in an application (other than what occurs in normal power-on reset or board test sequences) due to a special application requirement (such as V_{CC} dropping below required specifications, etc.) then three-stated signals and inputs should be pulled up or down. This decreases stress on the device transistors and saves power.

Refer to **Section 2.4.1 Reset (RESET)** and **Section 3.5 Reset Operation** for the condition of all pins during reset.

Refer to **Section 12 Ordering Information and Mechanical Data** for details of pin assignments.

2.12 SIGNAL INDEX

Table 2-11 lists the signal index.

Table 2-11. Signal Index

Pin Name	Description of Pin Function(s)	Direction
TDO	Test Data Out	Output
TDI	Test Data In	Input
TMS	Test Mode Select	Input
TCK	Test Clock In	Input
D15–D0	Data Bus	Bidirectional
A23–A8	Address Bus Out	Output
A7–A0/AD7–AD0	Address Bus Out/Multiplexed 8051 Address/Data	Bidirectional
$\overline{AS}$	Address Strobe	Output
$\overline{UDS}$	Upper Data Strobe	Output
$\overline{LDS}$	Lower Data Strobe	Output
$\overline{R/W}$	Read/Write	Output

Table 2-11. Signal Index (Continued)

Pin Name	Description of Pin Function(s)	Direction
$\overline{\text{DTACK}}$	Data Acknowledge	Bidirectional
$\overline{\text{HALT}}$	System Halt	Bidirectional
$\overline{\text{RESET}}$	System Reset	Bidirectional
$\overline{\text{RSTIN}}$	Power-On Reset	Input
$\overline{\text{CS0}}$	Chip Select 0 (ROM)	Output
$\overline{\text{CS1}}$	Chip Select 1 (RAM)	Output
$\overline{\text{CS2}}$/$\overline{\text{CS2A}}$	Chip Select 2 (Peripherals)	Output
$\overline{\text{CS3}}$	Chip Select 3 (8051)	Output
ALE	8051-Address Latch Enable	Output
$\overline{\text{RD}}$	8051-Bus Read	Output
$\overline{\text{WR}}$	8051-Bus Write	Output
EXTAL	External Clock/Crystal In	Input
XTAL	External Crystal	Output
CLKOUT	Clock to System	Output
BUSW0	Initial Data Bus Width for $\overline{\text{CS0}}$	Input
$\overline{\text{CS2B}}$/PA0	Chip Select 2B/I/O Port A Bit 0	Bidirectional
$\overline{\text{CS2C}}$/PA1	Chip Select 2C/I/O Port A Bit 1	Bidirectional
$\overline{\text{CS2D}}$/PA2	Chip Select 2D/I/O Port A Bit 2	Bidirectional
$\overline{\text{TOUT1}}$/PA3	Timer 1 Output/I/O Port A Bit 3	Bidirectional
$\overline{\text{TOUT2}}$/PA4	Timer 2 Output/I/O Port A Bit 4	Bidirectional
$\overline{\text{BR}}$/PA5	Bus Request Input/I/O Port A Bit 5	Bidirectional
$\overline{\text{BG}}$/PA6	Bus Grant Output/I/O Port A Bit 6	Bidirectional
$\overline{\text{BGACK}}$/PA7	Bus Grant Acknowledge Output/I/O Port A Bit 7	Bidirectional
$\overline{\text{IRQ7}}$	Interrupt Level 7	Input
SCL/PB0	Serial M-Bus Clock/Port B Bit 0	Bidirectional
SDA/PB1	Serial M-Bus Data/Port B Bit 1	Bidirectional
TxD/PB2	UART Transmit Data/Port B Bit 2	Bidirectional
RxD/PB3	UART Receive Data/Port B Bit 3	Bidirectional
$\overline{\text{RTS}}$/PB4	Request-to-Send/Port B Bit 4	Bidirectional
$\overline{\text{CTS}}$/PB5	Clear-To-Send/Port B Bit 5	Bidirectional
TIN1/PB6	Timer 1 Input/Port B Bit 6	Bidirectional
TIN2/PB7	Timer 2 Input/Port B Bit 7	Bidirectional
$\overline{\text{INT1}}$/PB8	Interrupt In 1/Port B Bit 8	Bidirectional
$\overline{\text{INT2}}$/PB9	Interrupt In 2/Port B Bit 9	Bidirectional
$\overline{\text{INT3}}$/PB10	Interrupt In 3/Port B Bit 10	Bidirectional
$\overline{\text{INT4}}$/PB11	Interrupt In 4/Port B Bit 11	Bidirectional
$\overline{\text{INT5}}$/PB12	Interrupt In 5/Port B Bit 12	Bidirectional
$\overline{\text{INT6}}$/PB13	Interrupt In 6/Port B Bit 13	Bidirectional
$\overline{\text{INT7}}$/PB14	Interrupt In 7/Port B Bit 14	Bidirectional
$\overline{\text{INT8}}$/PB15	Interrupt In 8/Port B Bit 15	Bidirectional

SECTION 3
BUS OPERATION

This section describes control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

NOTE

The terms **assertion** and **negation** are used extensively in this manual to avoid confusion when describing a mixture of “active-low” and “active-high” signals. The term assert or assertion is used to indicate that a signal is active or true, independently of whether that level is represented by a high or low voltage. The term negate or negation is used to indicate that a signal is inactive or false.

3.1 DATA TRANSFER OPERATIONS

Transfer of data between devices involves the following signals:

1. A23–A0
2. D7–D0 and/or D15–D0
3. Control signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cases, the bus master must deskew all signals it issues at both the start and end of a bus cycle. In addition, the bus master must deskew the acknowledge and data signals from the slave device.

The 8051-compatible bus uses A23–A0 for both address and data.

3.1.1 16-Bit M68000 Bus Operation

The M68000 16-bit bus mode of operation is the default mode of the MC68307. The internal data bus width is 16-bits, and the EC000 core always operates in a 16-bit mode.

This mode is appropriate for:

- External 16-bit memory and peripheral devices, which read and write from the whole data bus, or
- 8-bit peripherals which do not require a contiguous address space, that is, they occupy either even byte locations only (if connected to the upper half of the data bus) or odd byte locations only (if connecetd to the lower half of the data bus).

In this mode address bus A23–A1 is used, with data bus D15–D0 and control signals $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$, and $\overline{DTACK}$.

3.1.2 16-Bit M68000 Bus Read Cycle

During a read cycle, the processor receives either one or two bytes of data from the memory or from a peripheral device. If the instruction specifies a word or long-word operation, the processor reads both upper and lower bytes simultaneously by asserting both upper and lower data strobes. A long-word read is accomplished by two consecutive word reads. When the instruction specifies byte operation, the processor uses the internal A0 bit to determine which byte to read and issues the appropriate data strobe. When A0 is zero, the upper data strobe is issued; when A0 is one, the lower data strobe is issued. When the data is received, the processor internally positions the byte appropriately.

The word read cycle flowchart is shown in Figure 3-1. The byte read cycle flowchart is shown in Figure 3-2. The read and write cycle timing is shown in Figure 3-3. Figure 3-4 shows the word and byte read cycle timing diagram.

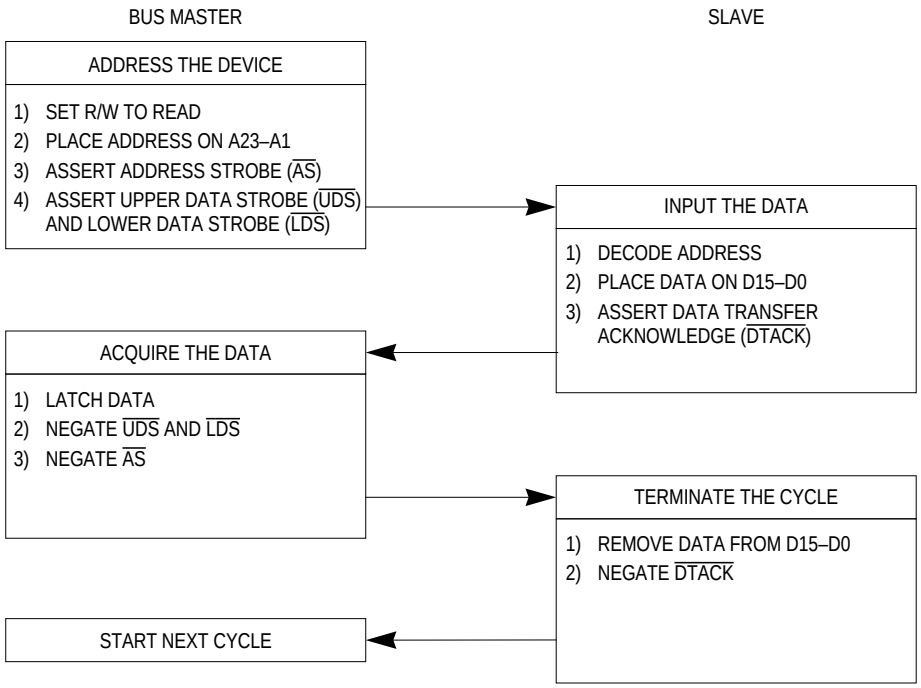


Figure 3-1. Word Read Cycle Flowchart (16-Bit Bus)

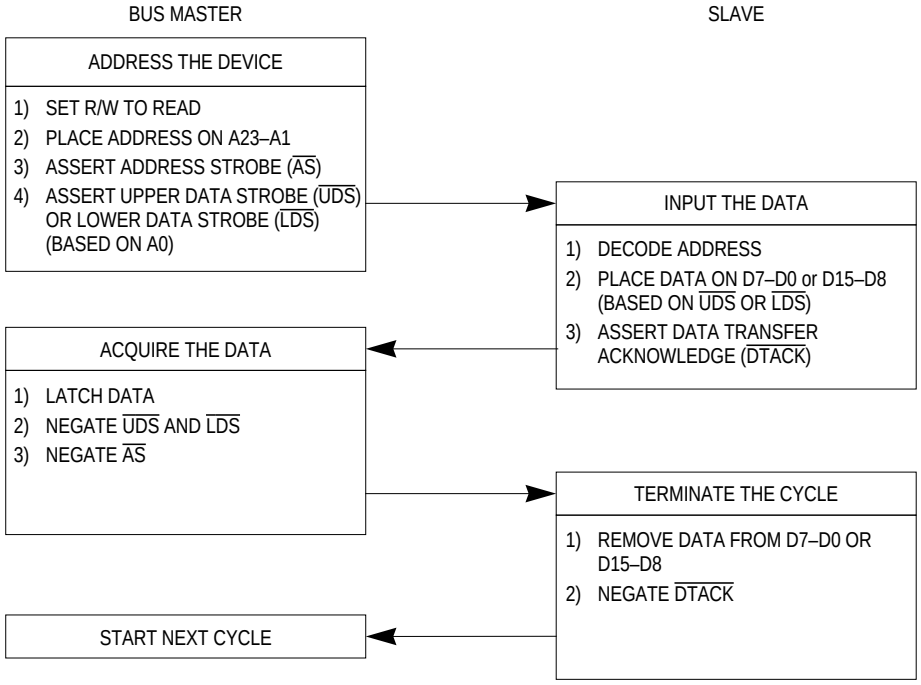


Figure 3-2. Byte Read Cycle Flowchart (16-Bit Bus)

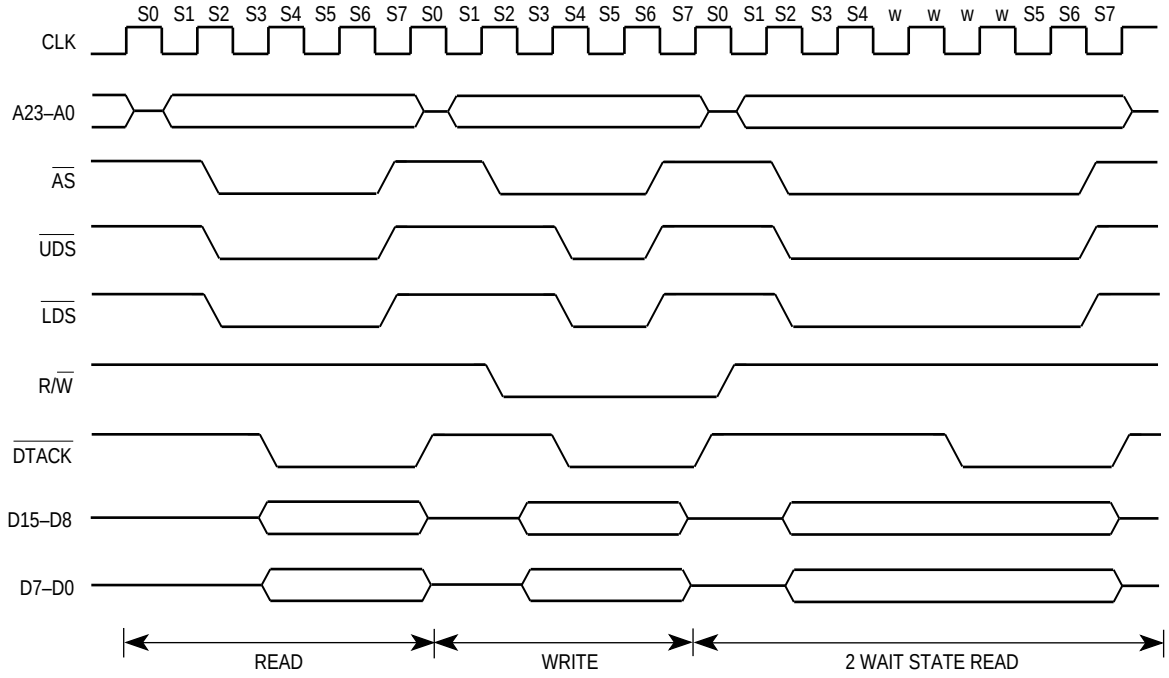


Figure 3-3. Read and Write Cycle Timing Diagram (16-Bit Bus)

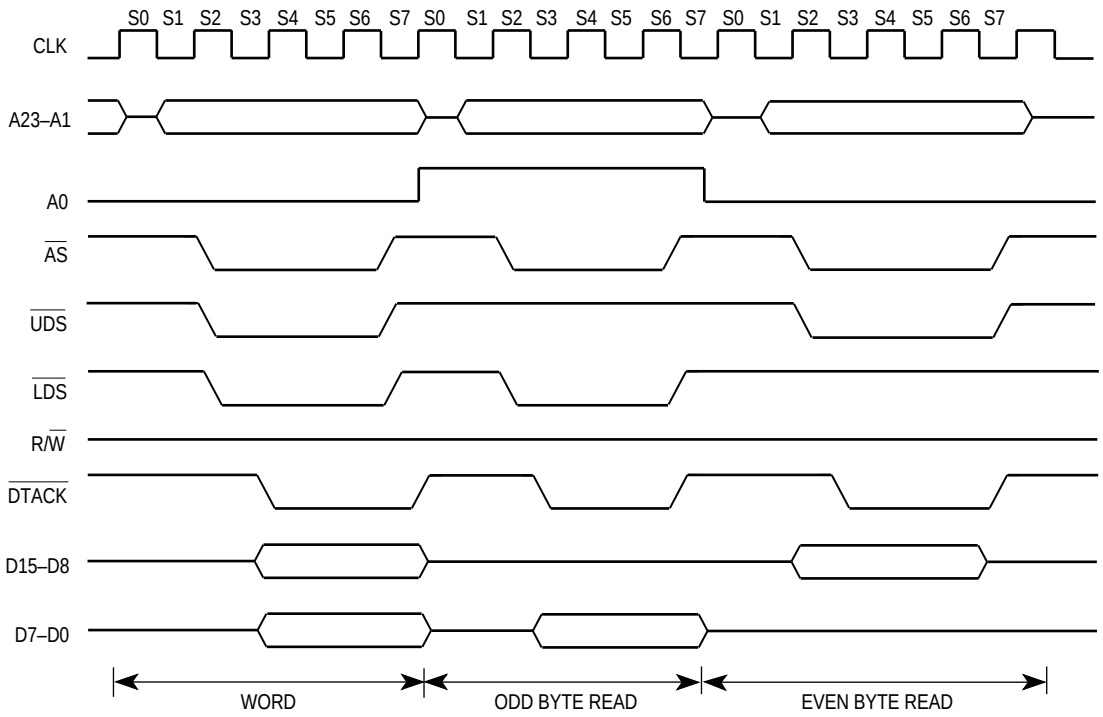


Figure 3-4. Word and Byte Read Cycle Timing Diagram (16-Bit Bus)

A bus cycle consists of eight states. The various signals are asserted during specific states of a read cycle as follows:

STATE 0 The read cycle starts in state 0 (S0). The processor places a valid address on the bus, and drives R/$\overline{W}$ high to identify a read cycle.

STATE 1 During state 1 (S1), no bus signals are altered.

STATE 2 On the rising edge of state 2 (S2), the processor asserts $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$.

STATE 3 During state 3 (S3), no bus signals are altered.

STATE 4 During state 4 (S4), the processor waits for a cycle termination signal ($\overline{DTACK}$ or $\overline{BERR}$). If neither termination signal is asserted before the falling edge at the end of S4, the processor inserts wait states (full clock cycles) until either $\overline{DTACK}$ or $\overline{BERR}$ is asserted. Note that the $\overline{BERR}$ signal does not appear on an external pin on the MC68307; the internal hardware watchdog generates it.

Case 1: $\overline{DTACK}$ received, with or without $\overline{BERR}$.

STATE 5 During state 5 (S5), no bus signals are altered.

STATE 6 Sometime between state 2 (S2) and state 6 (S6), data from the device is driven onto the data bus.

STATE 7 On the falling edge of the clock entering state 7 (S7), the processor latches data from the addressed device and negates $\overline{AS}$ and $\overline{UDS}$, $\overline{LDS}$. The device negates $\overline{DTACK}$ at this time.

Case 2: $\overline{BERR}$ received without $\overline{DTACK}$.

STATE 5 During state 5 (S5), no bus signals are altered.

STATE 6 During state 6 (S6), no bus signals are altered.

STATE 7 During state 7 (S7), no bus signals are altered.

STATE 8 During state 8 (S8), no bus signals are altered.

STATE 9 $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$ negated. Slave negates $\overline{BERR}$.

3.1.3 16-Bit M68000 Bus Write Cycle

During a write cycle, the processor sends bytes of data to the memory or peripheral device. If the instruction specifies a word or long-word operation, the processor issues both $\overline{UDS}$ and $\overline{LDS}$ and writes both bytes. A long-word write is accomplished by two consecutive word writes. When the instruction specifies a byte operation, the processor uses the internal A0 bit to determine which byte to write and issues the appropriate data strobe. When the A0 bit equals zero, $\overline{UDS}$ is asserted; when the A0 bit equals one, $\overline{LDS}$ is asserted. The word write cycle flowchart is shown in Figure 3-5. The byte write cycle flowchart is shown in Figure 3-1. The word and byte write cycle timing is shown in Figure 3-7.

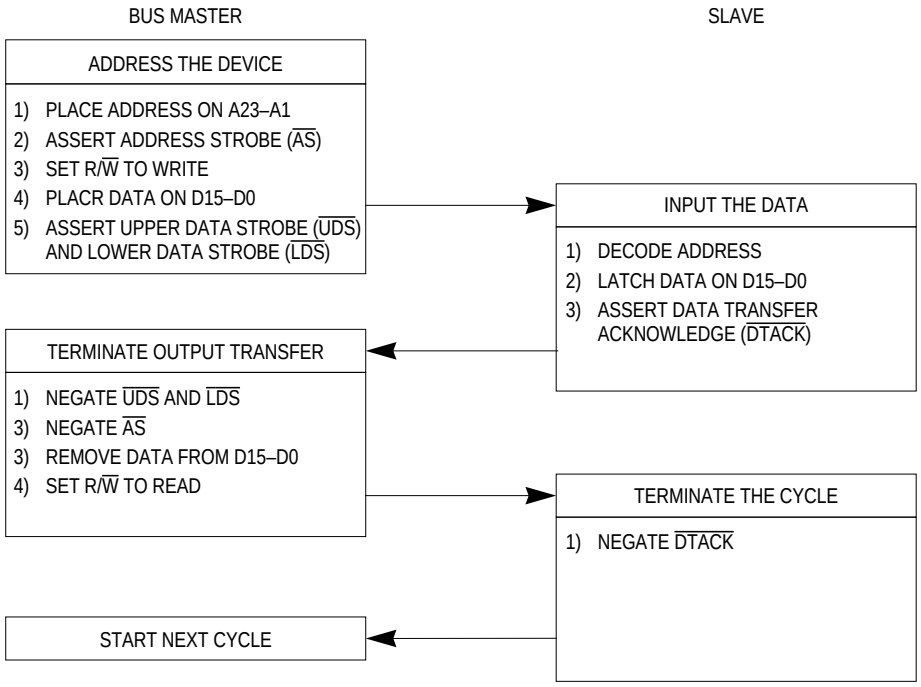


Figure 3-5. Word Write Cycle Flowchart (16-Bit Bus)

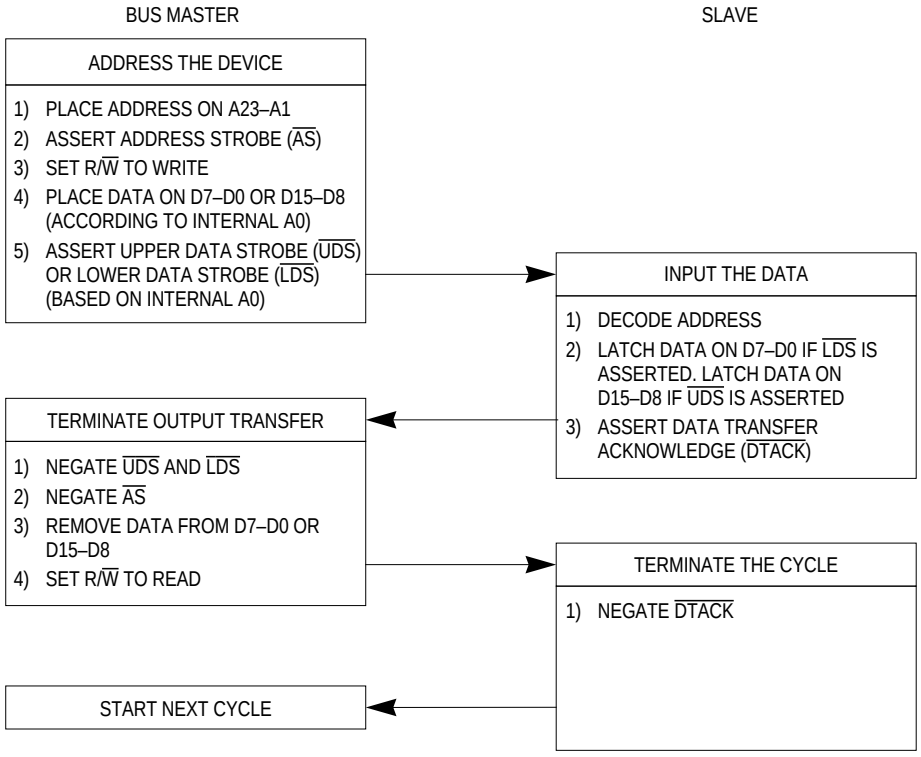


Figure 3-6. Byte Write Cycle Flowchart (16-Bit Bus)

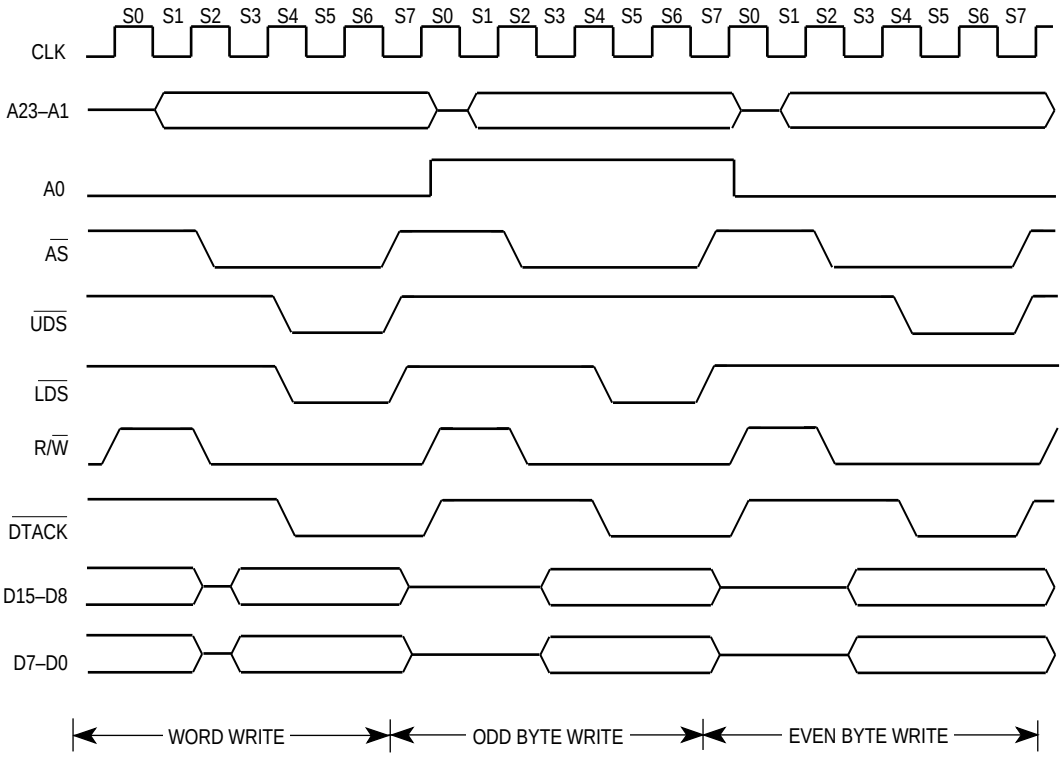


Figure 3-7. Word and Byte Write Cycle Timing Diagram

The descriptions of the eight states of a write cycle are as follows:

STATE 0 The write cycle starts in S0. The processor places a valid address on the address bus, and drives R/$\overline{W}$ high (if a preceding write cycle has left R/$\overline{W}$ low).

STATE 1 During S1, no bus signals are altered.

STATE 2 On the rising edge of S2, the processor asserts $\overline{AS}$ and drives R/$\overline{W}$ low.

STATE 3 During S3, the data bus is driven out of the high-impedance state as the data to be written is placed on the bus.

STATE 4 At the rising edge of S4, the processor asserts $\overline{UDS}$ and/or $\overline{LDS}$;. The processor waits for a cycle termination signal ($\overline{DTACK}$ or $\overline{BERR}$). If neither termination signal is asserted before the falling edge at the end of S4, the processor inserts wait states (full clock cycles) until either $\overline{DTACK}$ or $\overline{BERR}$ is asserted. Note that the $\overline{BERR}$ signal does not appear on an external pin on the MC68307; the internal hardware watchdog generates it.

Case 1: $\overline{\text{DTACK}}$ received, with or without $\overline{\text{BERR}}$.

STATE 5 During S5, no bus signals are altered.

STATE 6 During S6, no bus signals are altered.

STATE 7 On the falling edge of the clock entering S7, the processor negates $\overline{AS}$, $\overline{UDS}$, and/or $\overline{LDS}$. As the clock rises at the end of S7, the processor places the data bus in the high-impedance state, and drives R/$\overline{W}$ high. The device negates $\overline{DTACK}$ or $\overline{BERR}$ at this time.

Case 2: $\overline{\text{BERR}}$ received without $\overline{\text{DTACK}}$.

STATE 5 During state 5 (S5), no bus signals are altered.

STATE 6 During state 6 (S6), no bus signals are altered.

STATE 7 During state 7 (S7), no bus signals are altered.

STATE 8 During state 8 (S8), no bus signals are altered.

STATE 9 $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$ negated. Slave negates $\overline{BERR}$. At the end of S9, three-state data and drive R/$\overline{W}$ high.

3.1.4 Read-Modify-Write Cycle

The read-modify-write cycle performs a read operation, modifies the data in the arithmetic logic unit, and writes the data back to the same address. The address strobe ($\overline{AS}$) remains asserted throughout the entire cycle, making the cycle indivisible. The test and set (TAS) instruction uses this cycle to provide a signaling capability without deadlock between processors in a multiprocessing environment. The TAS instruction (the only instruction that uses the read-modify-write cycle) only operates on bytes. Thus, all read-modify-write cycles are byte operations. The read-modify-write flowchart is shown in Figure 3-8 and the timing diagram is shown in Figure 3-9.

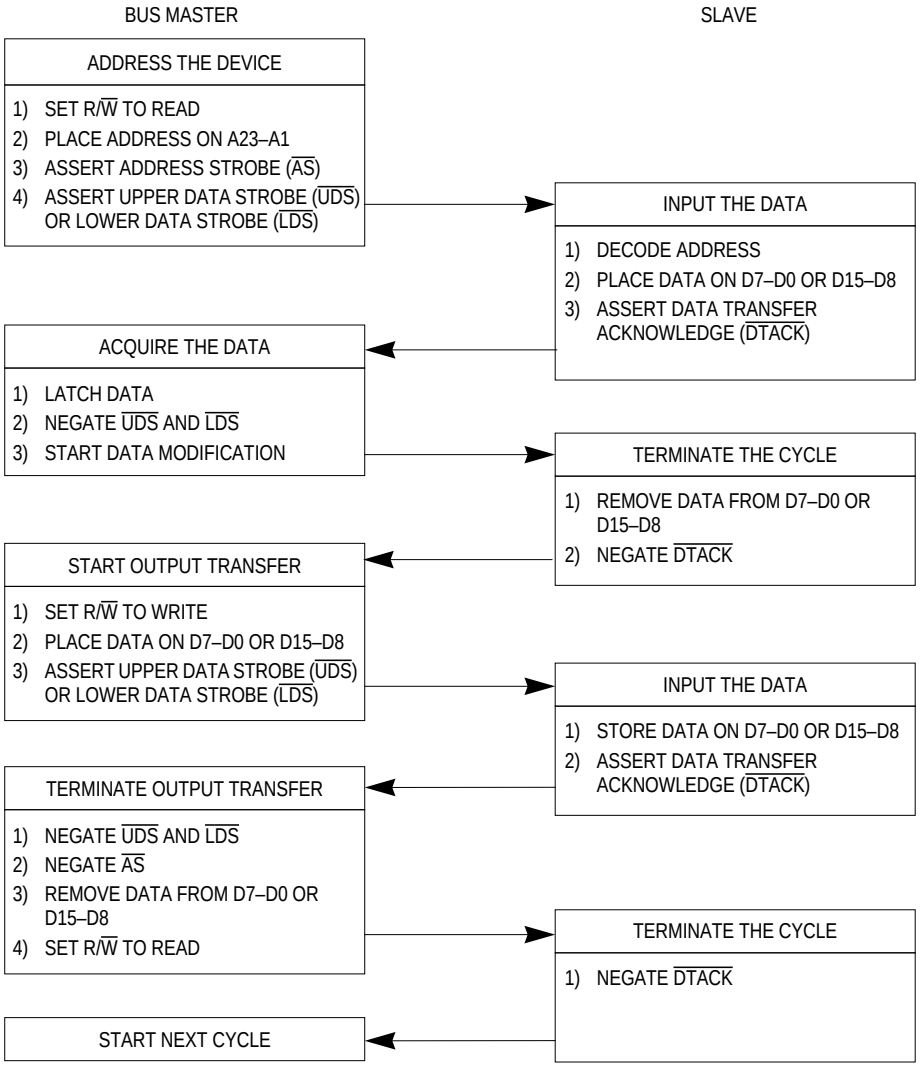


Figure 3-8. Read-Modify-Write Cycle Flowchart

The descriptions of the read-modify-write cycle states are as follows:

STATE 0 The read cycle starts in S0. The processor places a valid address on the address bus, and drives R/$\overline{W}$ high to identify a read cycle.

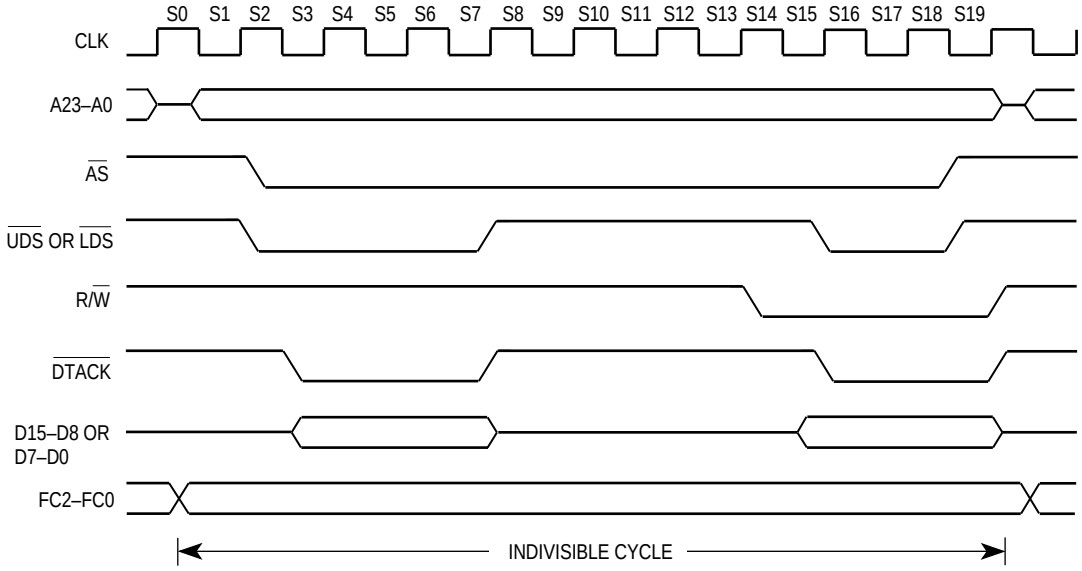


Figure 3-9. Read-Modify-Write Cycle Timing Diagram

STATE 1 During S1, no bus signals are altered.

STATE 2 On the rising edge of S2, the processor asserts $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$.

STATE 3 During S3, no bus signals are altered.

STATE 4 During S4, the processor waits for a cycle termination signal ($\overline{DTACK}$ or $\overline{BERR}$). If neither termination signal is asserted before the falling edge at the end of S4, the processor inserts wait states (full clock cycles) until either $\overline{DTACK}$ or $\overline{BERR}$ is asserted.

Case R1: $\overline{DTACK}$ only.

STATE 5 During S5, no bus signals are altered.

STATE 6 During S6, data from the device are driven onto the data bus.

STATE 7 On the falling edge of the clock entering S7, the processor accepts data from the device and negates $\overline{UDS}$/$\overline{LDS}$. The device negates $\overline{DTACK}$ or $\overline{BERR}$ at this time.

STATE 8–11 The bus signals are unaltered during S8–S11, during which the arithmetic logic unit makes appropriate modifications to the data.

STATE 12 The write portion of the cycle starts in S12. The address bus lines, $\overline{AS}$ and R/$\overline{W}$ remain unaltered.

STATE 13 During S13, no bus signals are altered.

STATE 14 On the rising edge of S14, the processor drives R/$\overline{W}$ low.

STATE 15 During S15, the data bus is driven out of the high-impedance state as the data to be written are placed on the bus.

STATE 16 At the rising edge of S16, the processor asserts $\overline{UDS}$/$\overline{LDS}$. The processor waits for $\overline{DTACK}$ or $\overline{BERR}$. If neither termination signal is asserted before the falling edge at the close of S16, the processor inserts wait states (full clock cycles) until either $\overline{DTACK}$ or $\overline{BERR}$ is asserted.

Case W1: $\overline{DTACK}$ with or without $\overline{BERR}$.

STATE 17 During S17, no bus signals are altered.

STATE 18 During S18, no bus signals are altered.

STATE 19 On the falling edge of the clock entering S19, the processor negates $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$. As the clock rises at the end of S19, the processor places the data bus in the high-impedance state, and drives R/$\overline{W}$ high. The device negates $\overline{DTACK}$ or $\overline{BERR}$ at this time.

Case R2: $\overline{DTACK}$ and $\overline{BERR}$ on read.

STATE 5 During S5, no bus signals are altered.

STATE 6 During S6, no bus signals are altered, and data from the device is ignored.

STATE 7 $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$ are negated. The cycle terminates without the write portion.

Case R3: $\overline{BERR}$ only on read.

STATE 5 During S5, no bus signals are altered.

STATE 6 During S6, no bus signals are altered.

STATE 7 During S7, no bus signals are altered.

STATE 8 During S8, no bus signals are altered.

STATE 9 $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$ are negated. The cycle terminates without the write portion.

Case W2: $\overline{BERR}$ only on write.

STATE 17 During S17, no bus signals are altered.

STATE 18 During S18, no bus signals are altered.

STATE 19 During S19, no bus signals are altered.

STATE 20 During S20, no bus signals are altered.

STATE 21 The processor negates $\overline{AS}$ and $\overline{UDS}$/$\overline{LDS}$.

3.1.5 CPU Space Cycle

A CPU space cycle, indicated when the internal function codes are all high, is a special processor cycle. In the EC000 core, CPU space is used only for interrupt acknowledge cycles. Figure 3-10 shows the encoding of an interrupt acknowledge cycle. No response is expected or allowed from external devices. On the MC68307 this cycle is an indication of the internal interrupt controller's vector response.

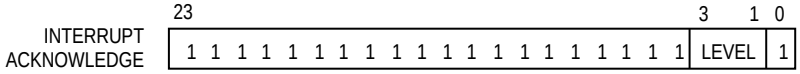


Figure 3-10. Interrupt Acknowledge Cycle – Address Bus

As the MC68307 implementation does not provide function code pins to differentiate CPU space cycles (the internal function code signals are all high in this type of cycle), care is required when decoding addresses which may match the above address. No problems will be encountered as long as the chip selects are used as a term in the decoding logic. This will ensure that CPU space cycles are not decoded. The interrupt acknowledge cycle places the level of the interrupt being acknowledged on address bits A3–A1 and drives all other address lines high. The interrupt acknowledge cycle reads a vector number when the MC68307 interrupt controller places a vector number on the data bus.

The timing diagram for an interrupt acknowledge cycle is shown in Figure 3-11.

3.1.6 8-Bit M68000 Dynamically-Sized Bus

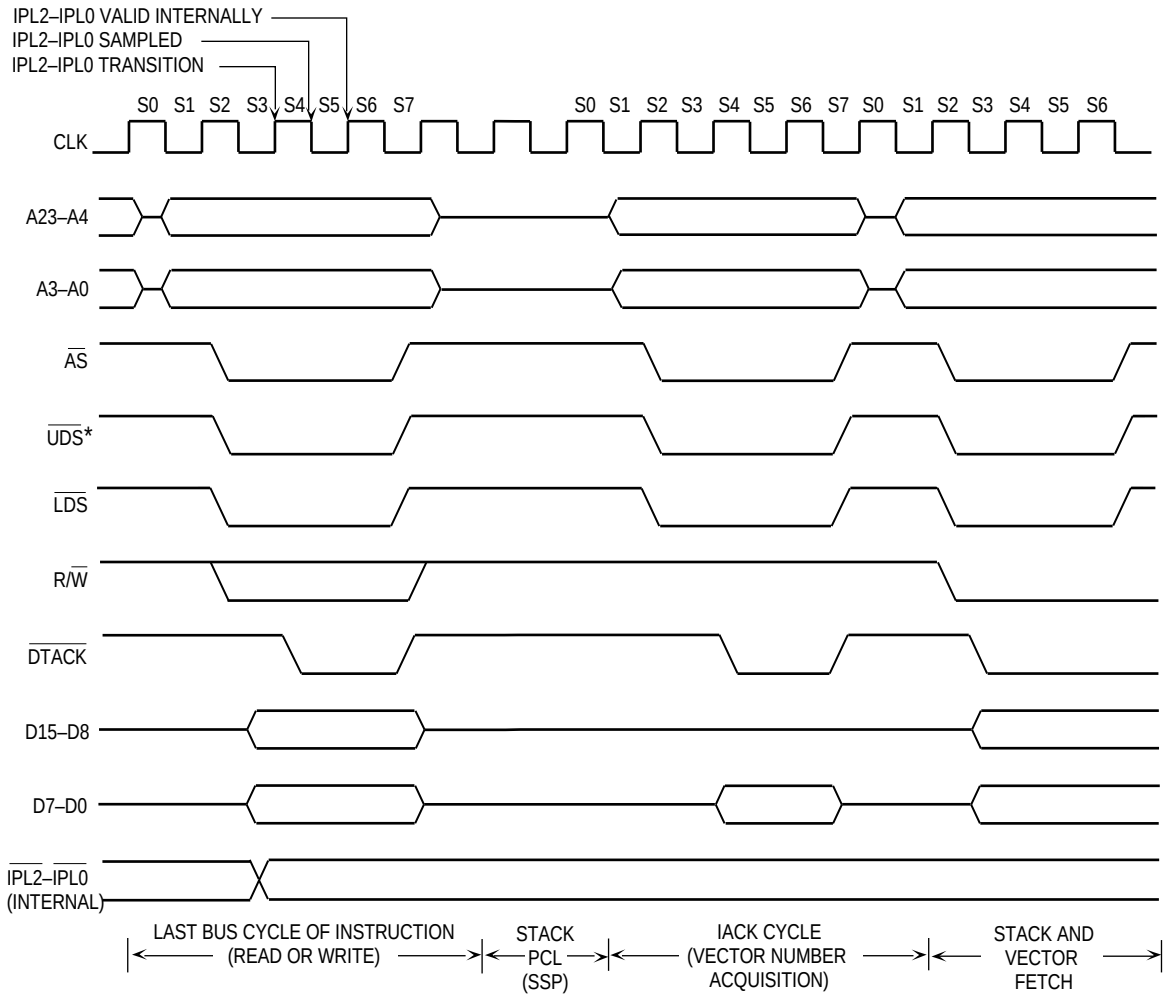
M68000 8-bit bus cycles appear when the MC68307 dynamic bus sizing is enabled. This bus sizing adds cycle-by-cycle control of data bus width to the EC000 core bus, as for the MC68020, but with the difference that the bus width is controlled via the chip selects, rather than external $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$ inputs.

This provides the flexibility of differing bus widths for RAM, ROM and peripherals, without the pin overhead of the MC68020 solution ($\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$, $\overline{\text{SIZ1}}$ and $\overline{\text{SIZ0}}$).

Each of the four programmable chip selects has a default bus width of 8- or 16-bits associated with it. The initial bus width of $\overline{\text{CS0}}$ is set upon reset by the state of the BUSW external pin (0 for an 8-bit data bus, 1 for a 16-bit data bus).

The bus widths for $\overline{\text{CS2}}$, $\overline{\text{CS3}}$ and $\overline{\text{CS4}}$ should be programmed during system initialization using the BUSWx bits in the system configuration register (SCR), according to the system design. The default after reset is 16-bits wide.

All bus accesses not matched by any of the chip selects or the internal peripheral address ranges require an external $\overline{\text{DTACK}}$ input to terminate the cycle. The data bus width of such cycles is 16-bits by default; the EBUSW bit in the system configuration register (SCR) can be cleared to specify external $\overline{\text{DTACK}}$ cycles as 8-bit data bus.



*Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D15–D8 at this time.

Figure 3-11. Interrupt Acknowledge Cycle Timing Diagram

When using the M68000 8-bit bus, transfer of the data between the MC68307 and other devices on the bus involves the following signals:

- Address Bus A23–A0
- Data Bus D15–D0
- Control Signals ($\overline{AS}$, $\overline{UDS}$, R/$\overline{W}$, $\overline{DTACK}$). $\overline{LDS}$ is not used.

All M68000 8-bit bus cycles use the upper half of the data bus (D15–D8) for reads and writes. Note that this differs from the static 8-bit bus provided by the MC68HC001, MC68008 and MC68302, which use the lower half (D7–D0).

Therefore, only $\overline{UDS}$ is asserted during such cycles, never $\overline{LDS}$. The chip select logic can control the data strobes in this way only because it determines the bus width that is programmed for a particular chip-select before $\overline{DTACK}$ is asserted.

3.1.7 8051-Bus Operation

The 8051-compatible bus is a multiplexed address/data bus scheme which provides an 8-bit-wide data interface to memories and peripherals. It is typically used for ASIC devices where pin count minimization is important.

Individual read and write strobes are provided, along with a latch enable signal which indicates that the multiplexed portion of the bus is carrying a valid address which is latched by the memory or peripheral device at the beginning of the cycle.

When using the 8051-compatible 8-bit bus, transfer of data between the MC68307 and other devices on the bus involves the following signals:

- Address/data multiplexed bus AD7–AD0
- Optionally higher-order address bus A23–A8 or part thereof
- Control signals ($\overline{CS3}$, ALE, $\overline{RD}$, $\overline{WR}$)

If the 8051-compatible interface is enabled, one of the MC68307 programmable chip selects ($\overline{CS3}$) is used to indicate the address range to be decoded as 8051 accesses. The base address and address mask should be programmed as described in **Section 5.1.2.3 8051-Compatible Bus Chip Select**.

The device being addressed can decode address lines A8 and higher if an addressing range larger than 256-bytes is required, up to the maximum address range of the MC68307.

During 8051-compatible bus cycles, the M68000 strobe outputs are still asserted, indicating the “underlying” M68000 bus-cycle, and the D15–D0 outputs reflect the internal data-bus value being presented to the EC000 processor core.

Figure 3-12 and Figure 3-13 show examples of 8051-compatible read and write cycle timing diagrams, respecitively.

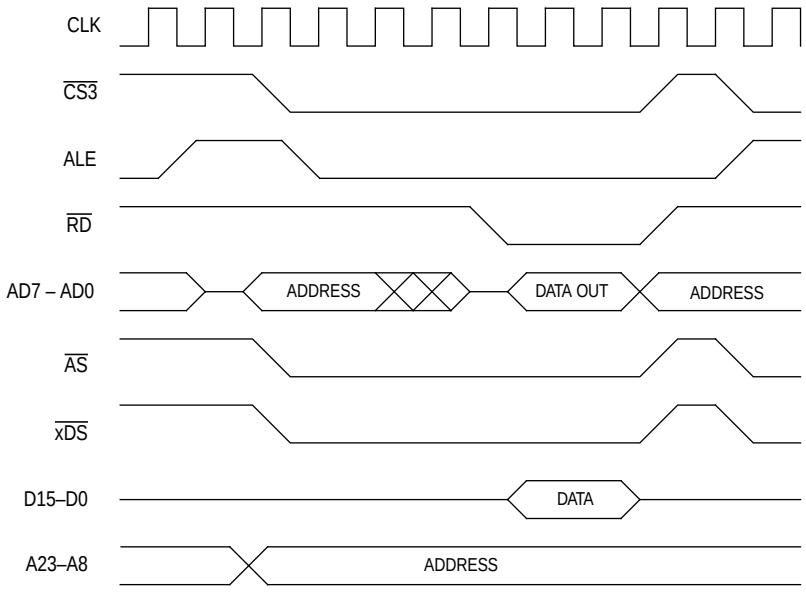


Figure 3-12. 8051-Compatible Read Cycle Signals

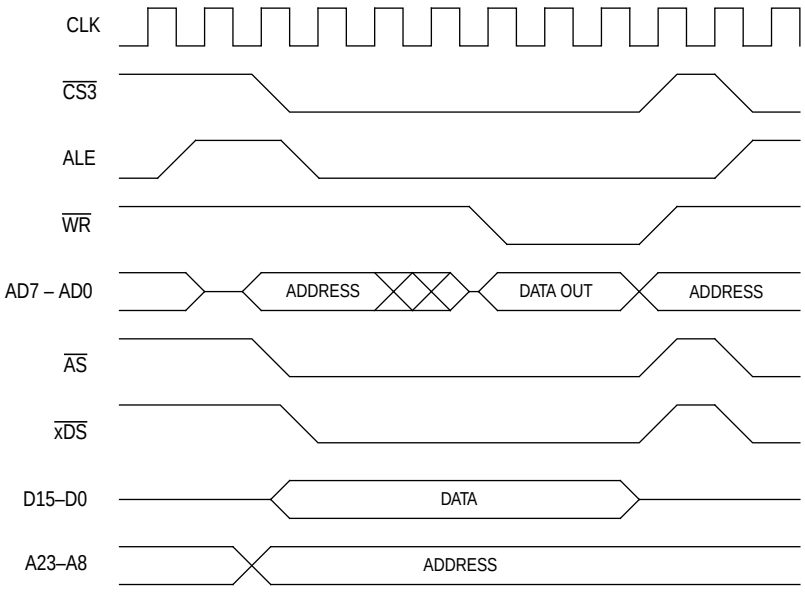


Figure 3-13. 8051-Compatible Write Cycle Signals

3.2 BUS ARBITRATION

Bus arbitration is a technique used by bus master devices to request, to be granted, and to acknowledge bus mastership. Bus arbitration consists of the following:

1. Asserting a bus mastership request
2. Receiving a grant indicating that the bus is available at the end of the current cycle
3. Acknowledging that mastership has been assumed

Figure 3-14 is a flowchart showing the bus arbitration cycle of the EC000 core. Figure 3-16 is a timing diagram of the bus arbitration cycle charted in Figure 3-14. This technique allows processing of bus requests during data transfer cycles.

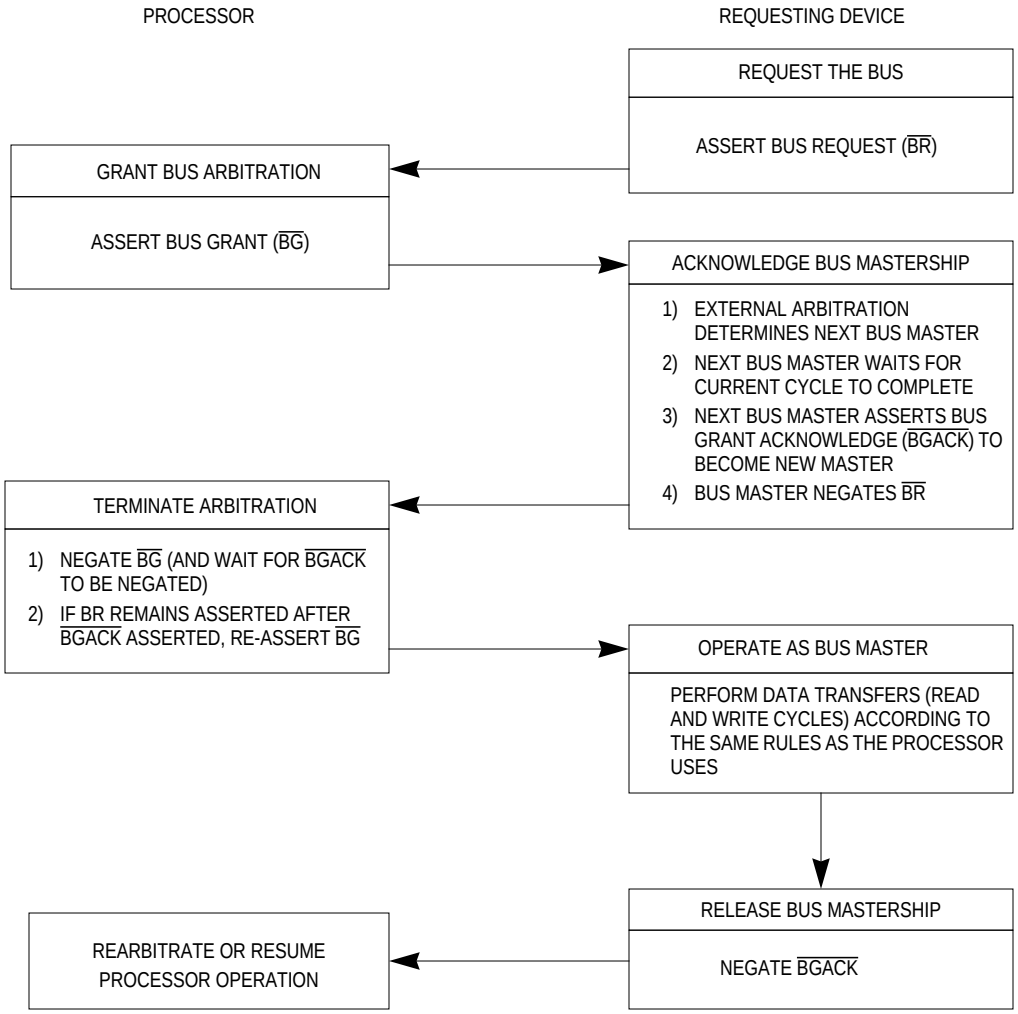


Figure 3-14. Three-Wire Bus Arbitration Cycle Flowchart

There are two ways to arbitrate the bus; three-wire and two-wire bus arbitration. The EC000 core can do either two-wire or three-wire bus arbitration. Figure 3-14 and Figure 3-16 show

three-wire bus arbitration and Figure 3-15 and Figure 3-17 show two-wire bus arbitration. $\overline{\text{BGACK}}$ must be pulled high for two-wire bus arbitration.

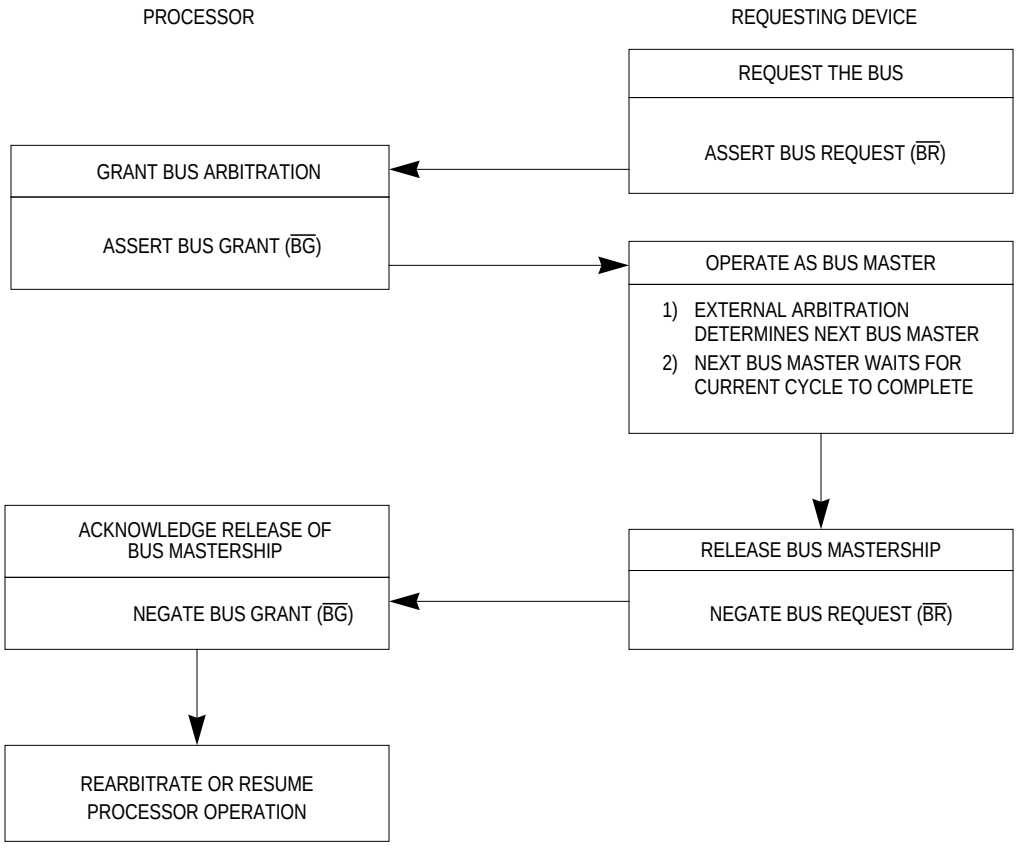


Figure 3-15. Two-Wire Bus Arbitration Cycle Flowchart

The timing diagram in Figure 3-16 shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation applies to a system consisting of a processor and one other device capable of becoming bus master. In systems having several devices that can be bus masters, bus request lines from these devices can be wire-ORed at the processor, and more than one bus request signal could occur.

The bus grant signal is negated a few clock cycles after the assertion of the bus grant acknowledge signal. However, if bus requests are pending, the processor reasserts bus grant for another request a few clock cycles after bus grant (for the previous request) is negated. In response to this additional assertion of bus grant, external arbitration circuitry selects the next bus master before the current bus master has completed the bus activity.

The timing diagram in Figure 3-17 also applies to a system consisting of a processor and one other device capable of becoming bus master. Since the two-wire bus arbitration scheme does not use a bus grant acknowledge signal, the external master must continue to assert $\overline{\text{BR}}$ until it has completed its bus activity. The processor negates $\overline{\text{BG}}$ when $\overline{\text{BR}}$ is negated.

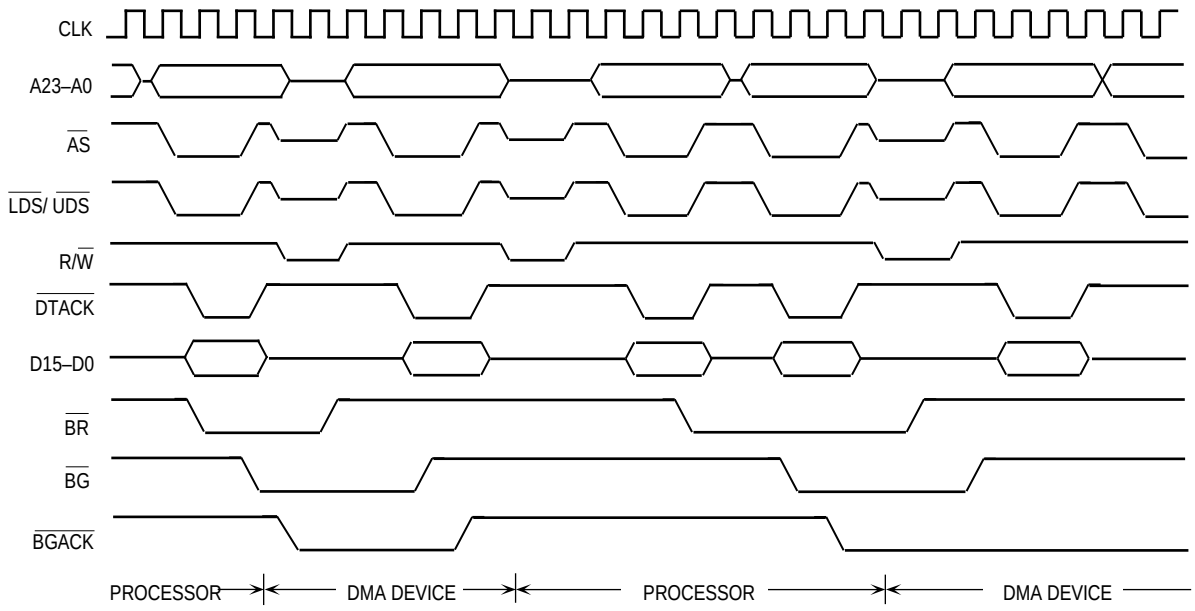


Figure 3-16. Three-Wire Bus Arbitration Timing Diagram

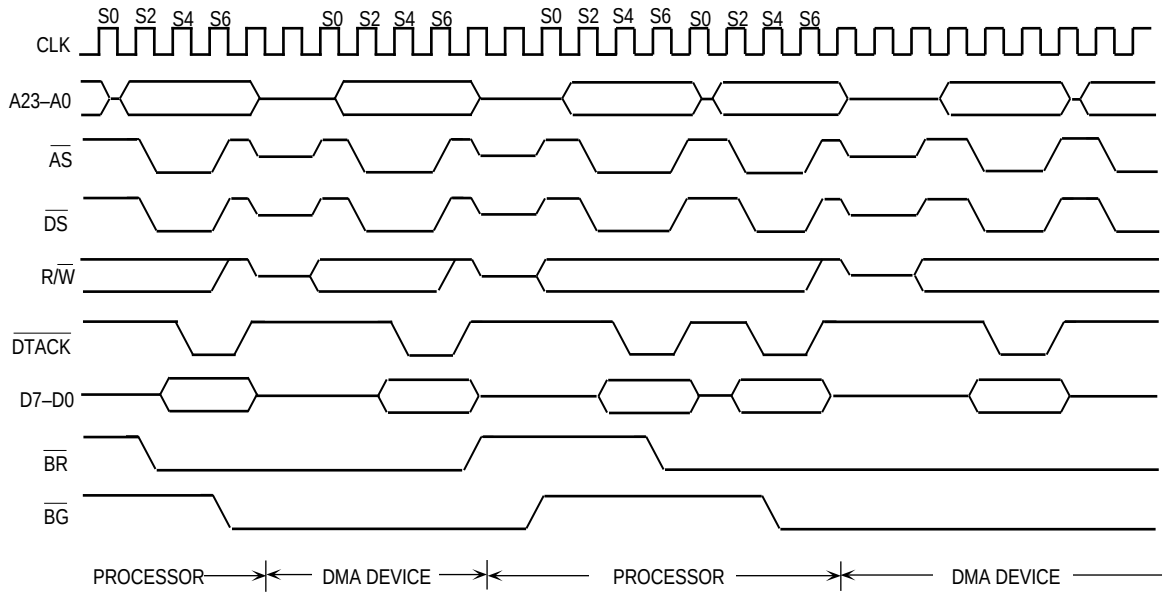


Figure 3-17. Two-Wire Bus Arbitration Timing Diagram

3.2.1 Requesting the Bus

External devices capable of becoming bus masters assert $\overline{BR}$ to request the bus. This signal can be wire-ORed (not necessarily constructed from open-collector devices) from any of the devices in the system that can become bus master. The processor, which is at a lower bus priority level than the external devices, relinquishes the bus after it completes the current bus cycle.

3.2.2 Receiving the Bus Grant

The processor asserts $\overline{BG}$ as soon as possible. Normally, this process immediately follows internal synchronization, except when the processor has made an internal decision to execute the next bus cycle but has not yet asserted $\overline{AS}$ for that cycle. In this case, $\overline{BG}$ is delayed until $\overline{AS}$ is asserted to indicate to external devices that a bus cycle is in progress. One such case is during a dynamically sized cycle; $\overline{BG}$ will not assert until the second half of the cycle.

$\overline{BG}$ can be routed through a daisy-chained network or through a specific priority-encoded network. Any method of external arbitration that observes the protocol can be used.

3.2.3 Acknowledgment of Mastership (Three-Wire Bus Arbitration Only)

Upon receiving $\overline{BG}$, the requesting device waits until $\overline{AS}$, $\overline{DTACK}$, and $\overline{BGACK}$ are negated before asserting $\overline{BGACK}$. The negation of $\overline{AS}$ indicates that the previous bus master has completed its cycle. (No device is allowed to assume bus mastership while $\overline{AS}$ is asserted.) The negation of $\overline{BGACK}$ indicates that the previous master has released the bus. The negation of $\overline{DTACK}$ indicates that the previous slave has terminated the connection to the previous master. (In some applications, $\overline{DTACK}$ might not be included in this function; general-purpose devices would be connected using $\overline{AS}$ only.) When $\overline{BGACK}$ is asserted, the asserting device is bus master until it negates $\overline{BGACK}$. $\overline{BGACK}$ should not be negated until after the bus cycle(s) is complete. A device relinquishes control of the bus by negating $\overline{BGACK}$.

The $\overline{BR}$ from the granted device should be negated after $\overline{BGACK}$ is asserted. If another bus request is pending, $\overline{BG}$ is reasserted within a few clocks, as described in **Section 3.3 Bus Arbitration Control**. The processor does not perform any external bus cycles before reasserting $\overline{BG}$.

3.3 BUS ARBITRATION CONTROL

All asynchronous bus arbitration signals to the processor are synchronized before being used internally. As shown in Figure 3-18, synchronization requires a maximum of one and a half cycles of the system clock. The input asynchronous signal is sampled on the falling edge of the clock and is valid internally after the next rising edge.

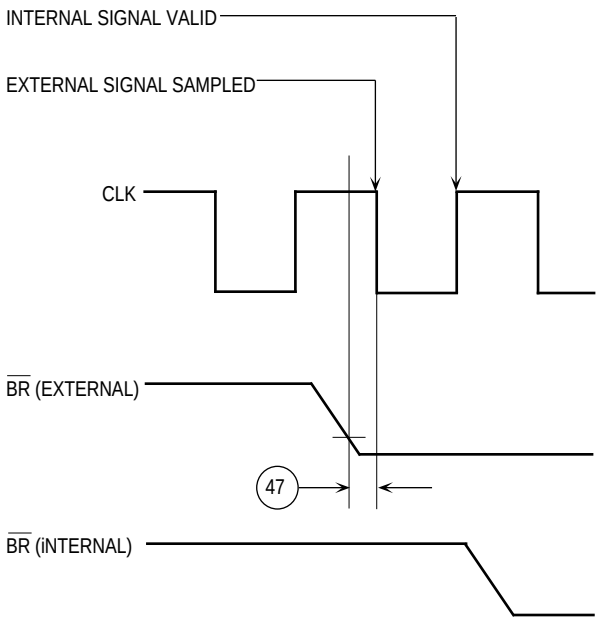


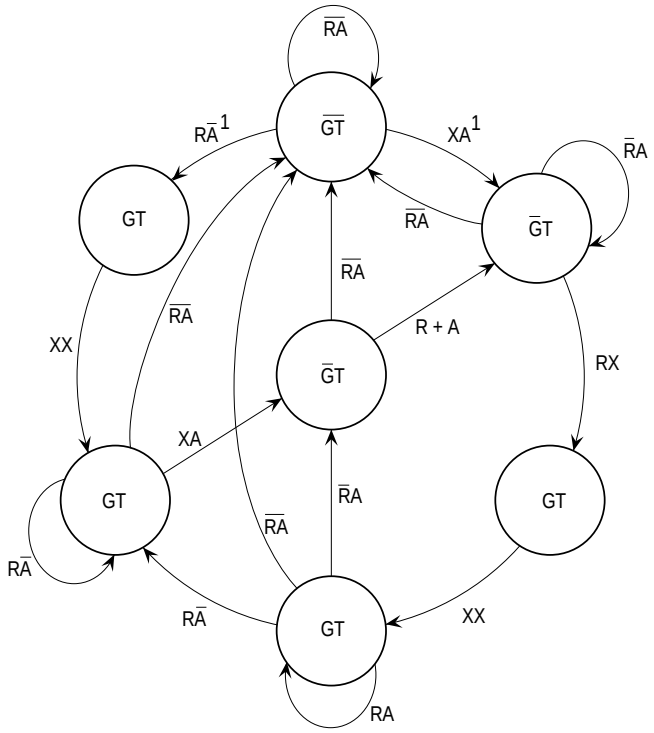
Figure 3-18. External Asynchronous Signal Synchronization

Bus arbitration control is implemented with a finite state machine (see Figure 3-19). In Figure 3-19, input signals R and A are the internally synchronized versions of $\overline{BR}$ and $\overline{BGACK}$. The $\overline{BG}$ output is shown as G, and the internal three-state control signal is shown as T. If T is true, the address, data, and control buses are placed in the high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high), regardless of their true active voltage level. State changes (valid outputs) occur on the next rising edge of the clock after the internal signal is valid.

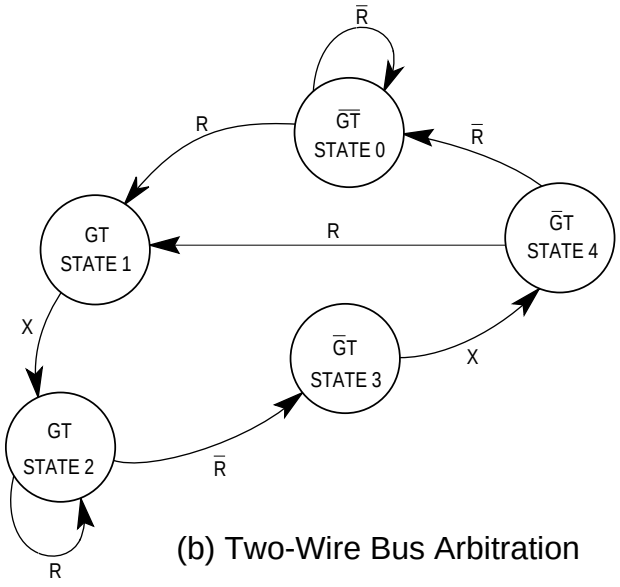
A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 3-20. The bus arbitration timing while the bus is inactive (e.g., the processor is performing internal operations for a multiply instruction) is shown in Figure 3-21.

When a bus request is made after the MPU has begun a bus cycle and before $\overline{AS}$ has been asserted (S0), the special sequence shown in Figure 3-22 applies. Instead of being asserted on the next rising edge of clock, $\overline{BG}$ is delayed until the second rising edge following its internal assertion.

Figure 3-20, Figure 3-21, and Figure 3-22 apply to processors using three-wire bus arbitration. Figure 3-23, Figure 3-24, and Figure 3-25 apply to processors using two-wire bus arbitration.



(a) Three-Wire Bus Arbitration



(b) Two-Wire Bus Arbitration

R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic
X = Don't Care

1. State machine will not change if the bus is S0 or S1.
2. The address bus will be placed in the high-impedance state if T is asserted and AS is negated.

Figure 3-19. Bus Arbitration Unit State Diagrams

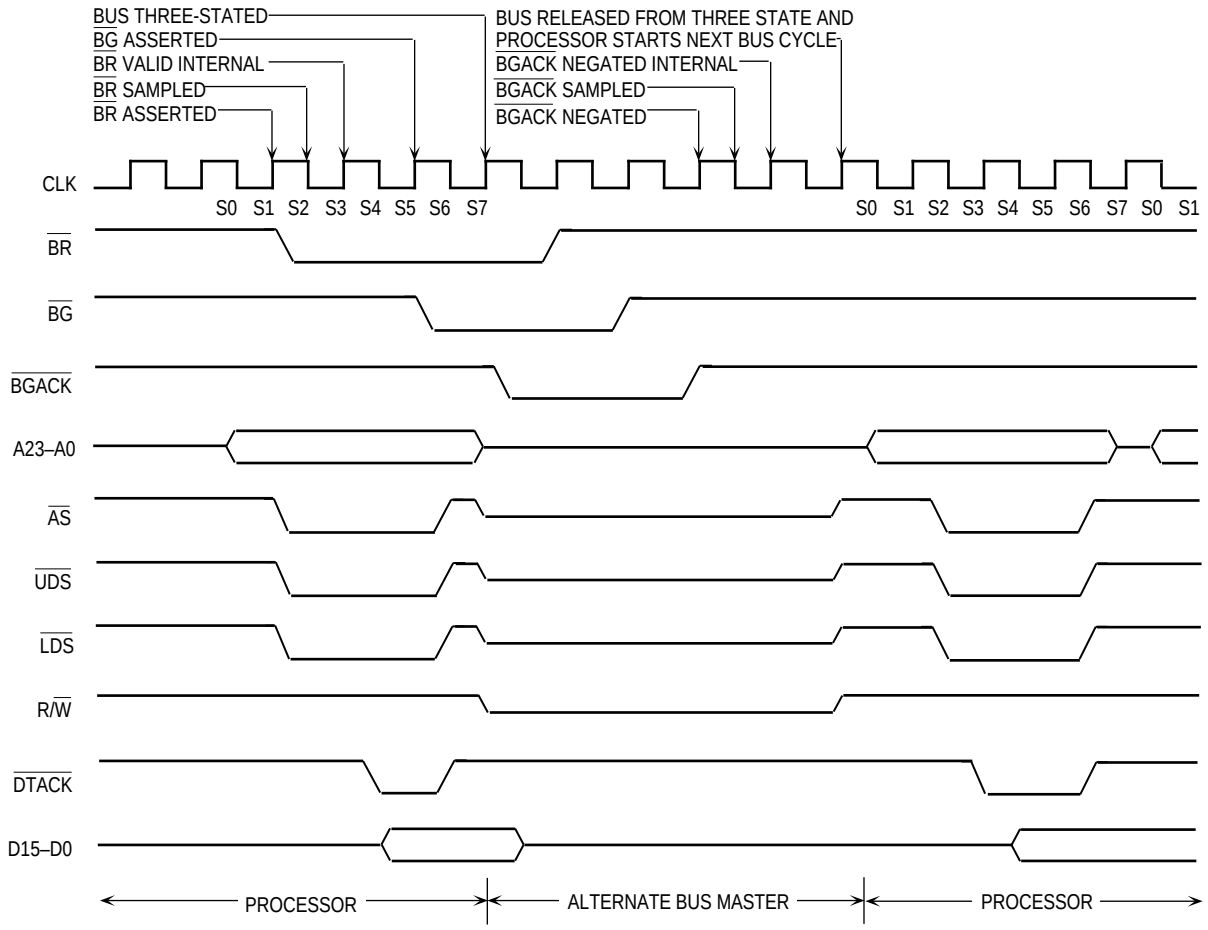


Figure 3-20. Three-Wire Bus Arbitration Timing Diagram—Processor Active

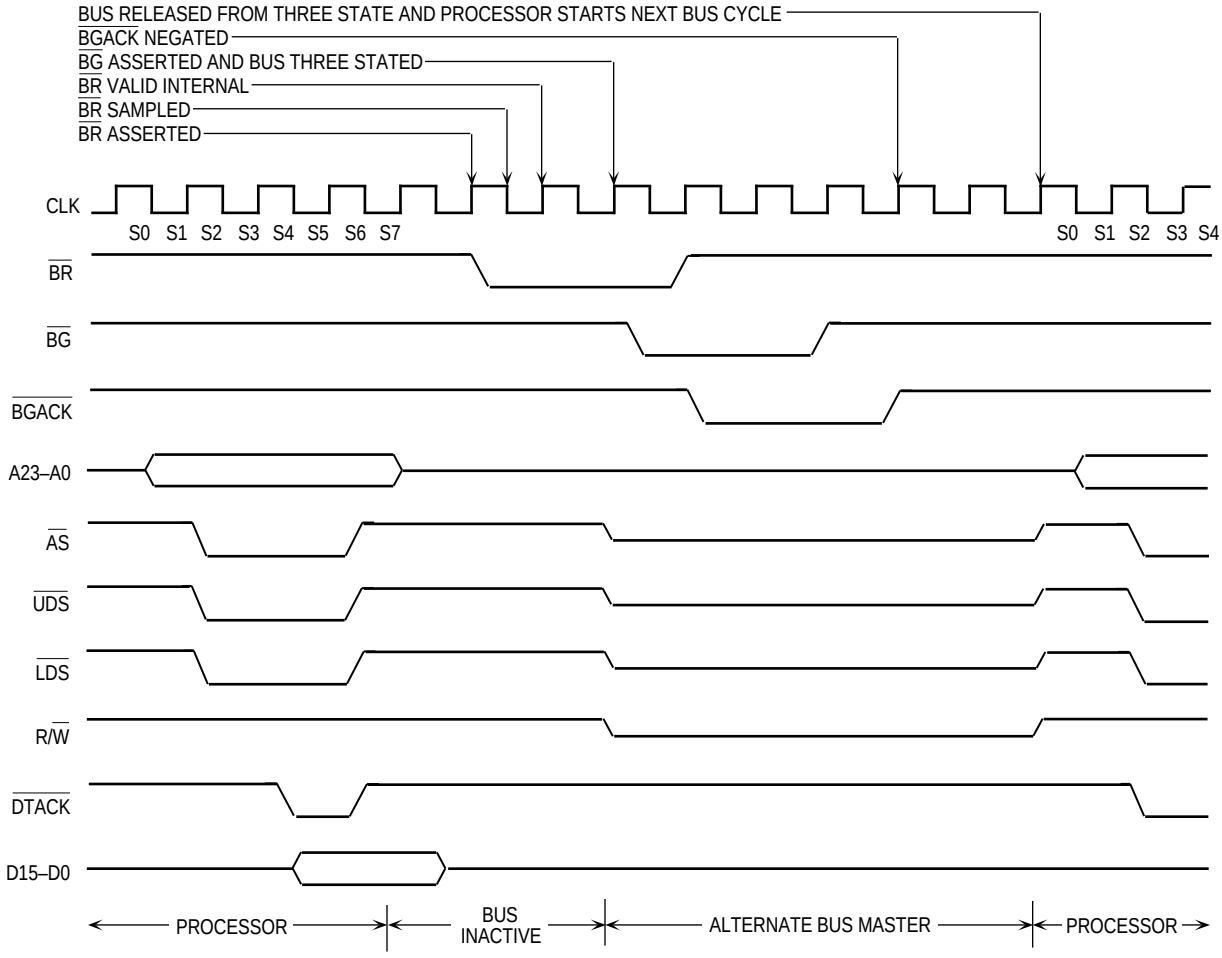


Figure 3-21. Three-Wire Bus Arbitration Timing Diagram—Bus Inactive

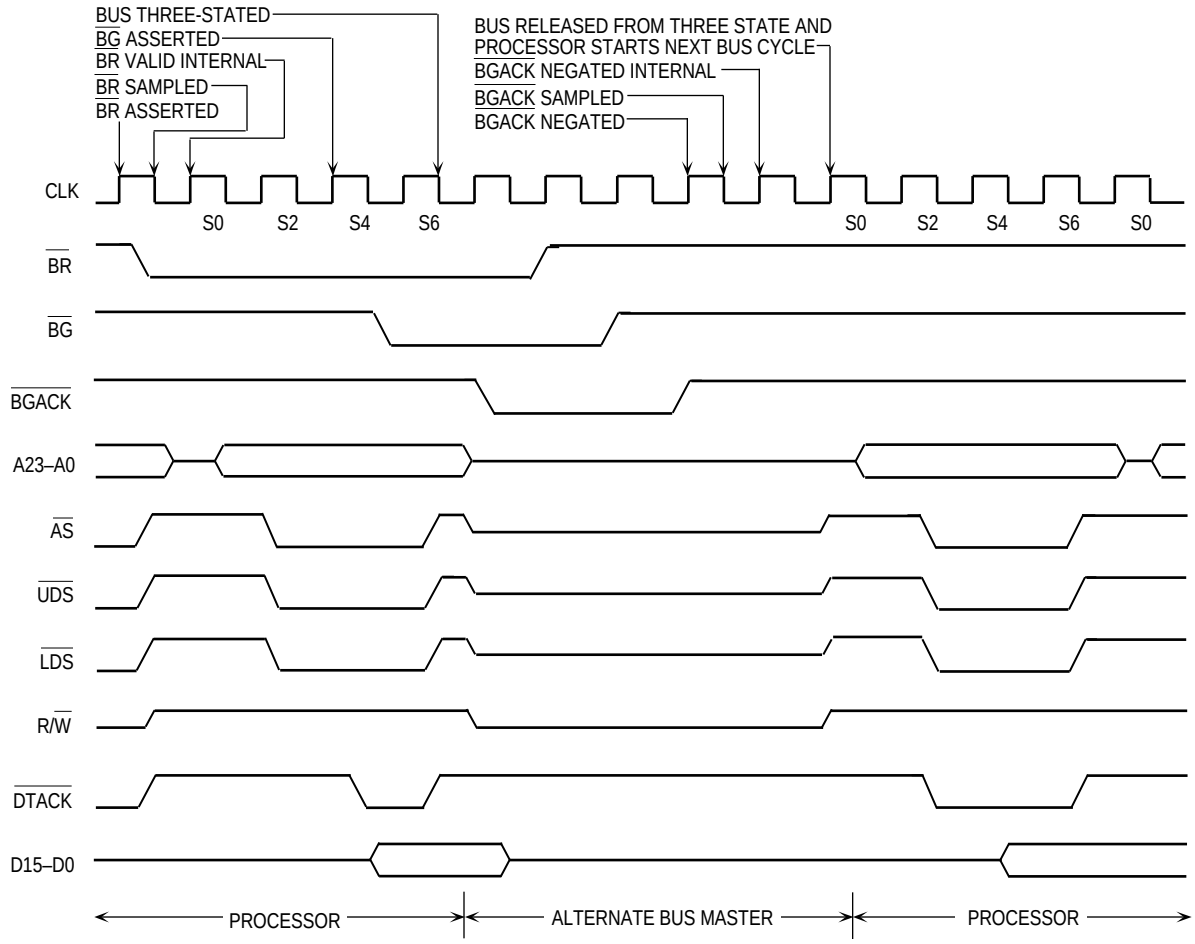


Figure 3-22. Three-Wire Bus Arbitration Timing Diagram—Special Case

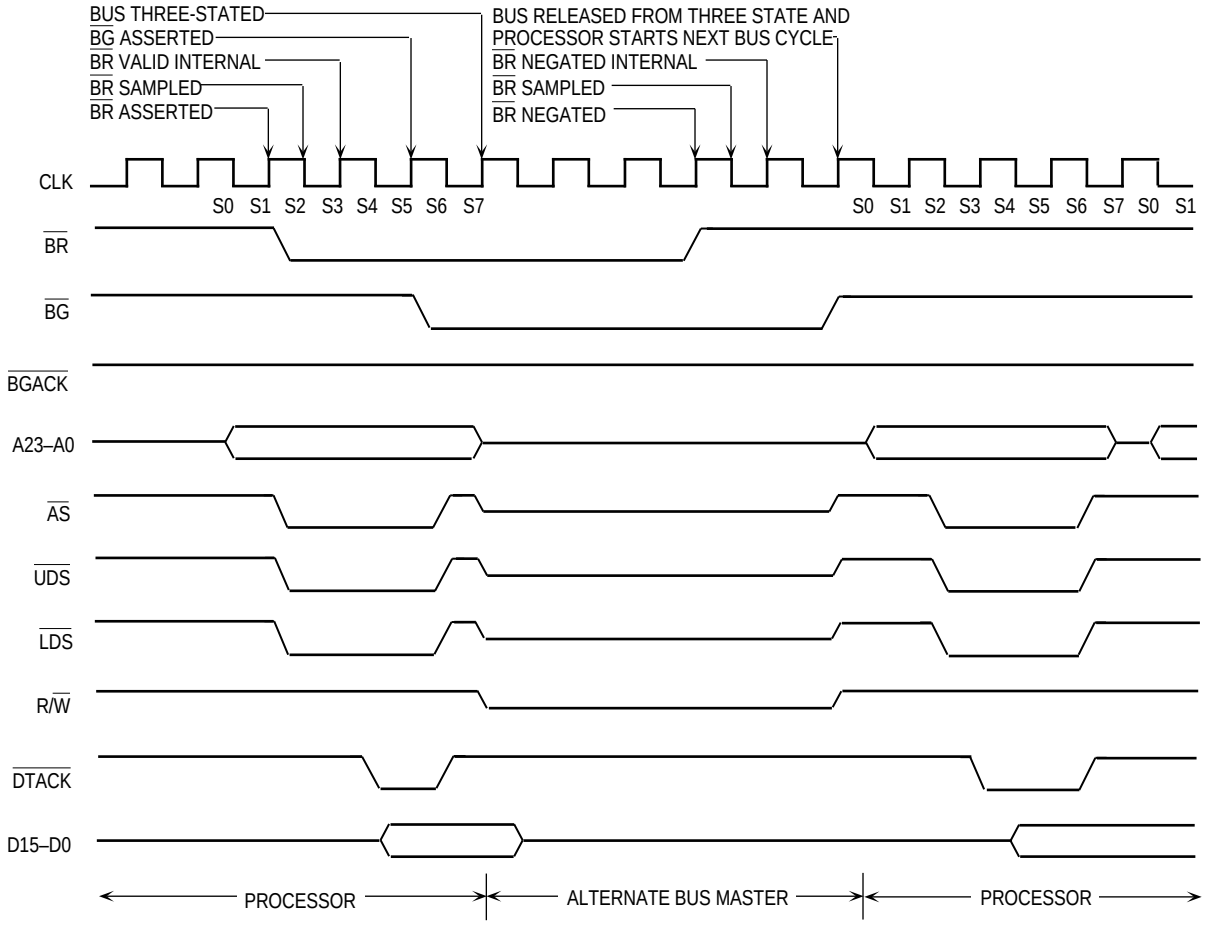


Figure 3-23. Two-Wire Bus Arbitration Timing Diagram—Processor Active

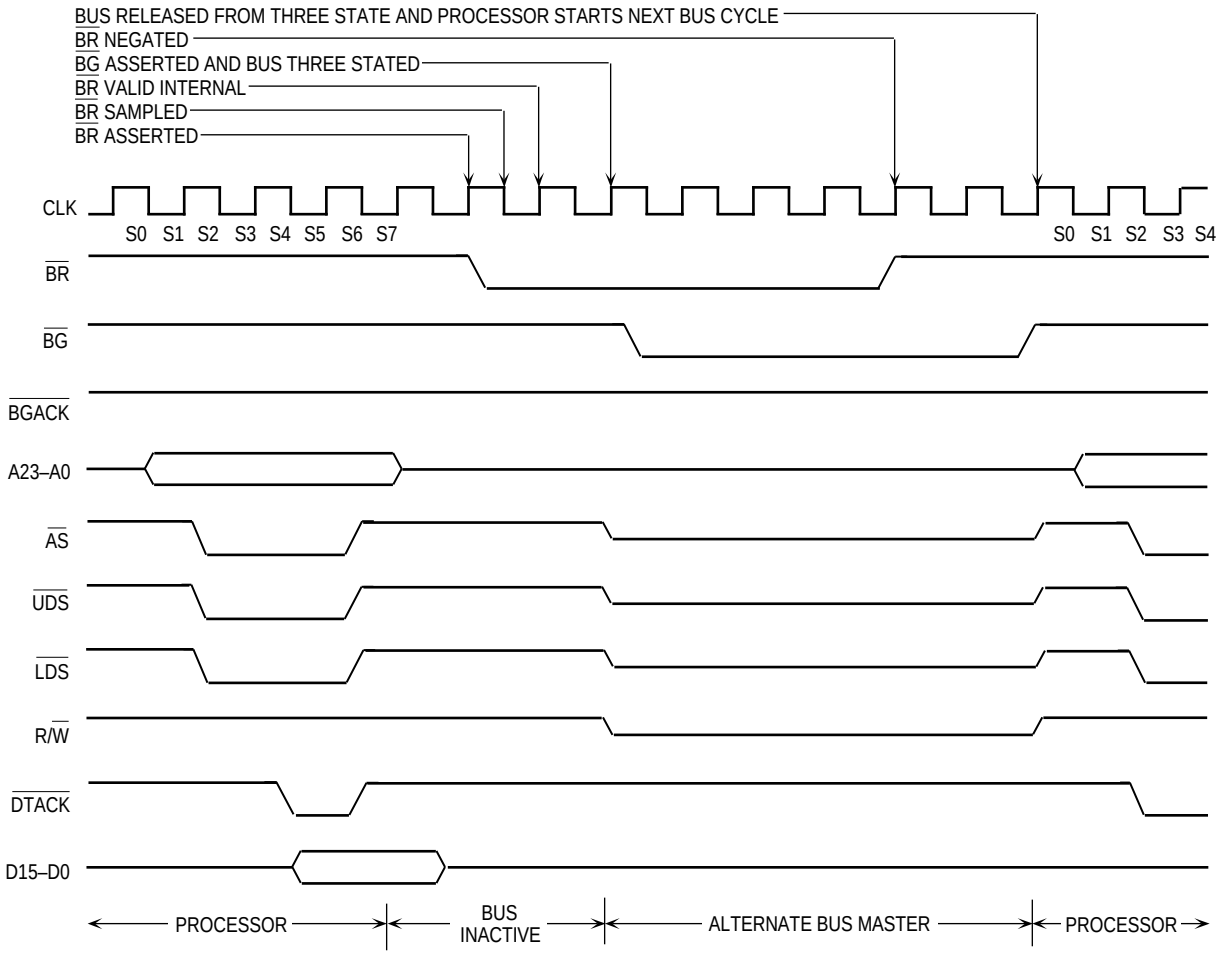


Figure 3-24. Two-Wire Bus Arbitration Timing Diagram—Bus Inactive

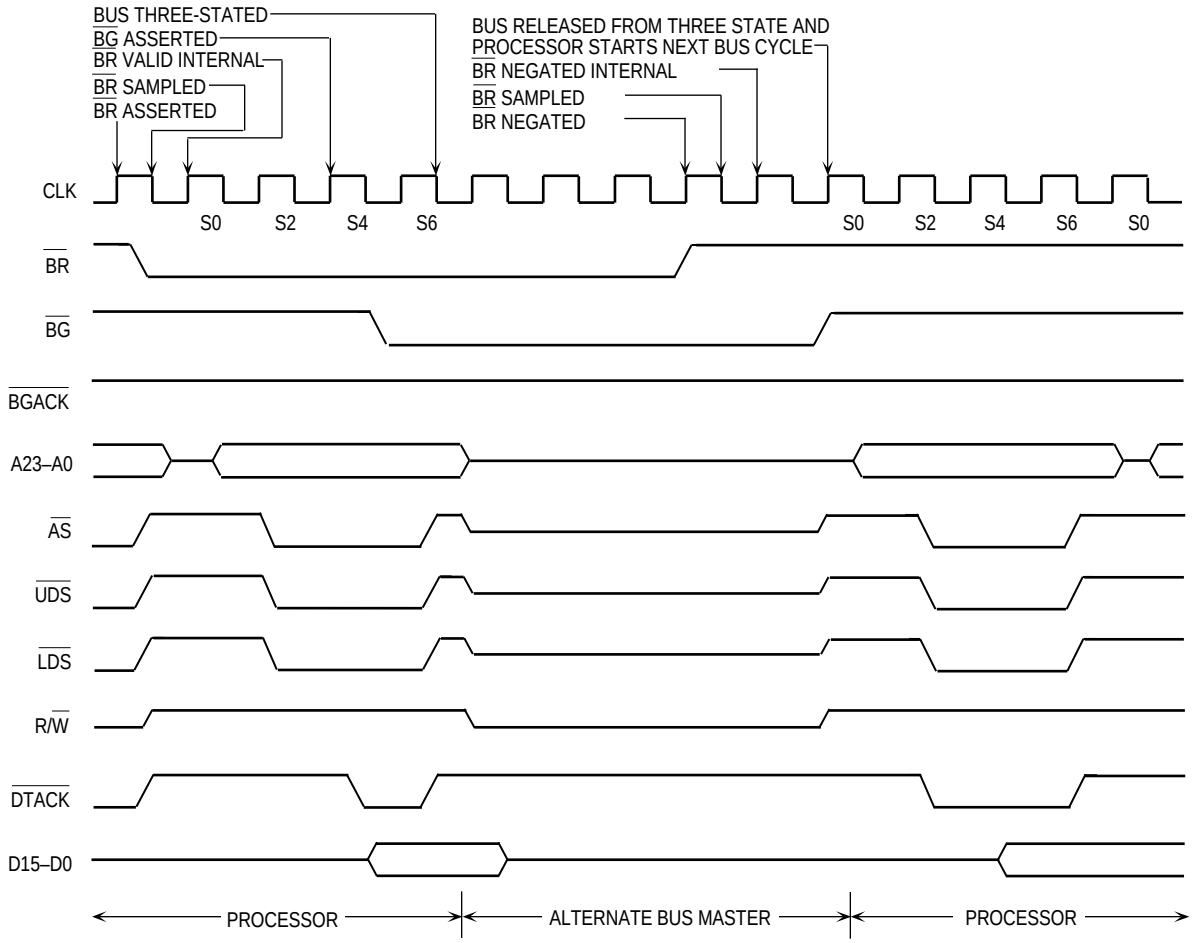


Figure 3-25. Two-Wire Bus Arbitration Timing Diagram—Special Case

3.4 BUS ERROR AND HALT OPERATION

In a bus architecture that requires a handshake from an external device, such as the asynchronous bus used in the M68000 family, the handshake may not always occur. A bus monitor is provided to terminate a bus cycle in error when the expected signal is not asserted. Different systems and different devices within the same system require different maximum-response times. This internal circuitry asserts the internal EC000 core bus error signal after the appropriate delay following the assertion of address strobe.

3.4.1 Bus Error Operation

When the bus error condition is recognized, the current bus cycle is terminated in S7 for a read cycle, a write cycle, or the read portion of a read-modify-write cycle. For the write portion of a read-modify-write cycle, the current bus cycle is terminated in S19.

After the aborted bus cycle is terminated, the processor enters exception processing for the bus error exception. During the exception processing sequence, the following information is placed on the supervisor stack:

1. Status register
2. Program counter (two words, which may be up to five words past the instruction being executed)
3. Error information

The first two items are identical to the information stacked by any other exception. The EC000 core stacks bus error information to help determine and to correct the error.

After the processor has placed the required information on the stack, the bus error exception vector is read from vector table entry 2 (offset $08) and placed in the program counter. The processor resumes execution at the address in the vector, which is the first instruction in the bus error handler routine. Refer to Figure 3-26 for an example bus error timing diagram.

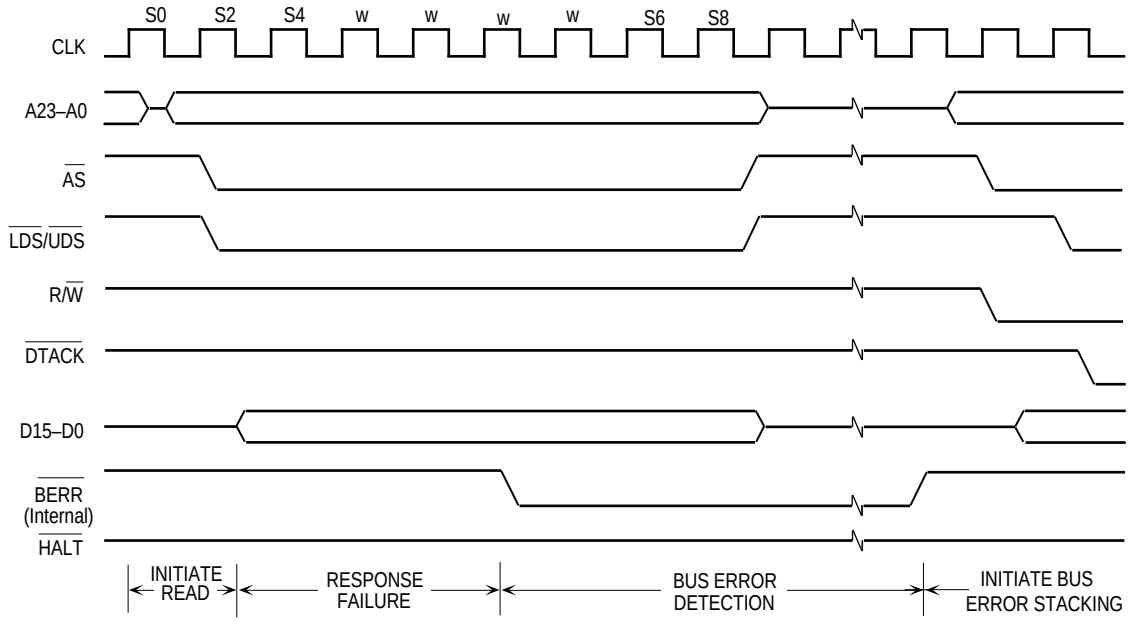


Figure 3-26. Bus Error Timing Diagram

3.4.2 Retrying the Bus Cycle

If the internal bus error signal is asserted during a bus cycle in which $\overline{\text{HALT}}$ is asserted by an external device, the EC000 core will initiate a retry operation. Figure 3-27 is a timing diagram of the retry operation.

The EC000 core terminates the bus cycle, then puts the data bus in the high-impedance state. The processor remains in this state until $\overline{\text{HALT}}$ is negated. Then the processor retries the preceding cycle using the same function codes, address, and data (for a write operation). $\overline{\text{BERR}}$ should be negated at least one clock cycle before $\overline{\text{HALT}}$ is negated.

NOTES

There is no external connection to $\overline{\text{BERR}}$; hence, users can not normally initiate a retry operation. The internal bus error is asserted whenever a write protect violation, address decode conflict, or hardware watchdog timeout occurs (assuming the offending condition is enabled in the SCR).

To guarantee that the entire read-modify-write cycle runs correctly and that the write portion of the operation is performed without negating the address strobe, the processor does not retry a read-modify-write cycle. When $\overline{\text{BERR}}$ is asserted during a read-modify-write operation, a bus error operation is performed whether or not $\overline{\text{HALT}}$ is asserted.

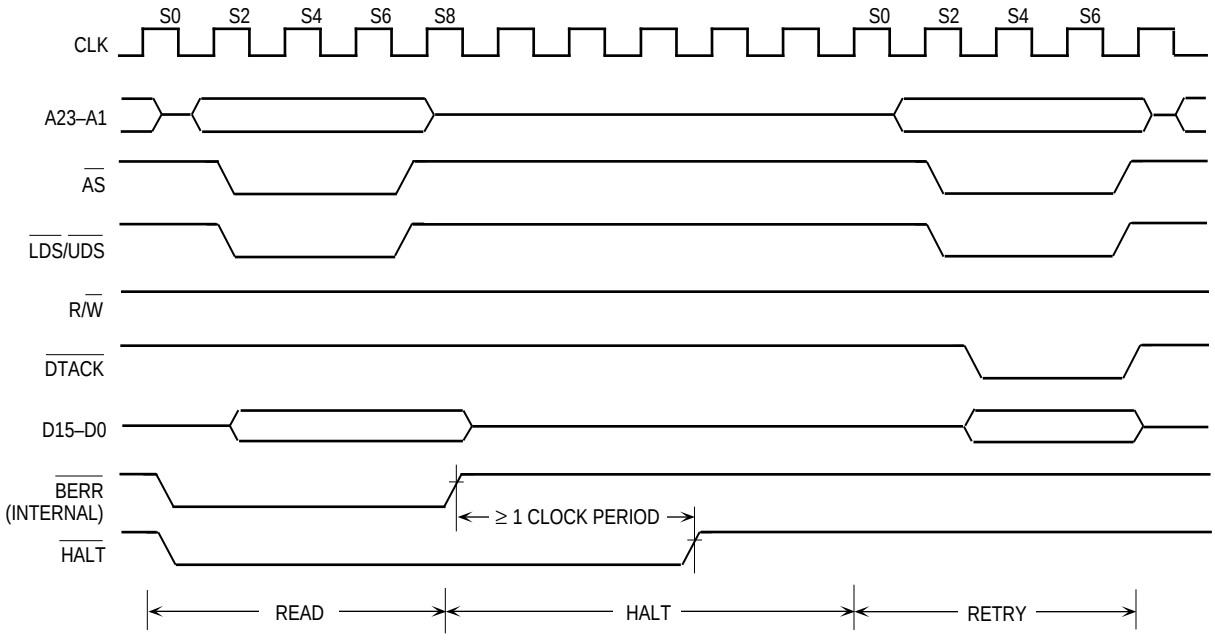


Figure 3-27. Retry Bus Cycle Timing Diagram

3.4.3 Halt Operation

$\overline{\text{HALT}}$ performs a halt/run/single-step operation. When $\overline{\text{HALT}}$ is asserted by an external device, the processor halts and remains halted as long as the signal remains asserted, as shown in Figure 3-28.

While the processor is halted, bus arbitration is performed as usual.

NOTE

If $\overline{\text{HALT}}$ is asserted while a RESET instruction is being executed, the CPU is reset.

The single-step mode is derived from correctly timed transitions of $\overline{\text{HALT}}$. $\overline{\text{HALT}}$ is negated to allow the processor to begin a bus cycle, then asserted to enter the halt mode when the cycle completes. The single-step mode proceeds through a program one bus cycle at a time for debugging purposes. The halt operation and the hardware trace capability allow tracing of either bus cycles or instructions one at a time. These capabilities and a software debugging package provide total debugging flexibility.

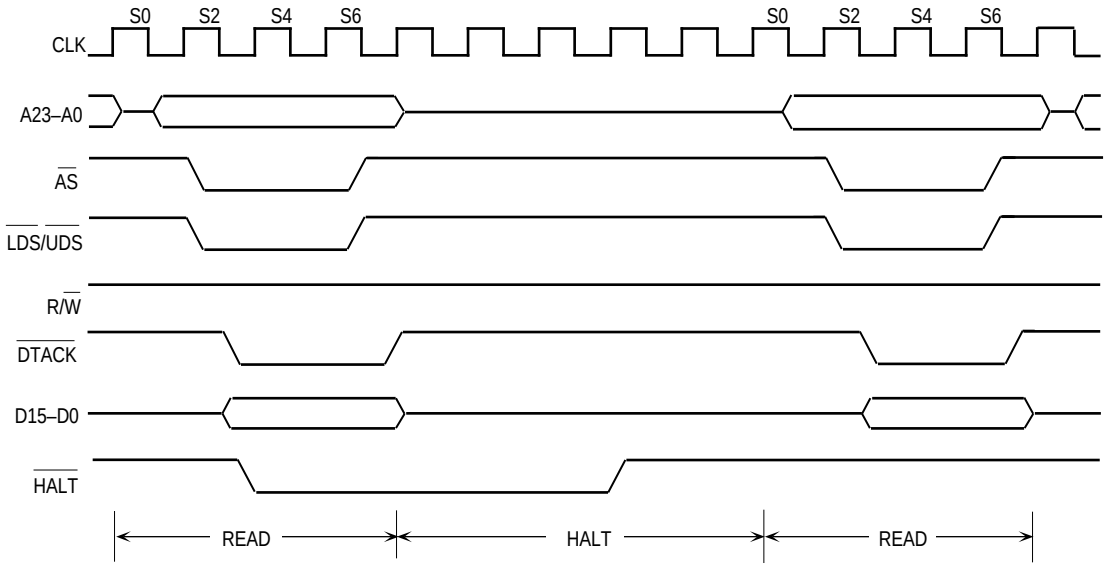


Figure 3-28. Halt Operation Timing Diagram

3.4.4 Double Bus Fault

When a bus error exception occurs, the processor begins exception processing by stacking information on the supervisor stack. If another bus error occurs during exception processing (i.e., before execution of another instruction begins) the processor halts and asserts $\overline{\text{HALT}}$. This is called a double bus fault. Only an external reset operation or a software watchdog timeout can restart a processor halted due to a double bus fault.

A double bus fault occurs during a reset operation when a bus error occurs while the processor is reading the vector table (before the first instruction is executed). The reset operation is described in the following paragraphs.

3.5 RESET OPERATION

$\overline{\text{RESET}}$ can be asserted externally for the initial processor reset. Subsequently, the signal can be asserted either externally or internally (executing a RESET instruction). For proper external reset operation, $\overline{\text{HALT}}$ must also be asserted.

The $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ bidirectional pins represent the standard M68000 method of reset. The MC68307 adds a master reset input ($\overline{\text{RSTIN}}$) which resets the MC68307. RSTIN generates a $\overline{\text{RESET}}$, causing external devices in the system to be reset; note that $\overline{\text{HALT}}$ is not asserted.

At initial power-on, the MC68307's power-on reset asserts $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ internally until V_{DD} reaches a minimum level. They are then held asserted for 32768 EXTAL clocks, to ensure that the clock source has time to stabilize.

Subsequent assertions of $\overline{\text{RSTIN}}$ also incur a 32768 clock hold after the negating edge, which equates to 2ms for a 16.667 MHz system clock. Because of this debouncing effect, this input is often used in preference to $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ when a reset switch is required.

After the processor is reset, it reads the reset vector table entry (address $00000) and loads the contents into the supervisor stack pointer (SSP). Next, the processor loads the contents of address $00004 (vector table entry 1) into the program counter. Then the processor initializes the interrupt level in the status register to a value of seven. No other register is affected by the reset sequence. Figure 3-29 shows the timing of the reset operation.

The active-low $\overline{\text{RESET}}$ signal is asserted by the EC000 core when a RESET instruction is executed. This signal should reset all external devices and internal peripherals (the EC000 core itself is not affected). The processor drives $\overline{\text{RESET}}$ for 124 clock periods. To guarantee a reset of the core during this time, internal logic will stretch any $\overline{\text{RESET}}$ or $\overline{\text{HALT}}$ assertion to 132 clocks.

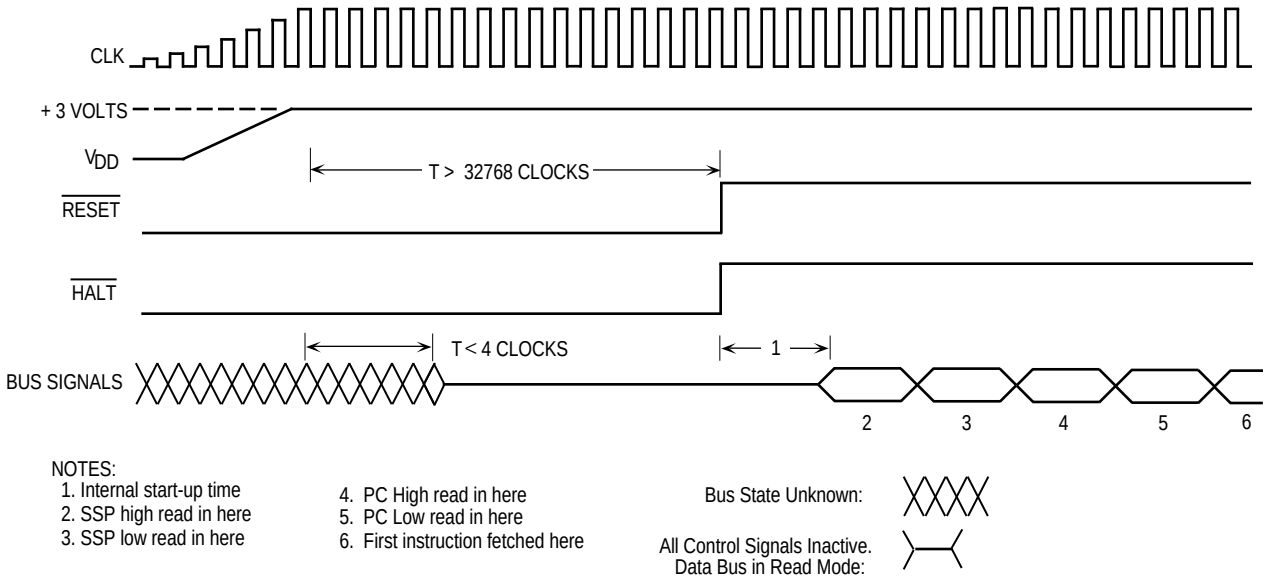


Figure 3-29. Power-On Reset Operation Timing Diagram

3.6 ASYNCHRONOUS OPERATION

To achieve clock frequency independence at a system level, the bus can be operated in an asynchronous manner. Asynchronous bus operation uses the bus handshake signals to control the transfer of data. The handshake signals are $\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, $\overline{\text{DTACK}}$, the internal $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$. $\overline{\text{AS}}$ indicates the start of the bus cycle, and $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ signal valid data for a write cycle. After placing the requested data on the data bus (read cycle) or latching the data (write cycle), the slave device (memory or peripheral) or the internal wait-state generator asserts $\overline{\text{DTACK}}$ to terminate the bus cycle. If no device responds or if the access is invalid, internal control logic asserts the internal $\overline{\text{BERR}}$, to abort the cycle. Figure 3-31 shows the use of the bus handshake signals in a fully asynchronous read cycle. Figure 3-30 shows a fully asynchronous write cycle.

In the asynchronous mode, the accessed device operates independently of the frequency and phase of the system clock. For example, the MC68681 dual universal asynchronous receiver/transmitter (DUART) does not require any clock-related information from the bus master during a bus transfer. Asynchronous devices are designed to operate correctly with processors at any clock frequency when relevant timing requirements are observed.

A device can use a clock at the same frequency as the system clock, but without a defined phase relationship to the system clock. This mode of operation is pseudo-asynchronous; it

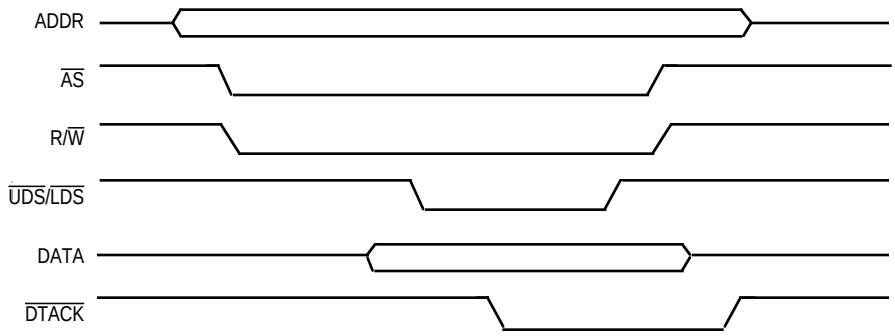


Figure 3-30. Fully Asynchronous Write Cycle

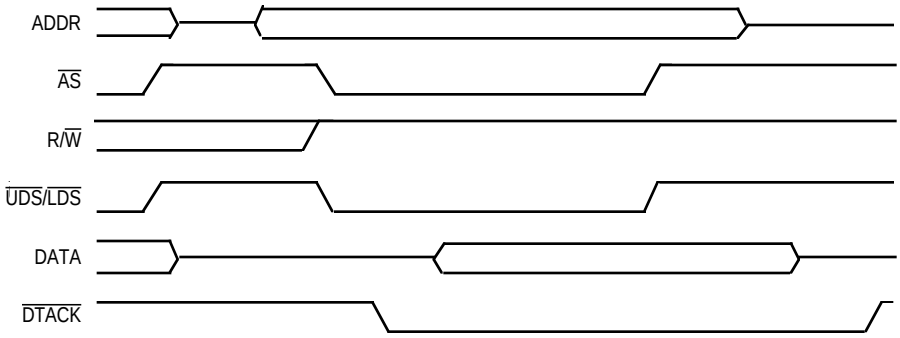


Figure 3-31. Fully Asynchronous Read Cycle

increases performance by observing timing parameters related to the system clock frequency without being completely synchronous with that clock. A memory array designed to operate with a particular frequency processor but not driven by the processor clock is a common example of a pseudo-asynchronous device.

The designer of a fully asynchronous system can make no assumptions about address setup time, which could be used to improve performance. With the system clock frequency known, the slave device can be designed to decode the address bus before recognizing an address strobe. Parameter #11 (refer to **Section 11.7 AC Electrical Specifications—Read and Write Cycles (VCC = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; TA = TL to TH) (see Figure 11-3 and Figure 11-4)**) specifies the minimum time before address strobe during which the address is valid.

In a pseudo-asynchronous system, timing specifications allow $\overline{\text{DTACK}}$ to be asserted for a read cycle before the data from a slave device is valid. The length of time that $\overline{\text{DTACK}}$ may precede data is specified as parameter #31. This parameter must be met to ensure the validity of the data latched into the processor. No maximum time is specified from the assertion of $\overline{\text{AS}}$ to the assertion of $\overline{\text{DTACK}}$. During this unlimited time, the processor inserts wait cycles in one-clock-period increments until $\overline{\text{DTACK}}$ is recognized. Figure 3-33 shows the important timing parameters for a pseudo-asynchronous read cycle.

During a write cycle, after the processor asserts $\overline{\text{AS}}$ but before driving the data bus, the processor drives R/$\overline{\text{W}}$ low. Parameter #55 specifies the minimum time between the transition

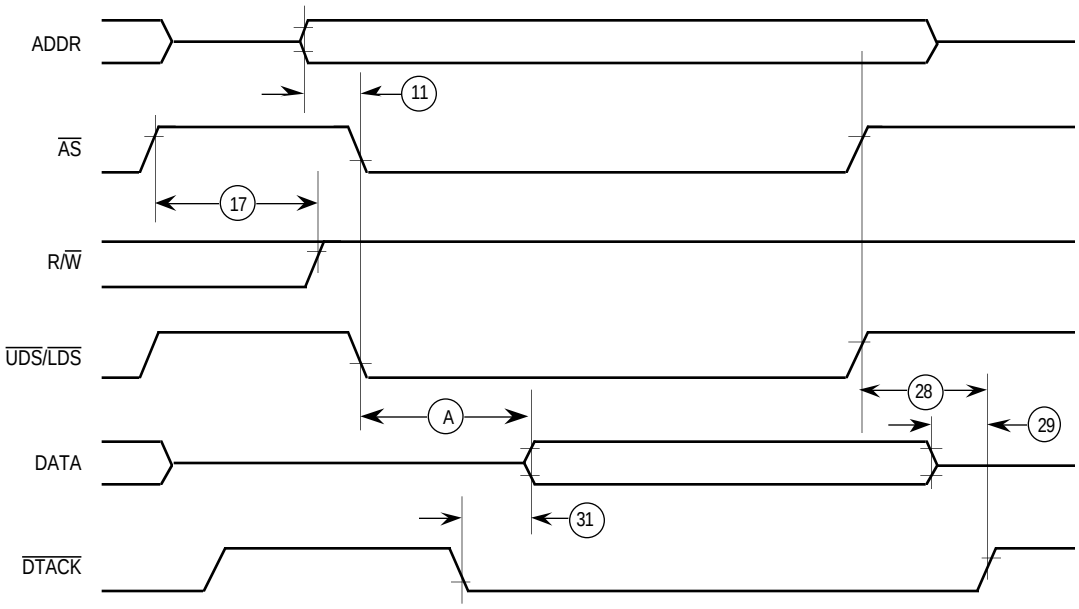


Figure 3-32. Pseudo-Asynchronous Read Cycle

of R/$\overline{W}$ and the driving of the data bus, which is effectively the maximum turnoff time for any device driving the data bus.

After the processor places valid data on the bus, it asserts the data strobe signal(s). A data setup time, similar to the address setup time previously discussed, can be used to improve performance. Parameter #26 is the minimum time a slave device can accept valid data before recognizing a data strobe. The slave device asserts $\overline{DTACK}$ after it accepts the data. Parameter #25 is the minimum time after negation of the strobes during which the valid data remains on the address bus. Parameter #28 is the maximum time between the negation of the strobes by the processor and the negation of $\overline{DTACK}$ by the slave device. If $\overline{DTACK}$ remains asserted past the time specified by parameter #28, the processor may recognize it as being asserted early in the next bus cycle and may terminate that cycle prematurely. Figure 3-33 shows the important timing specifications for a pseudo-asynchronous write cycle.

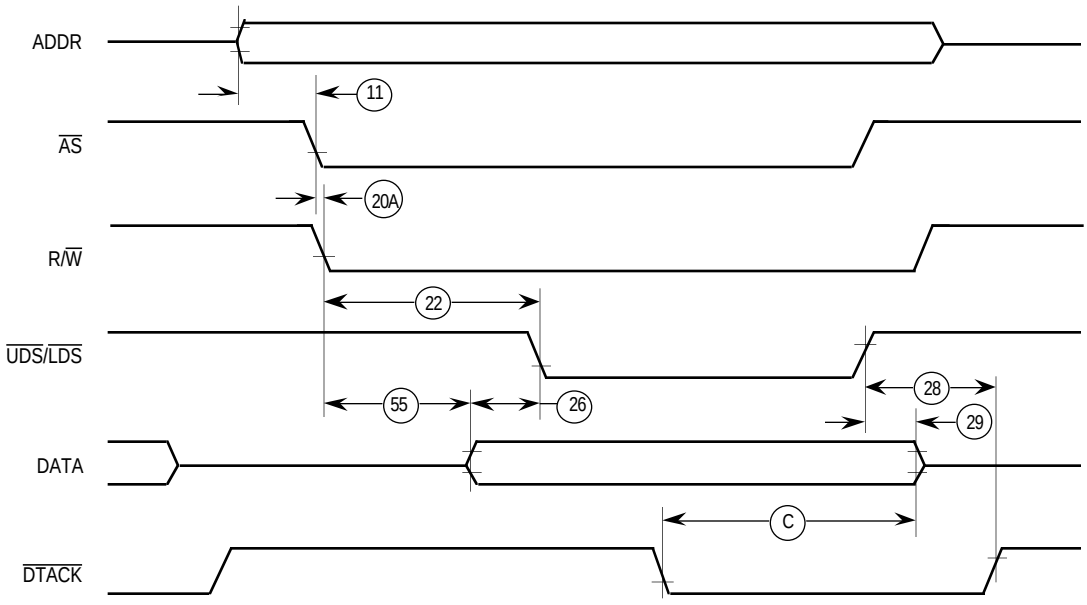


Figure 3-33. Pseudo-Asynchronous Write Cycle

3.7 SYNCHRONOUS OPERATION

In some systems, external devices use the system clock to generate $\overline{\text{DTACK}}$ and other asynchronous input signals. This synchronous operation provides a closely coupled design with maximum performance, appropriate for frequently accessed parts of the system. For example, memory can operate in the synchronous mode, but peripheral devices operate asynchronously. For a synchronous device, the designer uses explicit timing information shown in **Section 11.7 AC Electrical Specifications—Read and Write Cycles (VCC = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; TA = TL to TH) (see Figure 11-3 and Figure 11-4)**. These specifications define the state of all bus signals relative to a specific state of the processor clock.

The standard M68000 bus cycle consists of four clock periods (eight bus cycle states) and, optionally, an integral number of clock cycles inserted as wait states. Wait states are inserted as required to allow sufficient response time for the external device. The following state-by-state description of the bus cycle differs from those descriptions in **Section 3.1.2 16-Bit M68000 Bus Read Cycle** and **Section 3.1.3 16-Bit M68000 Bus Write Cycle** by including information about the important timing parameters that apply in the bus cycle states.

STATE 0 — The bus cycle starts in S0, during which the clock is high. At the rising edge of S0, the function code for the access is driven externally. Parameter #6A defines the delay from this rising edge until the function codes are valid. Also, the R/$\overline{\text{W}}$ signal is driven high; parameter #18 defines the delay from the same rising edge to the transition of R/$\overline{\text{W}}$. The minimum value for parameter #18 applies to a read cycle preceded by a write cycle; this value is the maximum hold time for a low on R/$\overline{\text{W}}$ beyond the initiation of the read cycle.

STATE 1 — Entering S1, a low period of the clock, the address of the accessed device is driven externally with an assertion delay defined by parameter #6.

STATE 2 — On the rising edge of S2, a high period of the clock, $\overline{\text{AS}}$ is asserted. During a read cycle, $\overline{\text{UDS}}$ and/or $\overline{\text{LDS}}$ is also asserted at this time. Parameter #9 defines the assertion delay for these signals. For a write cycle, the R/$\overline{\text{W}}$ signal is driven low with a delay defined by parameter #20.

STATE 3 — On the falling edge of the clock entering S3, the data bus is driven out of the high-impedance state with the data being written to the accessed device (in a write cycle). Parameter #23 specifies the data assertion delay. In a read cycle, no signal is altered in S3.

STATE 4 — Entering the high clock period of S4, $\overline{\text{UDS}}$/$\overline{\text{LDS}}$ is asserted (during a write cycle) on the rising edge of the clock. As in S2 for a read cycle, parameter #9 defines the assertion delay from the rising edge of S4 for $\overline{\text{UDS}}$/$\overline{\text{LDS}}$. In a read cycle, no signal is altered by the processor during S4.

Until the falling edge of the clock at the end of S4 (beginning of S5), no response from any external device except $\overline{\text{RESET}}$ is acknowledged by the

processor. If $\overline{\text{DTACK}}$ is asserted before the falling edge of S4 and satisfies the input setup time defined by parameter #47, the processor enters S5 and the bus cycle continues. If $\overline{\text{DTACK}}$ is asserted but without meeting the setup time defined by parameter #47, the processor may recognize the signal and continue the bus cycle; the result is unpredictable. If $\overline{\text{DTACK}}$ is not asserted before the next rise of clock, the bus cycle remains in S4, and wait states (complete clock cycles) are inserted until one of the bus cycle terminations is met. $\overline{\text{DTACK}}$ is normally generated by the internal wait-state generator.

STATE 5 S5 is a low period of the clock, during which the processor does not alter any signal.

STATE 6 S6 is a high period of the clock, during which data for a read operation is set up relative to the falling edge (entering S7). Parameter #27 defines the minimum period by which the data must precede the falling edge. For a write operation, the processor changes no signal during S6.

STATE 7 On the falling edge of the clock entering S7, the processor latches data and negates $\overline{\text{AS}}$ and $\overline{\text{UDS}}$/$\overline{\text{LDS}}$ during a read cycle. The hold time for these strobes from this falling edge is specified by parameter #12. The hold time for data relative to the negation of $\overline{\text{AS}}$ and $\overline{\text{UDS}}$/$\overline{\text{LDS}}$ is specified by parameter #29. For a write cycle, only $\overline{\text{AS}}$ and $\overline{\text{UDS}}$/$\overline{\text{LDS}}$, are negated; timing parameter #12 also applies.

On the rising edge of the clock, at the end of S7 (which may be the start of S0 for the next bus cycle), the processor places the address bus in the high-impedance state. During a write cycle, the processor also places the data bus in the high-impedance state and drives R/$\overline{\text{W}}$ high. External logic circuitry should respond to the negation of the $\overline{\text{AS}}$ and $\overline{\text{UDS}}$/$\overline{\text{LDS}}$ by negating $\overline{\text{DTACK}}$, if it was asserted externally. Parameter #28 is the hold time for $\overline{\text{DTACK}}$.

Figure 3-34 shows a synchronous read cycle and the important timing parameters that apply. The timing for a synchronous read cycle, including relevant timing parameters, is shown in Figure 3-35.

A key consideration when designing in a synchronous environment is the timing for the assertion of $\overline{\text{DTACK}}$ by an external device. To properly use external inputs, the processor must synchronize these signals to the internal clock. The processor must sample the external signal, which has no defined phase relationship to the CPU clock, which may be changing at sampling time, and must determine whether to consider the signal high or low during the succeeding clock period. Successful synchronization requires that the internal machine receives a valid logic level, whether the input is high, low, or in transition.

Parameter #47 of **Section 11.7 AC Electrical Specifications—Read and Write Cycles (VCC = 5.0 V ± 0.5 V or 3.3 Vdc ± 0.3 V; GND = 0 Vdc; TA = TL to TH) (see Figure 11-3 and Figure 11-4)** is the asynchronous input setup time. Signals that meet parameter #47 are guaranteed to be recognized at the next falling edge of the system clock. However, signals that do not meet parameter #47 are not guaranteed to be recognized. In addition, if $\overline{\text{DTACK}}$

is recognized on a falling edge, valid data is latched into the processor (during a read cycle) on the next falling edge, provided the data meets the setup time required (parameter #27). When parameter #27 has been met, parameter #31 may be ignored. If $\overline{\text{DTACK}}$ is asserted with the required setup time before the falling edge of S4, no wait states are incurred, and the bus cycle runs at its maximum speed of four clock periods.

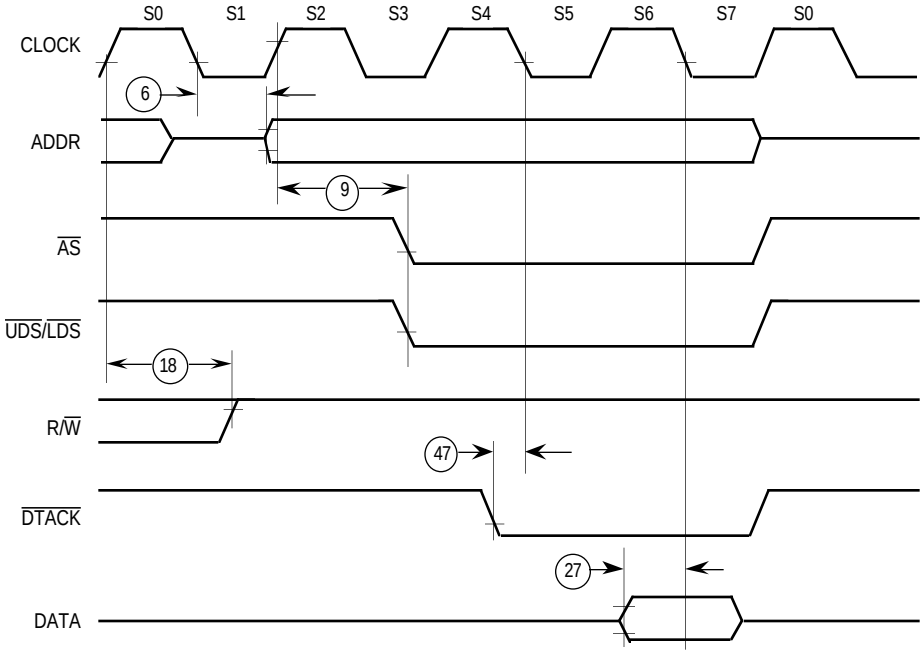


Figure 3-34. Synchronous Read Cycle

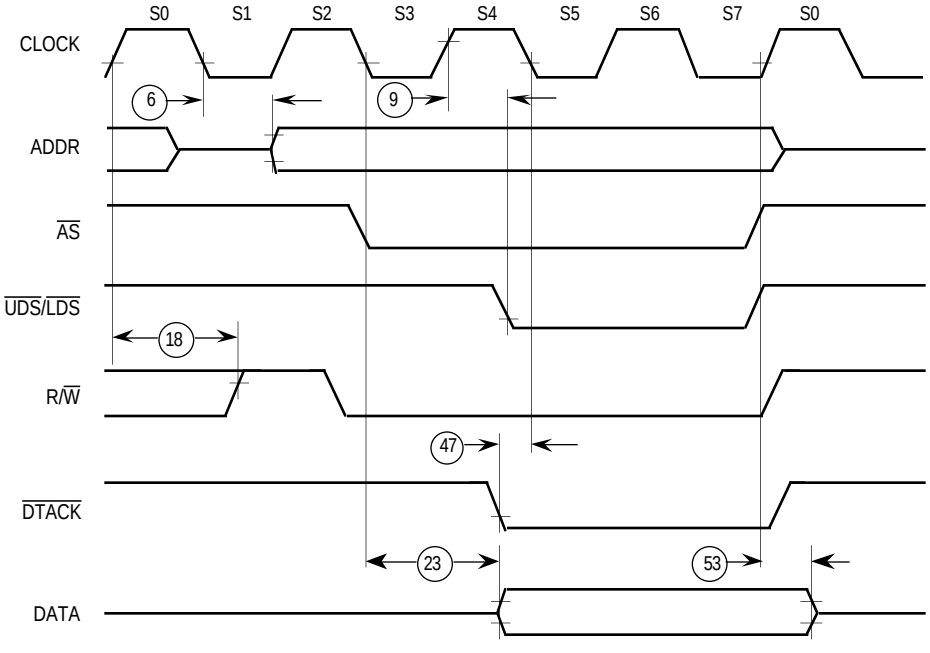


Figure 3-35. Synchronous Write Cycle

SECTION 4
EC000 CORE PROCESSOR

The EC000 core has a 16-bit data bus and 32-bit address bus while the full architecture provides for 32-bit address and data register operations.

4.1 FEATURES

The following resources are available to the EC000 core:

- Eight 32-Bit Address Registers
- Eight 32-Bit Data Registers
- 56 Powerful Instructions
- Operations on Five Main Data Types
- Memory-Mapped Input/Output (I/O)
- 14 Addressing Modes

4.2 PROCESSING STATES

The processor is always in one of three states: normal processing, exception processing, or halted. It is in the normal processing state when executing instructions, fetching instructions and operands, and storing instruction results.

Exception processing is the transition from program processing to system, interrupt, and exception handling. Exception processing includes fetching the exception vector, stacking operations, and refilling the instruction pipe after an exception. The processor enters exception processing when an exceptional internal condition arises such as tracing an instruction, an instruction results in a trap, or executing specific instructions. External conditions, such as interrupts and access errors, also cause exceptions. Exception processing ends when the first instruction of the exception handler begins to execute.

The processor halts when it receives an access error or generates an address error while in the exception processing state. For example, if during exception processing of one access error another access error occurs, the processor is unable to complete the transition to normal processing and cannot save the internal state of the machine. The processor assumes that the system is not operational and halts. Only an external reset can restart a halted processor. Note that when the processor executes a STOP instruction, it is in a special type of normal processing state, one without bus cycles. The processor stops, but it does not halt.

4.3 PROGRAMMING MODEL

The EC000 core executes instructions in one of two modes—user mode or supervisor mode. The user mode provides the execution environment for the majority of application programs. The supervisor mode, which allows some additional instructions and privileges, is used by the operating system and other system software.

To provide upward compatibility of code written for a specific implementation of the EC000 core, the user programmer's model, illustrated in Figure 4-1, is common to all implementations. In the user programmer's model, the EC000 core offers 16, 32-bit, general-purpose registers (D7–D0, A7–A0), a 32-bit program counter, and an 8-bit condition code register. The first eight registers (D7–D0) are used as data registers for byte (8-bit), word (16-bit), and long-word (32-bit) operations. The second set of seven registers (A6–A0) and the user stack pointer (USP) can be used as software stack pointers and base address registers. In addition, the address registers can be used for word and long-word operations. All of the 16 registers can be used as index registers. The supervisor programmer's model consists of supplementary registers used in the supervisor mode.

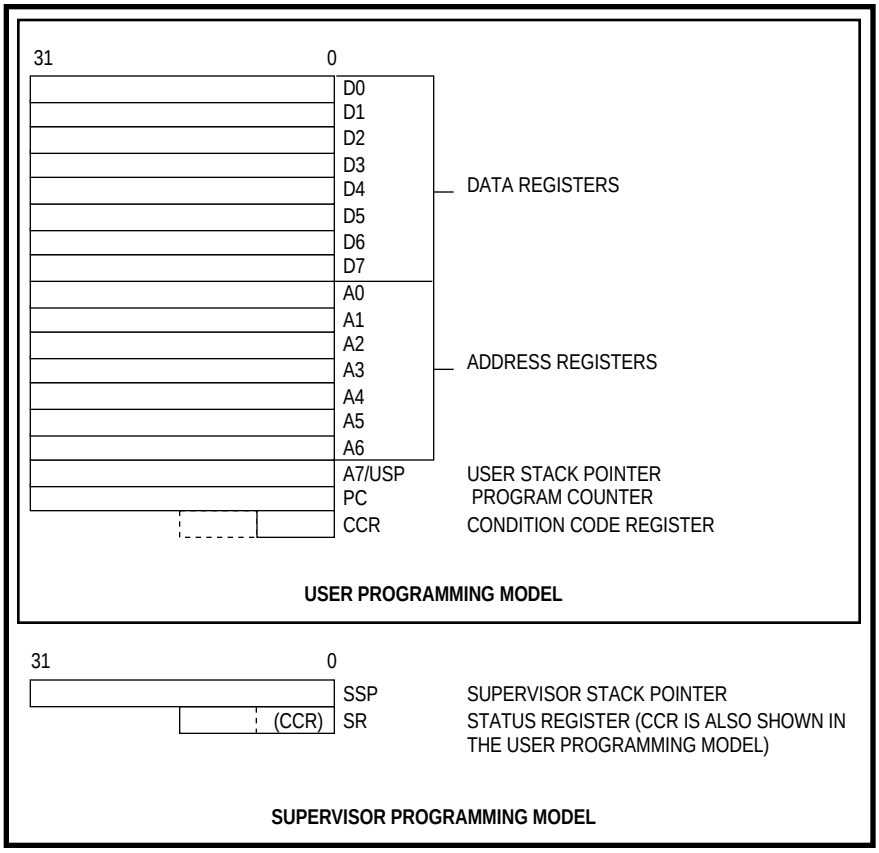


Figure 4-1. Programming Model

The status register, illustrated in Figure 4-2, contains the interrupt mask (eight levels available) and the following condition codes: overflow (V), zero (Z), negative (N), carry (C), and extend (X). Additional status bits indicate that the processor is in the trace (T) mode and/or in the supervisor (S) state.

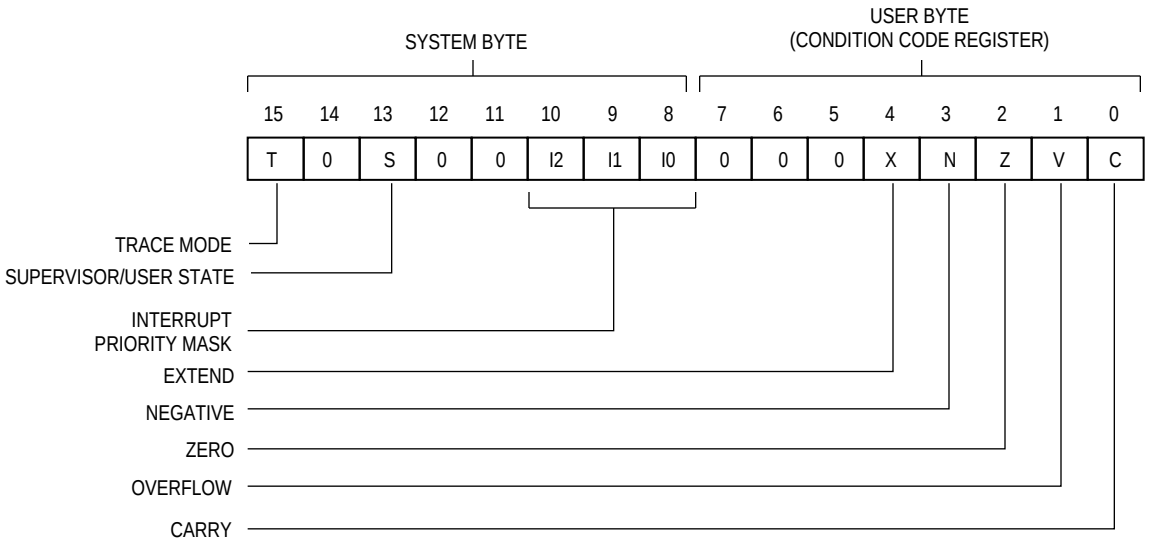


Figure 4-2. Status Register

4.3.1 Data Format Summary

The processor supports the basic data formats of the M68000 family. The instruction set supports operations on other data formats such as memory addresses.

The operand data formats supported by the integer unit (IU) are the standard twos-complement data formats defined in the M68000 family architecture. Registers, memory, or instructions themselves can contain IU operands. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Table 4-1 lists the data formats for the processor. Refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual,* for details on data format organization in registers and memory.

Table 4-1. Processor Data Formats

Operand Data Format	Size	Notes
Bit	1 Bit	—
Binary-Coded Decimal (BCD)	8 Bits	Packed: 2 Digits/Byte; Unpacked: 1 Digit/Byte
Byte Integer	8 Bits	—
Word Integer	16 Bits	—
Long-Word Integer	32 Bits	—

4.3.2 Addressing Capabilities Summary

The EC000 core supports the basic addressing modes of the M68000 family. The register indirect addressing modes support postincrement, predecrement, offset, and indexing, which are particularly useful for handling data structures common to sophisticated applications and high-level languages. The program counter indirect mode also has indexing and offset capabilities. This addressing mode is typically required to support position-independent software. Besides these addressing modes, the processor provides index sizing and scaling features.

An instruction's addressing mode can specify the value of an operand, a register containing the operand, or how to derive the effective address of an operand in memory. Each addressing mode has an assembler syntax. Some instructions imply the addressing mode for an operand. These instructions include the appropriate fields for operands that use only one addressing mode. Table 4-2 lists a summary of the effective addressing modes for the processor. Refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual,* for details on instruction format and addressing modes.

Table 4-2. Effective Addressing Modes

Addressing Modes	Syntax
Register Direct Addressing Data Register Direct Address Register Direct	 EA = Dn EA = An
Absolute Data Addressing Absolute Short Absolute Long	 EA = (Next Word) EA = (Next Two Words)
Program Counter Relative Addressing Relative with Offset Relative with Index and Offset	 EA = (PC)+d_{16} EA = (PC)+d_8
Register Indirect Addressing Register Indirect Postincrement Register Indirect Predecrement Register Indirect Register Indirect with Offset Indexed Register Indirect with Offset	 EA = (An) EA = (An), An ¨ An+N An ¨ An–N, EA = (An) EA = (An)+d_{16} EA = (An)+(Xn)+d_8
Immediate Data Addressing Immediate Quick Immediate	 DATA = Next Word(s) Inherent Data
Implied Addressing Implied Register	 EA = SR, USP, SSP, PC

4.3.3 Notation Conventions

Table 4-3 lists the notation conventions used in this manual unless otherwise specified.

Table 4-3. Notation Conventions

Single and Double Operand Operations	
+	Arithmetic addition or postincrement indicator.
–	Arithmetic subtraction or predecrement indicator.
×	Arithmetic multiplication.
÷	Arithmetic division or conjunction symbol.
~	Invert; operand is logically complemented.
Λ	Logical AND
V	Logical OR
≈	Logical exclusive OR
⇒	Source operand is moved to destination operand.
⇔	Two operands are exchanged.
<op>	Any double-operand operation.
<operand>tested	Operand is compared to zero and the condition codes are set appropriately.
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion.
Other Operations	
TRAP	Equivalent to Format ÷ Offset Word ⇒ (SSP); SSP – 2 ⇒ SSP; PC ⇒ (SSP); SSP – 4 ⇒ SSP; SR ⇒ (SSP); SSP – 2 ⇒ SSP; (Vector) ⇒ PC
STOP	Enter the stopped state, waiting for interrupts.
<operand>$_{10}$	The operand is BCD; operations are performed in decimal.

Table 4-3. Notation Conventions (Continued)

If <condition> then <operations> else <operations>	Test the condition. If true, the operations after “then” are performed. If the condition is false and the optional “else” clause is present, the operations after “else” are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.
Register Specification	
An	Any Address Register n (example: A3 is address register 3)
Ax, Ay	Source and destination address registers, respectively.
BR	Base Register—An, PC, or suppressed.
Dc	Data register D7–D0, used during compare.
Dh, Dl	Data registers high- or low-order 32 bits of product.
Dn	Any Data Register n (example: D5 is data register 5)
Dr, Dq	Data register’s remainder or quotient of divide.
Du	Data register D7–D0, used during update.
Dx, Dy	Source and destination data registers, respectively.
Rn	Any Address or Data Register
Rx, Ry	Any source and destination registers, respectively.
Xn	Index Register—An, Dn, or suppressed.
Data Format And Type	
<fmt>	Operand Data Format: Byte (B), Word (W), Long (L), or Packed (P).
B, W, L	Specifies a signed integer data type (twos complement) of byte, word, or long word.
k	A twos complement signed integer (–64 to +17) specifying a number’s format to be stored in the packed decimal format.
Subfields and Qualifiers	
#<xxx> or #<data>	Immediate data following the instruction word(s).
( )	Identifies an indirect address in a register.
[ ]	Identifies an indirect address in memory.
bd	Base Displacement
d_n	Displacement Value, n Bits Wide (example: d_{16} is a 16-bit displacement).
LSB	Least Significant Bit
LSW	Least Significant Word
MSB	Most Significant Bit
MSW	Most Significant Word
od	Outer Displacement
SCALE	A scale factor (1, 2, 4, or 8, for no-word, word, long-word, or quad-word scaling, respectively).
SIZE	The index register’s size (W for word, L for long word).
{offset:width}	Bit field selection.
Register Names	
CCR	Condition Code Register (lower byte of status register)
PC	Program Counter
SR	Status Register
Register Codes	
*	General Case.
C	Carry Bit in CCR
cc	Condition Codes from CCR
FC	Function Code
N	Negative Bit in CCR
U	Undefined, Reserved for Motorola Use.
V	Overflow Bit in CCR
X	Extend Bit in CCR
Z	Zero Bit in CCR
—	Not Affected or Applicable.

Table 4-3. Notation Conventions (Continued)

Stack Pointers	
SP	Active Stack Pointer
SSP	Supervisor Stack Pointer
USP	User Stack Pointer
Miscellaneous	
<ea>	Effective Address
<label>	Assemble Program Label
<list>	List of registers, for example D3–D0.
LB	Lower Bound
m	Bit m of an Operand
m–n	Bits m through n of Operand
UB	Upper Bound

4.4 EC000 CORE INSTRUCTION SET OVERVIEW

Design of the instruction set gives special emphasis to support of structured, high-level languages and to ease of assembly language programming. Each instruction, with a few exceptions, operates on bytes, words, and long words, and most instructions can use any of the 14 addressing modes. Over 1000 useful instructions are provided by combining instruction types, data types, and addressing modes. These instructions include signed and unsigned multiply and divide, “quick” arithmetic operations, BCD arithmetic, and expanded operations (through traps). Additionally, the highly symmetric, proprietary microcoded structure of the instruction set provides a sound, flexible base for the future.

The EC000 core instruction set is listed in Table 4-4. For detailed information on the EC000 core instruction set, refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*.

Table 4-4. EC000 Core Instruction Set Summary

Opcode	Operation	Syntax
ABCD	BCD Source + BCD Destination + X ⇒ Destination	ABCD Dy,Dx ABCD –(Ay),–(Ax)
ADD	Source + Destination ⇒ Destination	ADD <ea>,Dn ADD Dn,<ea>
ADDA	Source + Destination ⇒ Destination	ADDA <ea>,An
ADDI	Immediate Data + Destination ⇒ Destination	ADDI #<data>,<ea>
ADDQ	Immediate Data + Destination ⇒ Destination	ADDQ #<data>,<ea>
ADDX	Source + Destination + X ⇒ Destination	ADDX Dy,Dx ADDX –(Ay),–(Ax)
AND	Source Λ Destination ⇒ Destination	AND <ea>,Dn AND Dn,<ea>
ANDI	Immediate Data Λ Destination ⇒ Destination	ANDI #<data>,<ea>
ANDI to CCR	Source Λ CCR ⇒ CCR	ANDI #<data>,CCR
ANDI to SR	If supervisor state then Source Λ SR ⇒ SR else TRAP	ANDI #<data>,SR
ASL, ASR	Destination Shifted by count ⇒ Destination	ASd Dx,Dy[1] ASd #<data>,Dy[1] ASd <ea>[1]
Bcc	If condition true then PC + d_n ⇒ PC	Bcc <label>

Table 4-4. EC000 Core Instruction Set Summary (Continued)

BCHG	~(bit number of Destination) ⇒ Z; ~(bit number of Destination) ⇒ (bit number) of Destination	BCHG Dn,<ea> BCHG #<data>,<ea>
BCLR	~(bit number of Destination) ⇒ Z; 0 ⇒ bit number of Destination	BCLR Dn,<ea> BCLR #<data>,<ea>
BRA	PC + d_n ⇒ PC	BRA <label>
BSET	~(bit number of Destination) ⇒ Z; 1 ⇒ bit number of Destination	BSET Dn,<ea> BSET #<data>,<ea>
BSR	SP – 4 ⇒ SP; PC ⇒ (SP); PC + d_n ⇒ PC	BSR <label>
BTST	–(bit number of Destination) ⇒ Z;	BTST Dn,<ea> BTST #<data>,<ea>
CHK	If Dn < 0 or Dn > Source then TRAP	CHK <ea>,Dn
CLR	0 ⇒ Destination	CLR <ea>
CMP	Destination – Source ⇒ cc	CMP <ea>,Dn
CMPA	Destination – Source	CMPA <ea>,An
CMPI	Destination – Immediate Data	CMPI #<data>,<ea>
CMPM	Destination – Source ⇒ cc	CMPM (Ay)+,(Ax)+
DBcc	If condition false then (Dn–1 ⇒ Dn; If Dn ≠ –1 then PC + d_n ⇒ PC)	DBcc Dn,<label>
DIVS	Destination ÷ Source ⇒ Destination	DIVS.W <ea>,Dn 32 ÷ 16 ⇒ 16r:16q DIVS.L <ea>,Dq 32 ÷ 32 ⇒ 32q DIVS.L <ea>,Dr:Dq 64 ÷ 32 ⇒ 32r:32q
DIVU	Destination ÷ Source ⇒ Destination	DIVU.W <ea>,Dn 32 ÷ 16 ⇒ 16r:16q DIVU.L <ea>,Dq 32 ÷ 32 ⇒ 32q DIVU.L <ea>,Dr:Dq 64 ÷ 32 ⇒ 32r:32q
EOR	Source ⊕ Destination ⇒ Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination ⇒ Destination	EORI #<data>,<ea>
EORI to CCR	Source ⊕ CCR ⇒ CCR	EORI #<data>,CCR
EORI to SR	If supervisor state then Source ⊕ SR ⇒ SR else TRAP	EORI #<data>,SR
EXG	Rx ⇔ Ry	EXG Dx,Dy EXG Ax,Ay EXG Dx,Ay EXG Ay,Dx
EXT	Destination Sign – Extended ⇒ Destination	EXT.W Dn extend byte to word EXT.L Dn extend word to long word
JMP	Destination Address ⇒ PC	JMP <ea>
JSR	SP – 4 ⇒ SP; PC ⇒ (SP) Destination Address ⇒ PC	JSR <ea>
LEA	<ea> ⇒ An	LEA <ea>,An
LINK	SP – 4 ⇒ SP; An ⇒ (SP) SP ⇒ An, SP+d ⇒ SP	LINK An,d_n
LSL, LSR	Destination Shifted by count ⇒ Destination	LSd Dx,Dy[1] LSd #<data>,Dy[1] LSd <ea>[1]
MOVE	Source ⇒ Destination	MOVE <ea>,<ea>
MOVE from SR	If supervisor state then SR ⇒ Destination else TRAP	MOVE SR,<ea>
MOVE to CCR	Source ⇒ CCR	MOVE <ea>,CCR
MOVE to SR	If supervisor state then Source ⇒ SR else TRAP	MOVE <ea>,SR
MOVE USP	If supervisor state then USP ⇒ An or An ⇒ USP else TRAP	MOVE USP,An MOVE An,USP

Table 4-4. EC000 Core Instruction Set Summary (Continued)

MOVEA	Source ⇒ Destination	MOVEA <ea>,An
MOVEM	Registers ⇒ Destination Source ⇒ Registers	MOVEM <list>,<ea>[2] MOVEM <ea>,<list>[2]
MOVEP	Source ⇒ Destination	MOVEP Dx,(d_n,Ay) MOVEP (d_n,Ay),Dx
MOVEQ	Immediate Data ⇒ Destination	MOVEQ #<data>,Dn
MULS	Source × Destination ⇒ Destination	MULS.W <ea>,Dn 16 × 16 ⇒ 32 MULS.L <ea>,Dl 32 × 32 ⇒ 32 MULS.L <ea>,Dh–Dl 32 × 32 ⇒ 64
MULU	Source × Destination ⇒ Destination	MULU.W <ea>,Dn 16 × 16 ⇒ 32 MULU.L <ea>,Dl 32 × 32 ⇒ 32 MULU.L <ea>,Dh–Dl 32 × 32 ⇒ 64
NBCD	0 – (Destination$_{10}$) – X ⇒ Destination	NBCD <ea>
NEG	0 – (Destination) ⇒ Destination	NEG <ea>
NEGX	0 – (Destination) – X ⇒ Destination	NEGX <ea>
NOP	None	NOP
NOT	~ Destination ⇒ Destination	NOT <ea>
OR	Source V Destination ⇒ Destination	OR <ea>,Dn OR Dn,<ea>
ORI	Immediate Data V Destination ⇒ Destination	ORI #<data>,<ea>
ORI to CCR	Source V CCR ⇒ CCR	ORI #<data>,CCR
ORI to SR	If supervisor state then Source V SR ⇒ SR else TRAP	ORI #<data>,SR
PEA	SP – 4 ⇒ SP; <ea> ⇒ (SP)	PEA <ea>
RESET	If supervisor state then Assert $\overline{\text{RSTO}}$ Line else TRAP	RESET
ROL, ROR	Destination Rotated by count ⇒ Destination	ROd Rx,Dy[1] ROd #<data>,Dy[1]
ROXL, ROXR	Destination Rotated with X by count ⇒ Destination	ROXd Dx,Dy[1] ROXd #<data>,Dy[1] ROXd <ea>[1]
RTE	If supervisor state then (SP) ⇒ SR; SP + 2 ⇒ SP; (SP) ⇒ PC; SP + 4 ⇒ SP; restore state and deallocate stack according to (SP) else TRAP	RTE
RTR	(SP) ⇒ CCR; SP + 2 ⇒ SP; (SP) ⇒ PC; SP + 4 ⇒ SP	RTR
RTS	(SP) ⇒ PC; SP + 4 ⇒ SP	RTS
SBCD	Destination$_{10}$ – Source$_{10}$ – X ⇒ Destination	SBCD Dx,Dy SBCD –(Ax),–(Ay)
Scc	If condition true then 1s ⇒ Destination else 0s ⇒ Destination	Scc <ea>
STOP	If supervisor state then Immediate Data ⇒ SR; STOP else TRAP	STOP #<data>
SUB	Destination – Source ⇒ Destination	SUB <ea>,Dn SUB Dn,<ea>
SUBA	Destination – Source ⇒ Destination	SUBA <ea>,An
SUBI	Destination – Immediate Data ⇒ Destination	SUBI #<data>,<ea>
SUBQ	Destination – Immediate Data ⇒ Destination	SUBQ #<data>,<ea>
SUBX	Destination – Source – X ⇒ Destination	SUBX Dx,Dy SUBX –(Ax),–(Ay)
SWAP	Register 31–16 ⇔ Register 15–0	SWAP Dn

Table 4-4. EC000 Core Instruction Set Summary (Continued)

TAS	Destination Tested ⇒ Condition Codes; 1 ⇒ bit 7 of Destination	TAS <ea>
TRAP	SSP – 2 ⇒ SSP; Format ÷ Offset ⇒ (SSP); SSP – 4 ⇒ SSP; PC ⇒ (SSP); SSP – 2 ⇒ SSP; SR ⇒ (SSP); Vector Address ⇒ PC	TRAP #<vector>
TRAPV	If V then TRAP	TRAPV
TST	Destination Tested ⇒ Condition Codes	TST <ea>
UNLK	An ⇒ SP; (SP) ⇒ An; SP + 4 ⇒ SP	UNLK An

NOTES:
1. d is direction, left or right.
2. List refers to register.

4.5 EXCEPTION PROCESSING

This section describes the processing for each type of exception, exception priorities, the return from an exception, and bus fault recovery. This section also describes the formats of the exception stack frames.

Exception processing is the activity performed by the processor in preparing to execute a special routine for any condition that causes an exception. In particular, exception processing does not include the execution of the routine itself. Exception processing is the transition from the normal processing of a program to the processing required for any special internal or external condition that preempts normal processing. External conditions that cause exceptions are interrupts from external devices, bus errors, and resets. Internal conditions that cause exceptions are instructions, address errors, and tracing. For example, the TRAP, TRAPV, CHK, RTE, and DIV instructions can generate exceptions as part of their normal execution. In addition, illegal instructions and privilege violations cause exceptions. Exception processing uses an exception vector table and an exception stack frame.

Exception processing occurs in four functional steps. However, all individual bus cycles associated with exception processing (vector acquisition, stacking, etc.) are not guaranteed to occur in the order in which they are described in this section. Figure 4-4 illustrates a general flowchart for the steps taken by the processor during exception processing.

During the first step, the processor makes an internal copy of the status register, S-bit, T-bits, I-bits (SR). Then the processor changes to the supervisor mode by setting the S-bit and inhibits tracing of the exception handler by clearing the trace enable (T) bit in the SR. For the reset and interrupt exceptions, the processor also updates the interrupt priority mask in the SR

During the second step, the processor determines the vector number for the exception. For interrupts, the processor performs an interrupt acknowledge bus cycle to obtain the vector number. For all other exceptions, internal logic provides the vector number. This vector number is used in the last step to calculate the address of the exception vector. Throughout this section, vector numbers are given in decimal notation.

The third step is to save the current processor contents for all exceptions other than reset exception, which does not stack information. The processor creates an exception stack frame on the active supervisor stack and fills it with information appropriate for the type of

exception. Other information can also be stacked, depending on which exception is being processed and the state of the processor prior to the exception. Figure 4-3 illustrates the general form of the exception stack frame.

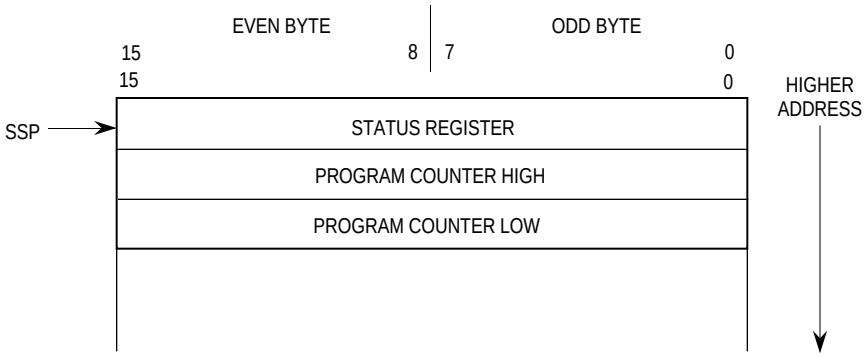
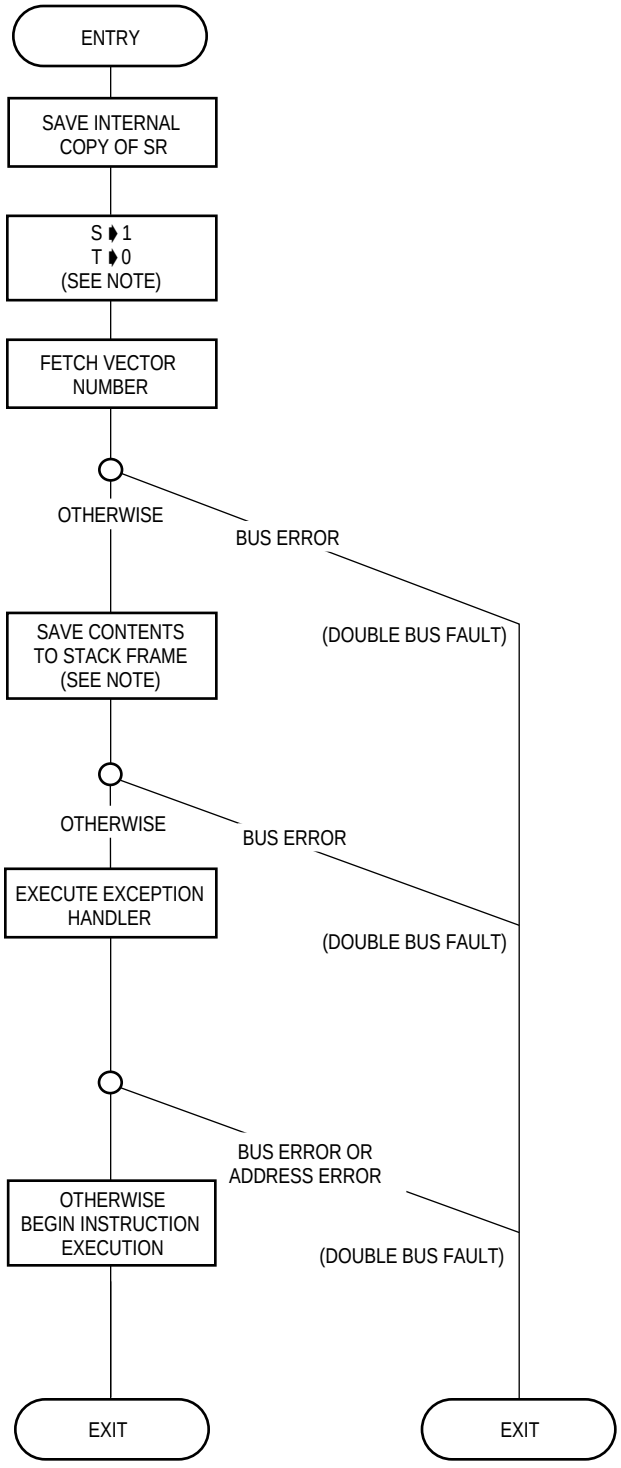


Figure 4-3. General Form of Exception Stack Frame

The last step initiates execution of the exception handler. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. The instruction at the address in the exception vector is fetched, and normal instruction decoding and execution is started.

Figure 4-4 shows a general exception processiong flowchart.



NOTE: These blocks vary for reset and interrupt exceptions.

Figure 4-4. General Exception Processing Flowchart

4.5.1 Exception Vectors

An exception vector is a memory location from which the processor fetches the address of a routine to handle an exception. Each exception type requires a handler routine and a unique vector. All exception vectors are two words in length (see Figure 4-5), except for the reset vector, which is four words long. All exception vectors reside in the supervisor data space, except for the reset vector, which is in the supervisor program space. A vector number is an 8-bit number that is multiplied by four to obtain the offset of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. For interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (see Figure 4-6) to the processor on data bus lines D7–D0.

The processor forms the vector offset by left-shifting the vector number two bit positions and zero-filling the upper-order bits to obtain a 32-bit long-word vector offset. In the EC000 core this offset is used as the absolute address to obtain the exception vector itself, which is illustrated in Figure 4-6.

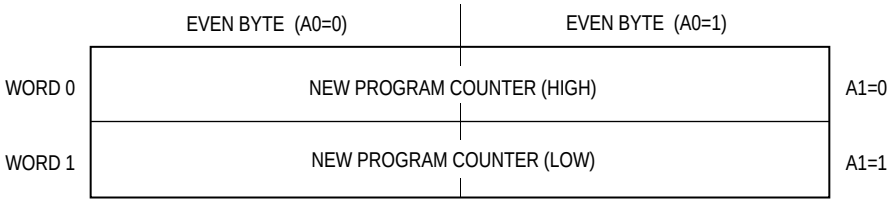


Figure 4-5. Exception Vector Format

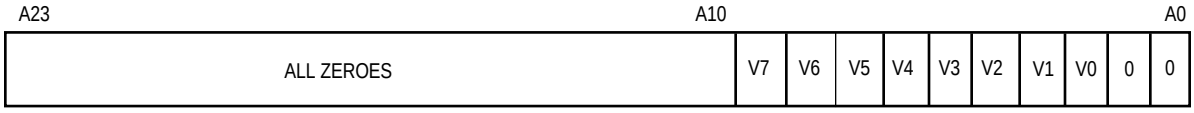


Figure 4-6. Address Translated from 8-Bit Vector Number

The actual address on the address bus is truncated to the number of address bits available on the bus of the particular implementation of the M68000 architecture. In the EC000 core, this is 24 address bits. The memory map for exception vectors is shown in Table 4-5.

The vector table is 512 words long (1024 bytes), starting at address 0 (decimal) and proceeding through address 1023 (decimal). The vector table provides 255 unique vectors, some of which are reserved for trap and other system function vectors. Of the 255, 192 are reserved for user interrupt vectors. However, the first 64 entries are not protected, so user interrupt vectors may overlap at the discretion of the systems designer.

4.6 PROCESSING OF SPECIFIC EXCEPTIONS

The exceptions are classified according to their sources, and each type is processed differently. The following paragraphs describe in detail the types of exceptions and the processing of each type. The exception vector assignments are listed in Table 4-5.

Table 4-5. Exception Vector Assignments

Vector Number	Address (Decimal)	Address (Hex)	Address Space	Vector Assignment
0	0	000	SP	Reset: Initial SSP[3]
1	4	004	SP	Reset: Initial PC[3]
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12[1]	48	030	SD	(Unassigned, Reserved)
13[1]	52	034	SD	(Unassigned, Reserved)
14[1]	56	038	SD	(Unassigned, Reserved)
15	60	03C	SD	Uninitialized Interrupt Vector[6]
16–23[1]	64–92	040–05C	SD	(Unassigned, Reserved)
24	96	060	SD	Spurious Interrupt by BERR[4]
25	100	064	SD	(Unassigned, Reserved)
26	104	068	SD	(Unassigned, Reserved)
27	108	06C	SD	(Unassigned, Reserved)
28	112	070	SD	(Unassigned, Reserved)
29	116	074	SD	(Unassigned, Reserved)
30	120	078	SD	(Unassigned, Reserved)
31	124	07C	SD	(Unassigned, Reserved)
32–47	128–188	080–0BC	SD	TRAP Instruction Vectors[5]
48–59[1]	192–236	0C0–0EE	SD	(Unassigned, Reserved)
60–63[1]	240–252	0F0–0FC	SD	Module Base Addr & System Configuration Regs
64–255	256–1020	100–3FC	SD	User Interrupt Vectors

NOTES:
1. In this table, SP = Supervisor Program Space, SD = Supervisor Data Space.
2. Vector numbers 12–14, 16–23, and 48–63 are reserved for future enhancements by Motorola (with vectors 60–63 being used by the MC68307 SIM). No user peripheral devices should be assigned these numbers.
3. Unlike the other vectors which only require two words, reset vector (0) requires four words and is located in the supervisor program space.
4. The EC000 core spurious interrupt vector is taken when there is no bus error indication during interrupt processing. This feature is not used by the MC68307. The MC68307 handles this condition separately; it outputs the spurious interrupt vector from the interrupt controller when it cannot provide an interrupt source.
5. TRAP #n uses vector number 32+n.
6. Vector number 15 is never used in the MC68307. The interrupt controller ensures that there is never an uninitialized interrupt.

4.6.1 Reset Exception

The reset exception corresponds to the highest exception level. The processing of the reset exception is performed for system initiation and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, and the trace state is forced off. The interrupt priority mask is set at level 7. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the SSP, neither the program counter nor the status register are saved. The address in the first two words of the reset exception vector is fetched as the initial SSP, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The initial program counter should point to the power-up/restart code.

The RESET instruction does not cause a reset exception; it asserts the $\overline{\text{RESET}}$ signal to reset external devices, which allows the software to reset the system to a known state and continue processing with the next instruction.

4.6.2 Interrupt Exceptions

NOTE

In the MC68307, all external and internal interrupt requests are controlled by the interrupt controller. The interrupt controller is always the interrupting device to the EC000 core, supplying the appropriate interrupt request level to the EC000 core via the internal $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$ lines. When the EC000 core performs an interrupt acknowledge, the interrupt controller provides the corresponding vector on the data bus and terminates the access with a $\overline{\text{DTACK}}$. This section considers the interrupt processing from the perspective of the EC000 core. It is worth noting that, from the MC68307's perspective, the EC000 $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$ lines are internal, and also the EC000 autovectoring is not supported, but is instead handled separately by the interrupt controller. Refer to **Section 5.1.4 Interrupt Processing** for further information on the interrupt operation.

Seven levels of interrupt priorities are provided, numbered from 1–7. Level 7 has the highest priority. Devices can be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. The status register contains a 3-bit mask indicating the current interrupt priority, and interrupts are inhibited for all priority levels less than or equal to the current priority. Priority level 7 is a special case. Level 7 interrupts cannot be inhibited by the interrupt priority mask, thus providing a non-maskable interrupt capability. An interrupt is generated each time the interrupt request level changes from some lower level to level 7. A level 7 interrupt may still be caused by the level comparison if the request level is a 7 and the processor priority is set to a lower level by an instruction.

An interrupt request is made to the processor by encoding the interrupt request level on the $\overline{IPL2}$–$\overline{IPL0}$; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but the requests are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction, and the interrupt exception processing is postponed until the priority of the pending interrupt becomes greater than the current processor priority.

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. A copy of the status register is saved; the privilege mode is set to supervisor mode; tracing is suppressed; and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device by executing an interrupt acknowledge cycle, which displays the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vector, the processor internally generates a vector number corresponding to the interrupt level number. If external logic indicates a bus error, the interrupt is considered spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing. The saved value of the program counter is the address of the instruction that would have been executed had the interrupt not been taken. The appropriate interrupt vector is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine.

4.6.3 Uninitialized Interrupt Exception

NOTE

The uninitialized interrupt vector of the EC000 core is never used in the MC68307, but is described here for completeness; the MC68307 handles all interrupt conditions separately in the interrupt controller (see **Section 5.1.4 Interrupt Processing**).

An interrupting device provides a EC000 core interrupt vector number and asserts data transfer acknowledge ($\overline{DTACK}$) or bus error ($\overline{BERR}$) during an interrupt acknowledge cycle by the EC000 core. If the vector register has not been initialized, the responding M68000 family peripheral provides vector number 15, the uninitialized interrupt vector. This response conforms to a uniform way to recover from a programming error.

4.6.4 Spurious Interrupt Exception

NOTE

The spurious interrupt vector of the EC000 core is never used in the MC68307, but is described here for completeness; the MC68307 handles all interrupt conditions separately in the interrupt controller (see **Section 5.1.4 Interrupt Processing**).

During the interrupt acknowledge cycle, if no device responds by asserting $\overline{DTACK}$, $\overline{BERR}$ should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by forming a short format exception stack and fetching

the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

4.6.5 Instruction Traps

Traps are exceptions caused by instructions; they occur when a processor recognizes an abnormal condition during instruction execution or when an instruction is executed that normally traps during execution.

Exception processing for traps is straightforward. The status register is copied; the supervisor mode is entered; and tracing is turned off. The vector number is internally generated; for the TRAP instruction, part of the vector number comes from the instruction itself. The program counter, and the copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the instruction following the instruction that generated the trap. Finally, instruction execution commences at the address in the exception vector.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a run-time error, which may be an arithmetic overflow or a subscript out of bounds. A signed divide (DIVS) or unsigned divide (DIVU) instruction forces an exception if a division operation is attempted with a divisor of zero.

4.6.6 Illegal and Unimplemented Instructions

Illegal instruction is the term used to refer to any of the word bit patterns that do not match the bit pattern of the first word of a legal processor instruction. If such an instruction is fetched, an illegal instruction exception occurs. Motorola reserves the right to define instructions using the opcodes of any of the illegal instructions. Three bit patterns always force an illegal instruction trap on all M68000 family-compatible microprocessors. The patterns are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for Motorola system products. The third pattern, $4AFC, is reserved for customer use (as the take illegal instruction trap (ILLEGAL) instruction).

Word patterns with bits 15–12 equaling 1010 or 1111 are distinguished as unimplemented instructions, and separate exception vectors are assigned to these patterns to permit efficient emulation. These separate vectors allow the operating system to emulate unimplemented instructions in software.

Exception processing for illegal instructions is similar to that for traps. After the instruction is fetched and decoding is attempted, the processor determines that execution of an illegal instruction is being attempted and starts exception processing. The exception stack frame is then pushed on the supervisor stack, and the illegal instruction vector is fetched.

4.6.7 Privilege Violations

To provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user mode causes an exception. The privileged instructions are as follows:

AND Immediate to SR	MOVE USP
EOR Immediate to SR	OR Immediate to SR
MOVE to SR	RESET
MOVE from SR	RTE
MOVEC	STOP
MOVES	

Exception processing for privilege violations is nearly identical to that for illegal instructions. After the instruction is fetched and decoded and the processor determines that a privilege violation is being attempted, the processor starts exception processing. The status register is copied; the supervisor mode is entered; and tracing is turned off. The vector number is generated to reference the privilege violation vector, and the current program counter and the copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the first word of the instruction causing the privilege violation. Finally, instruction execution commences at the address in the privilege violation exception vector.

4.6.8 Tracing

To aid in program development, the EC000 core includes a facility to allow tracing following each instruction. When tracing is enabled, an exception is forced after each instruction is executed. Thus, a debugging program can monitor the execution of the program under test.

The trace facility is controlled by the T-bit in the supervisor portion of the status register. If the T-bit is cleared (off), tracing is disabled and instruction execution proceeds from instruction to instruction as normal. If the T-bit is set (on) at the beginning of the execution of an instruction, a trace exception is generated after the instruction is completed. If the instruction is not executed because an interrupt is taken or because the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. During the execution of the instruction, if an exception is forced by that instruction, the exception processing for the instruction exception occurs before that of the trace exception.

As an extreme illustration of these rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First, the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

After the execution of the instruction is complete and before the start of the next instruction, exception processing for a trace begins. A copy is made of the status register. The transition to supervisor mode is made, and the T-bit of the status register is turned off, disabling further tracing. The vector number is generated to reference the trace exception vector, and the cur-

rent program counter and the copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the next instruction. Instruction execution commences at the address contained in the trace exception vector.

4.6.9 Bus Error

When a bus error exception occurs, the current bus cycle is aborted. The current processor activity, whether instruction or exception processing, is terminated, and the processor immediately begins exception processing.

Exception processing for a bus error follows the usual sequence of steps. The status register is copied, the supervisor mode is entered, and tracing is turned off. The vector number is generated to refer to the bus error vector. Since the processor is fetching the instruction or an operand when the error occurs, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are saved. The value saved for the program counter is advanced 2–10 bytes beyond the address of the first word of the instruction that made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. In addition to the usual information, the processor saves its internal copy of the first word of the instruction being processed and the address being accessed by the aborted bus cycle. Specific information about the access is also saved: type of access (read or write), processor activity (processing an instruction), and function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a group 2 exception; the processor is not processing an instruction if it is processing a group 0 or a group 1 exception. Figure 4-7 illustrates how this information is organized on the supervisor stack. If a bus error occurs during the last step of exception processing, while either reading the exception vector or fetching the instruction, the value of the program counter is the address of the exception vector. Although this information is not generally sufficient to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, an address error, or a reset, the processor halts and all processing ceases. This halt simplifies the detection of a catastrophic system failure, since the processor removes itself from the system to protect memory contents from erroneous accesses. Only an external reset operation can restart a halted processor.

4.6.10 Address Error

An address error exception occurs when the processor attempts to access a word or longword operand or an instruction at an odd address. An address error is similar to an internally generated bus error. The bus cycle is aborted, and the processor ceases current processing and begins exception processing. The exception processing sequence is the same as that for a bus error, including the information to be stacked, except that the vector number refers to the address error vector. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted.

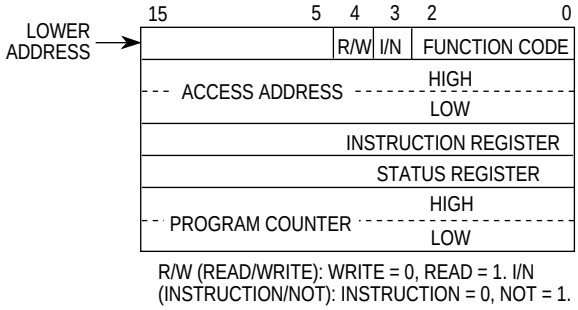


Figure 4-7. Supervisor Stack Order for Bus or Address Error Exception

4.6.11 Multiple Exceptions

When multiple exceptions occur simultaneously, they are processed according to a fixed priority. Table 4-6 lists the exceptions, grouped by characteristics, with group 0 as the highest priority. Within group 0, reset has highest priority, followed by address error and then bus error. Within group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, no priority relationship applies within group 2.

Table 4-6. Exception Grouping and Priority

Group	Exception	Processing
0	Reset, Address Error, and Bus Error	Exception processing begins within two clock cycles.
1	Trace, Interrupt, Illegal, and Privilege	Exception processing begins before the next instruction.
2	TRAP, TRAPV, CHK, and DIV	Exception processing is started by normal instruction execution.

The priority relationship between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit in the status register (SR) is asserted, the trace exception has priority and is processed first. Before instruction execution resumes, however, the interrupt exception is also processed, and instruction processing finally commences in the interrupt handler routine. As a general rule, the lower the priority of an exception, the sooner the handler routine for that exception executes. This rule does not apply to the reset exception; its handler is executed first even though it has the highest priority, because the reset operation clears all other exceptions.

SECTION 5
SYSTEM INTEGRATION MODULE

The MC68307 system integration module (SIM) consists of several functions that control the system start-up, initialization, configuration, and the external bus with a minimum number of external devices. The SIM contains the following functions:

- System Configuration
- Oscillator and Clock Dividers
- Reset Control, Power-Down Mode Control
- Chip Selects and Wait States
- External Bus Interfaces, M68000 and 8051-Compatible
- Parallel Input/Output Lines with Interrupt Capability
- Interrupt Configuration/Response

The MC68307 SIM is similar to the system integration logic found in other Motorola integrated processors, the M68000 family, although each is tailored to the specific needs of the part and its intended application.

5.1 MODULE OPERATION

The various internal functional blocks of the SIM are described here, with a description of operation of each, along with methods and recommendations for programming the register locations that configure the MC68307 to suit the user's target system.

5.1.1 MC68307 System Configuration

The MC68307 system configuration logic consists of a module base address register (MBAR), and a system control register (SCR) which together allow the user to configure operation of the following functions:

- Base Address and Address Space of Internal Peripheral Registers
- Low-Power (Standby) Modes
- Hardware Watchdog Modes for System Protection
- 8051-Compatible Bus Enable or Disable
- Peripheral Chip Selects Enable or Disable
- Data Bus Size Control for Chip Selected Address Ranges

5.1.1.1 MODULE BASE ADDRESS REGISTER OPERATION. The MBAR, must be programmed upon cold reset in order to specify the base address of the various registers which comprise the SIM and the other on-chip peripherals. The format of this register is described later in this section, along with a description of the complete memory map of the EC000 internal registers and peripherals.

The on-chip peripherals require a reserved 4096-byte block of address space for their registers. This block location is determined by writing the intended base address to the MBAR in supervisor data space. The MBAR is an on-chip read/write register which is located at an unused vector entry in the MC68307 vector table. The address of the MBAR entry within the vector table is $0000F2; it is a 16-bit value. For further details, refer to **Section 5.2.1 System Configuration and Protection Registers**.

The module base address and on-chip peripherals address decode logic block diagram is shown here in Figure 5-1.

After a cold reset, the on-chip peripheral base address is undefined and it is not possible to access the on-chip peripherals at any address until the MBAR is written. The MBAR and the SCR can always be accessed at their fixed addresses.

Do not assign other devices on the system bus an address that falls within the address range of the on-chip peripherals defined by the MBAR. Bus contention could result if this is done inadvertently. If this happens, the address decode conflict (ADC) status bit in the SCR is set. This can cause a $\overline{\text{BERR}}$ to be generated, if the address decode conflict enable (ADCE) bit in the SCR is set.

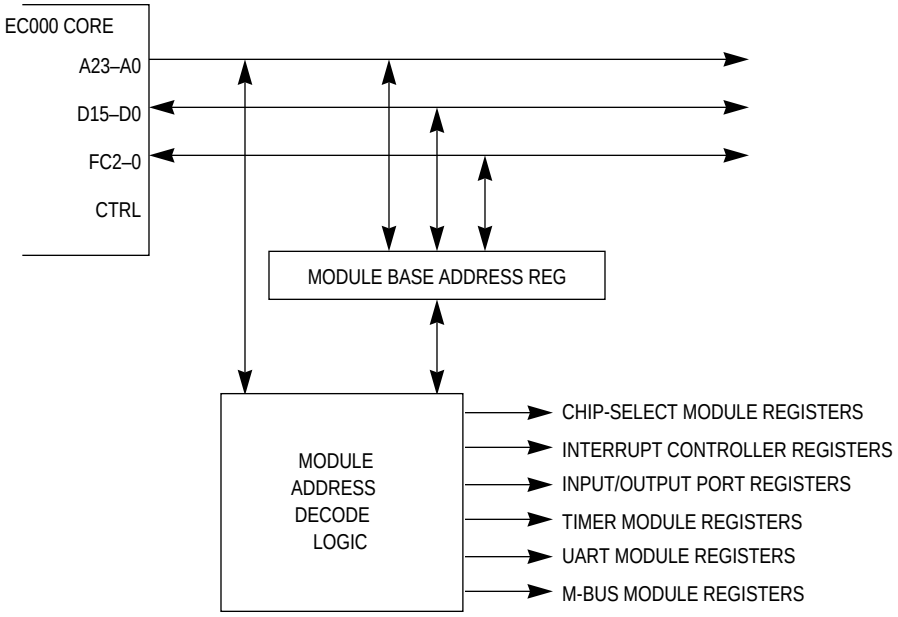


Figure 5-1. Module Base Address, Decode Logic

NOTE

The MBAR and SCR registers are internally reset only when a cold reset occurs by the simultaneous assertion of $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$, assertion of $\overline{\text{RSTIN}}$, power-on reset or software watchdog timeout. They are unaffected by a wake-up from power-down mode. The chip select ($\overline{\text{CSx}}$) lines are not asserted on accesses to the MBAR and SCR locations. Thus, it is very convenient to use $\overline{\text{CSx}}$ lines to select external ROM/RAM that overlaps or encloses the MBAR and SCR register locations ($0000F0-$0000FF), since no potential bus contention can occur. If externally-decoded chip selects are used, then contention cannot be ignored.

5.1.1.2 SYSTEM CONTROL REGISTER FUNCTIONS. The SCR allows various settings to be made which influence the operation of the system, for example, power-down and oscillator control logic, the bus interface, and the hardware watchdog protection. It also includes status bits which allow exception handler code to monitor the cause of exceptions and resets.

Most of the functions controlled by these register bits are described in detail elsewhere; they include:

CD	Enable divided frequency to EC000 core
LPEN	Low-power enable (to select low-power sleep mode)
CKD	Clock-output disable (to inhibit CLKOUT pin and therefore save power)
LPCD2–0	Processor clock speed during low-power modes
UACD	Clock disable for UART peripheral module
MBCD	Clock disable for M-bus peripheral module
TMCD	Clock disable for timer peripheral module
UACW	Auto wakeup of UART clock if data received
BUSW0	Chip select 0 bus width 8-bit or 16-bit
BUSW1	Chip select 1 bus width 8-bit or 16-bit
BUSW2	Chip select 2 bus width 8-bit or 16-bit
BUSW3	Chip select 3 bus width 8-bit or 16-bit
EPCS	Chip select 2A-2D peripheral chip select mode enable
E8051	Chip select 3 8051-compatible bus interface enable
ADCE	Address decode conflict enable to cause $\overline{\text{BERR}}$
WPVE	Write protect violation enable to cause $\overline{\text{BERR}}$
HWDE	Hardware watchdog enable
HW2–0	Hardware watchdog timeout value setting

Status bits included in the SCR are:

RS1–0	Reset source register
ADC	Address decode conflict status bit
WPV	Write protect violation status bit
HWT	Hardware timeout status bit

5.1.1.3 SYSTEM PROTECTION FUNCTIONS. The facilities provided for system protection are the hardware watchdog (bus monitor) and the software watchdog timer.

The hardware watchdog provides a bus monitor which causes an internal bus error ($\overline{\text{BERR}}$ assertion) when a bus cycle is not terminated by $\overline{\text{DTACK}}$ after a programmable number of clock cycles has elapsed. The hardware watchdog timeout (HWT) status bit in the SCR is also set, so that a bus error exception handler can determine the cause of the bus error.

The hardware watchdog logic consists of a 10-bit down-counter and a 4-bit fixed prescaler. When enabled, the watchdog timer commences counting clock cycles as $\overline{\text{AS}}$ is asserted (for internal or external bus masters). The count is terminated normally by the negation of $\overline{\text{AS}}$; however, if the count reaches zero before $\overline{\text{AS}}$ is negated, $\overline{\text{BERR}}$ is asserted until $\overline{\text{AS}}$ is negated. The hardware watchdog logic uses four control bits and one status bit in the SCR.

The effective range of the bus timeout is from 128 clock cycles to 16384 clock cycles (at 16MHz, from 8µs to 1ms).

For operation of the software watchdog timer, refer to **Section 6 Dual Timer Module**.

5.1.2 Chip Select and Wait-State Logic

The MC68307 provides a set of four programmable chip-select signals. Each has a common set of features and some have particular special features associated with them. These features were described in terms of the MC68307 input/output pins in **Section 2 Signal Description**, but will be described again here in detail. For each memory area the user may also define an internally generated cycle termination signal ($\overline{\text{DTACK}}$) with programmable number of wait-states. This feature eliminates board space that would otherwise be necessary for cycle termination logic.

The four chip selects allow up to four different classes of memory to be used in a system without external decode or wait-state generation logic. For example, a typical configuration could be a 8-bit EPROM, a fast 16-bit SRAM, up to four simple I/O peripherals, and a non-volatile RAM with an 8051-compatible interface.

The chip select block diagram is shown in Figure 5-2.

The basic chip select model allows the chip select output signal to assert in response to an address match. The signals are asserted externally shortly after $\overline{\text{AS}}$ goes low. The address match is described in terms of a base address and an address mask. Thus the size in bytes of the matching block must be a power of 2, and the base address must be an integer multiple of this size. Thus an 8-Kbyte block size must begin on an 8-Kbyte boundary, and a 64-Kbyte block size can only begin on a 64-Kbyte boundary, etc.

The minimum resolution of block size, and hence base address, is any multiple of 8192, because only address lines A23 down to A13 are compared or masked. Each chip select can be enabled or disabled independently of the others, and the registers are read-write so that the values programmed can be read back.

For a given chip select block, the user may also choose whether the chip select allows read-only, write-only, or read/write accesses, whether the chip select should match only one function code value or all values, whether a $\overline{\text{DTACK}}$ is automatically generated for this chip select and after how many wait states (from zero to six).

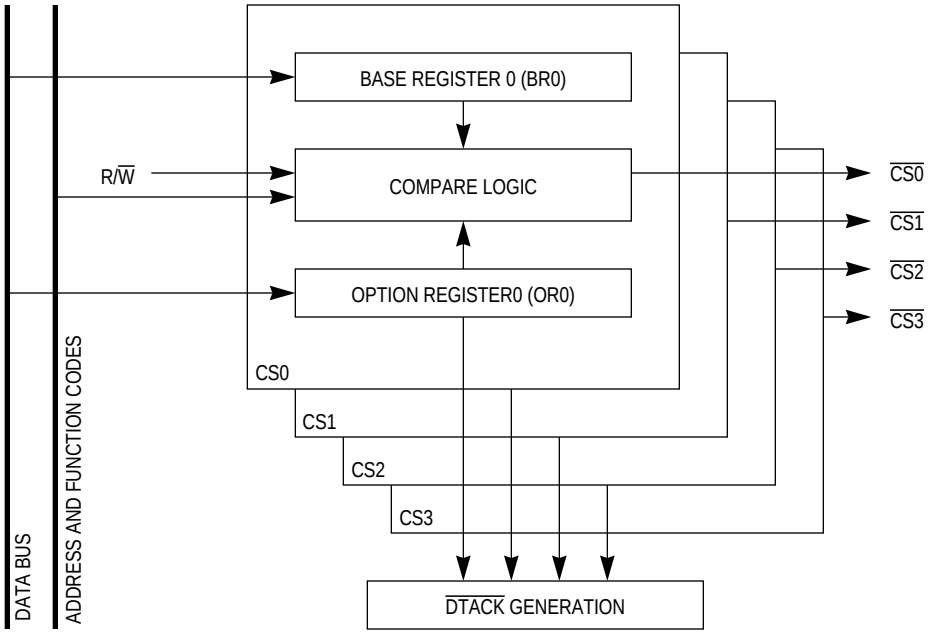


Figure 5-2. Chip-Select Block Diagram

5.1.2.1 PROGRAMMABLE DATA-BUS SIZE. Each of the chip selects include the facility of a data-bus port sizing extension to the basic M68000 bus which allows the system designer to mix 16-bit and 8-bit contiguous address memory devices (RAM, ROM) on a 16-bit data bus system. If the EC000 core executes a 16-bit data transfer instruction, then two 8-bit bus cycles appear, using the external M68000 data bus upper half (D15–D8) only, the least significant bit of address (A0) incrementing automatically from one to the next. A0 should be ignored in 16-bit data-bus cycles, even if only the upper or lower byte is being read/written.

Note that a 16-bit data bus is always used internally for access to peripheral registers, regardless of any mode settings for the external bus. Where peripheral registers are 16-bits wide, they can be read or written in one bus cycle only, eliminating possible conflicts and reading of inaccurate values where 16-bit-wide register contents are volatile (timer counter registers, for example) or where the whole 16-bit value affects some aspect of system operation (chip select base address, for example).

Where internal peripherals are 8-bits wide, e.g., the MC68681-compatible UART, they are accessed at every alternate (odd) address. It is recommended that any external peripheral that needs only an 8-bit data-bus interface but does not require contiguous address locations, uses a chip select configured as 16-bit data-bus width, and connects to D7–D0. This balances more evenly the load on the two halves of the data bus in an 8-bit system.

The default for each chip select is a 16-bit data-bus width. The BUSWx bits in the SCR enable 8-bit data-bus width for each of the four chip select ranges. The initial bus width for chip select 0 is selected by placing a logic 0 or 1 on the BUSW pin at reset, to specify 8-bit or 16-bit wide data bus respectively. This allows a boot EPROM of either data bus width to be used in any given system.

All external accesses which do not match one of the chip select address ranges use the value of the EBUSW bit in the SCR to determine their data bus width.

NOTE

If byte operations are performed where 8-bit bus mode is used, there is no difference in throughput compared to 16-bit bus mode. However, if 32-bit long word or 16-bit word operations are used over the 8-bit data bus (including all instruction fetches), the performance is exactly half that of the equivalent speed 16-bit-wide data bus.

5.1.2.2 PERIPHERAL CHIP SELECTS. Chip select 2 ($\overline{CS2}$ signal) features a peripheral chip select (PCS) mode, selected by setting the enable peripheral chip select (EPCS) bit in the SCR. If this chip select is programmed with a block size of 64 Kbytes, and the EPCS bit in the SCR set, then the $\overline{CS2}$/$\overline{CS2A}$ pin, along with $\overline{CS2B}$, $\overline{CS2C}$, and $\overline{CS2D}$ (which are multiplexed with port A input/output lines) function as four peripheral chip selects, each gated to select a particular 16-Kbyte block within the programmed 64-Kbyte range.

For example, if the base address is programmed as $100000 with size $10000 (64K), and the PCS mode is enabled, then the blocks are selected according to Table 5-1:

Table 5-1. Address Block Selection in Peripheral Chip Select Mode

Chip Select Pin	Address Block		A15	A14
	Start Address	End Address		
$\overline{CS2A}$	$100000	$103FFF	0	0
$\overline{CS2B}$	$104000	$107FFF	0	1
$\overline{CS2C}$	$108000	$10BFFF	1	0
$\overline{CS2D}$	$10C000	$10FFFF	1	1

This feature is ideal for miscellaneous peripheral devices in the address space, even if they only require one or two bytes per peripheral. If this feature is not enabled (refer to **Section 5.2.1.2 System Control Register (SCR)**), then the $\overline{CS2}$/$\overline{CS2A}$ pin functions as $\overline{CS2}$, a general-purpose programmable M68000-bus chip select, and the $\overline{CS2B}$, $\overline{CS2C}$ and $\overline{CS2D}$ signals are never asserted.

NOTE

Port A general-purpose I/O lines have programmable control over their function on a bit by bit basis, via the port A control register (PACNT). For example it is possible to use only $\overline{CS2A}$ and $\overline{CS2B}$, leaving the port A lines which would have been used by $\overline{CS2C}$ and $\overline{CS2D}$, to function as general-purpose I/O.

5.1.2.3 8051-COMPATIBLE BUS CHIP SELECT. Chip select 3 ($\overline{CS3}$ signal) can be used to define the addressing range of the 8051-compatible bus mode, when this mode is enabled in the SCR E8051 bit. Otherwise (if the 8051-compatible bus mode is not used) chip select 3 is available for any general purpose memory or peripheral. In this case, the 8051-compatible bus read and write strobes ($\overline{RD}$ and $\overline{WR}$), and the address latch enable (ALE) signal are always negated. This bus always uses an 8-bit data-bus width, and so the BUSW3 bit in the SCR should be set along with the E8051 bit.

5.1.2.4 GLOBAL CHIP SELECT OPERATION (RESET DEFAULTS). Chip select 0 is initialized from cold reset to assert in response to any address in the first 8K bytes of memory space, in order to ensure a chip select to the boot ROM or EPROM, to fetch the reset vector and execute the initialization code, which should set up the module base address and the four chip select ranges early on in that initialization sequence.

The data bus port size for CS0 on reset, and hence the data width of the boot ROM device, are programmed by placing logic 0 or 1 on the BUSW pin during reset, for 8-bit and 16-bit wide data bus respectively.

The other 3 chip selects are initialized to be invalid, and so do not assert until they are programmed.

5.1.2.5 OVERLAP IN CHIP SELECT RANGES. The user should not normally program more than one chip select line to the same area. If this accidentally occurs, only one chip select line is driven because of internal line priorities. $\overline{CS0}$ has the highest priority, and $\overline{CS3}$ the lowest. The address compare logic sets the address decode conflict (ADC) status bit in the SCR, and also generates a bus error (or $\overline{BERR}$ is asserted) if the address decode conflict enable (ADCE) bit was set by the user in the SCR.

$\overline{BERR}$ is never asserted on write accesses to the chip select registers.

If one chip select is programmed to be read-only, and another is programmed to be write-only, then there is no overlap conflict between these two chip selects, and the address decode conflict (ADC) status bit in the SCR is not set.

When the CPU attempts to write to a read-only location, as programmed by the user when setting up the chip selects, the chip select logic sets the write protect violation (WPV) bit in the SCR, and will also generate $\overline{BERR}$ if the write protect violation enable (WPVE) bit is set in the SCR. The $\overline{CSx}$ line is not asserted.

NOTE

The chip select logic is reset only on cold reset (assertion of $\overline{RESET}$ and $\overline{HALT}$, or $\overline{RSTIN}$). The chip select ($\overline{CSx}$) lines are never asserted on accesses to the MBAR and SCR locations. Thus, it is very convenient to use $\overline{CSx}$ lines to select external ROM/RAM that overlaps or encloses the MBAR and SCR

register locations ($0000F0–$0000FF), since no potential bus contention can occur.

The chip select logic does not allow an address match during interrupt acknowledge (function code 7) cycles.

The programming of the chip select module is discussed in the next chapter. For further information, refer to **Section 5.2.2 Chip Select Registers**.

5.1.3 External Bus Interface Logic

This logic controls the interaction of the EC000 core processor, the on-chip M68000 peripheral bus, the external M68000 and 8051-compatible buses, in conjunction with the programmable chip select logic and the various user settings in the system configuration and protection logic. A block diagram of this logic is shown in Figure 5-3.

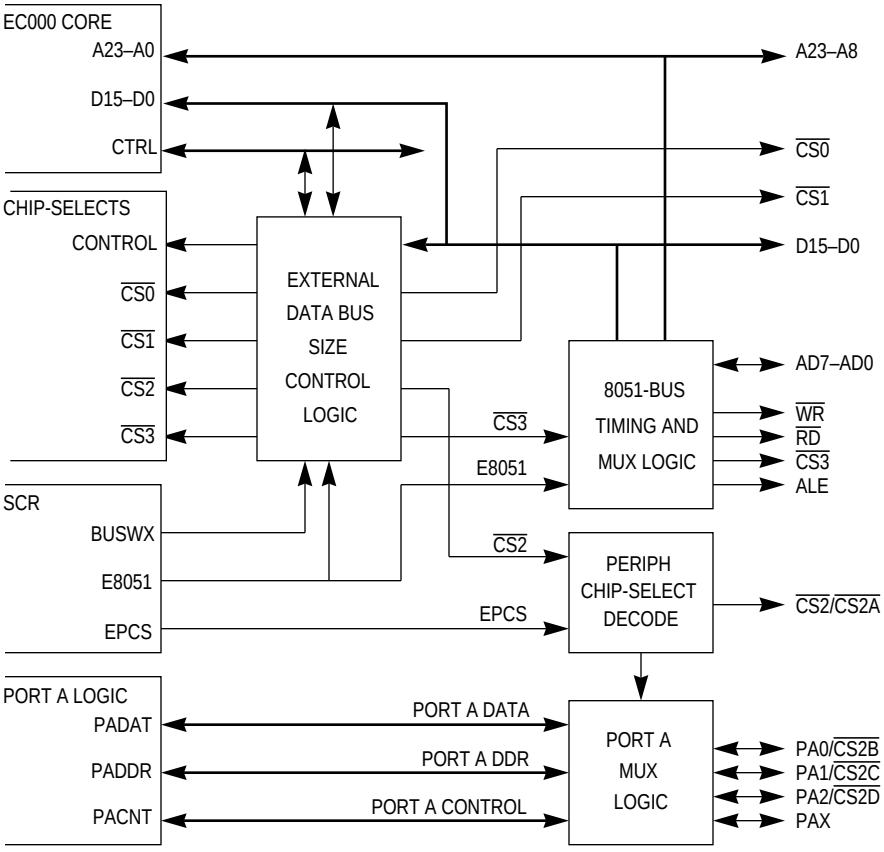


Figure 5-3. External Bus Interface Logic

5.1.3.1 M68000 BUS INTERFACE. The M68000 bus interface adds bus-sizing functionality (8-bit or 16-bit data bus) to the basic M68000 bus of the EC000 core processor. The external bus interface logic links the data bus multiplexers with the programmable chip selects and SCR bits, and co-ordinates an 8-bit or 16-bit bus transfer. The BUSWx bits in the SCR define the current data-bus width (port size) for each of the four chip selects. On cold reset, the

default state of BUSW0 reflects the value of the BUSW external pin (0 = 8-bit data bus, 1 = 16-bit data bus), and the default state of the other BUSWx bits is for a 16-bit port size.

When the data-bus sizing logic is used to configure a chip select as an 8-bit port select, any 16-bit bus cycles from the processor appears on the external pins as two 8-bit read or write cycles with normal M68000 timings on all signals and the addition of a valid A0 signal. D15–D8 is always used for bus cycles of this kind.

NOTE

If byte operations are performed where 8-bit bus mode is used, there is no difference in throughput compared to 16-bit bus mode. However, if 32-bit long word or 16-bit word operations are used over the 8-bit data bus (including all instruction fetches), the performance is exactly half that of the equivalent speed 16-bit-wide data bus.

5.1.3.2 8051-COMPATIBLE BUS INTERFACE. The 8051-compatible bus interface contains logic to multiplex the low eight address lines (A7–A0) and the low eight data lines (D7–D0) onto the same pins. $\overline{CS3}$ is used to select the memory area assigned to the 8051-compatible bus, it controls the external bus interface logic, triggering the address latch enable signal, and gating the $\overline{RD}$ and $\overline{WR}$ signals. Refer to **Section 5.1.2 Chip Select and Wait-State Logic** and **Section 5.2.2 Chip Select Registers** for details of how to configure the chip select logic for this interface. Also refer to **Section 5.2.1 System Configuration and Protection Registers** for details of how to configure the MC68307 pins for the 8051-compatible bus.

The 8051-compatible interface uses the same 8-bit data bus sizing scheme as offered with the M68000 chip selects. As such, there is no restriction on the data operations which can be used—both byte, word and long transfers can be initiated by the processor to an 8051-compatible device which can have contiguous byte locations.

5.1.3.3 PORT A, PORT B GENERAL-PURPOSE I/O PORTS. The MC68307 supports two general purpose I/O ports, port A and port B, whose pins can be configured as general-purpose input/output pins or dedicated peripheral interface pins for the on-chip modules (UART, M-bus, timer, interrupts, peripheral chip selects). Port A is an 8-bit I/O port. Port B is a 16-bit I/O port.

Each of the 8 port A and 16 port B pins are independently configured as a general-purpose I/O pin if the corresponding bit in the port A or B control register (PACNT, PBCNT) is cleared. Port pins are configured as dedicated on-chip peripheral pins if the corresponding PACNT or PBCNT bit is set. Refer to **Section 5.2.3 External Bus Interface Control Registers** for details of programming the general-purpose ports.

For each port pin, when acting as a general-purpose I/O pin, the signal direction for that pin is determined by the corresponding control bit in the port A or B data direction register (PADDR, PBDDR). The port I/O pin is configured as an input if the corresponding PADDR or PBDDR bit is cleared; it is configured as an output if the corresponding PADDR or PBDDR bit is set. All PADDR, PBDDR, PACNT, and PBCNT bits are cleared on cold reset,

configuring all port A and B pins as general-purpose inputs. (Note that the port pins do not have internal pullup resistors).

When a port pin is used as general-purpose I/O, it may be accessed through the port A or B data register (PADAT, PBDAT). Data written to the PADAT or PBDAT is stored in an output latch.

- If a port pin is configured as an output, the output latch data is gated onto the port pin. In this case, when the PADAT or PBDAT is read, the contents of the output latch data associated with the output port pin are read.
- If a port pin is configured as an input, data written to PADAT or PBDAT is still stored in the output latch, but is prevented from reaching the port pin. In this case, when PADAT or PBDAT is read, the current state of the port pin is read.

If port pins are selected as dedicated on-chip peripheral pins, the corresponding bit in the PADDR or PBDDR is ignored, and the actual direction of the pin is determined by the operating mode of the on-chip peripheral. In this case, the PADAT or PBDAT register contains, for those bit positions, the current state of the peripheral's input pin or output driver.

The dedicated functions for port A input/output lines are the extra peripheral chip select outputs ($\overline{CS2B}$, $\overline{CS2C}$, and $\overline{CS2D}$) for simple external peripherals (PA2–PA0), the timer output signals TOUT1 and TOUT2 (PA4 and PA3), and the bus arbitration signals $\overline{BR}$, $\overline{BG}$, and $\overline{BGACK}$ (PA7–PA5).

Certain port pins may be selected as general-purpose I/O pins, even when other pins related to the same on-chip peripheral are used as dedicated pins. For example, a system which requires a UART interface using TxD and RxD signals only, and not requiring $\overline{RTS}$ or $\overline{CTS}$, is free to program $\overline{RTS}$/PB4 and $\overline{CTS}$/PB5 as general-purpose I/O. What the peripheral now receives as input, given that some of its pins have been reassigned, is shown in Table 5-2. If an input pin to an I/O peripheral is used as a general-purpose I/O pin, then the actual input to the peripheral is automatically connected internally to V_{DD} or GND, based on the pin's function. This does not affect the operation of the port I/O pins in their general-purpose I/O function. Note also that even if all the pins for a particular peripheral are configured as general-purpose input/output, the peripheral still operates as normal, although this is only useful in the case of the timer module.

The MC68307 port A pins which have alternate functions are shown in Table 5-2.

Table 5-2. Port A Pin Functions

PACNT Bit = 0 Pin Function	PACNT Bit = 1 Pin Function	Default Input to On-Chip Peripheral
PA0	CS2B	(Output)
PA1	CS2C	(Output)
PA2	CS2D	(Output)
PA3	TOUT1	(Output)
PA4	TOUT2	(Output)
PA5	BR	1
PA6	BG	(Output)
PA7	BGACK	1

The MC68307 port B pins that have alternate functions are shown in Table 5-3.

Table 5-3. Port B Pin Functions

PBCNT Bit = 0 Pin Function	PBCNT Bit = 1 Pin Function	Default Input to On-Chip Peripheral
PB0	SCL	1
PB1	SDA	1
PB2	TxD	(Output)
PB3	RxD	1
PB4	RTS	(Output)
PB5	CTS	1
PB6	TIN1	1
PB7	TIN2	1
PB8	INT1	1
PB9	INT2	1
PB10	INT3	1
PB11	INT4	1
PB12	INT5	1
PB13	INT6	1
PB14	INT7	1
PB15	INT8	1

The high eight lines of port B (PB15–PB8) have as their dedicated function a latched interrupt capability. Each of these pins, when configured as a dedicated interrupt input in the port B control register (PBCNT), functions as an active-low input to a latch in the interrupt controller logic (refer to **Section 5.1.4.1 Interrupt Controller Logic**). As before, even if these pins are configured for their dedicated function, their current state may still be read by software as if they were general-purpose input signals.

5.1.4 Interrupt Processing

Interrupt processing on the MC68307 involves four steps. A typical sequence is as follows:

1. The interrupt controller on the MC68307 collects interrupt events from on and off-chip peripherals, prioritizes them, and presents the highest priority request to the EC000 core processor.
2. The EC000 core processor responds to the interrupt request by executing an interrupt acknowledge bus cycle after the completion of the current instruction.
3. The interrupt controller recognizes the interrupt acknowledge cycle and places the interrupt vector for that interrupt request onto the EC000 core processor bus.
4. The EC000 core processor reads the vector, reads the address of the interrupt handler in the exception vector table, and then begins execution at that address.

Steps 2 and 4 are the responsibility of the EC000 core processor on the MC68307, whereas steps 1 and 3 are the responsibility of the interrupt controller on the MC68307.

External devices are forbidden from responding to IACK cycles with a vector, this is always done by the interrupt controller. No FC2–FC0 signals are available for decode.

The EC000 core processor is not modified for use on the MC68307, thus steps 2 and 4 operate exactly as they would on the M68000 devices. In step 2, the EC000 processor status register (SR) is available to mask interrupts globally or to determine which priority levels can currently generate interrupts. Also in step 2, the interrupt acknowledge cycle is executed.

The interrupt acknowledge cycle carries out a M68000 bus read cycle except that FC2–FC0 are encoded as 111, A3–A1 are encoded with the interrupt priority level (1–7, with 7 (i.e., 111) being the highest), and A19–A16 are driven high. This cycle is visible externally, but no chip selects are asserted.

In step 4, the EC000 core processor reads the vector number, multiplies it by 4 to get the vector address, fetches a 4-byte program address from that vector address, and then jumps to that 4-byte address. That 4-byte address is the location of the first instruction in the interrupt handler.

Steps 1 and 3 are the responsibility of the interrupt controller on the MC68307. Here a number of configuration options are available. For instance, for step 1 (interrupt generation), all interrupt sources have programmable interrupt priority levels (IPL). In step 3 (vector response), a block of vectors can be allocated to the interrupt sources under program control. These and other interrupt controller options are introduced in the following paragraphs.

5.1.4.1 INTERRUPT CONTROLLER LOGIC. This block of logic coordinates all the interrupt sources on the MC68307, gathering interrupt request signals from both the on-chip peripheral modules and external inputs, prioritizing them and performing programmed IPL requests to the EC000 core processor. When the EC000 core processor responds to a request with an interrupt acknowledge cycle, as is standard in M68000 implementations, the interrupt controller logic forwards the correct vector depending on the original source of the interrupt. Software can clear pending interrupts from any source via the registers in the interrupt controller logic, and can program the location of the block of vectors used for interrupt sources via the programmable interrupt vector register (PIVR).

For an external interrupt interface, the interrupt controller logic provides two distinct facilities. First is the nonmaskable interrupt input on the $\overline{\text{IRQ7}}$ pin, which always causes an interrupt priority level 7 request to the EC000 core processor. Assuming no other source is programmed as a level 7 source, this input always obtains the immediate attention of the core. For an interrupt to be successfully processed, RAM must be available for the stack, and often this RAM is selected by one of the programmable chip selects. So upon system startup there is a brief period where RAM is not available for the stack. To ensure no problems resulting from interrupts (particularly $\overline{\text{IRQ7}}$) during this period, there is an interlock which prevents any interrupt from reaching the EC000 core processor until the first write cycle to the PIVR. The user should ensure that both RAM chip selects and the system stack are set up prior to this write operation.

The second of the external input methods is an 8-channel latched interrupt port, multiplexed with the port B input/output pins. Each of the 8 inputs can be programmed with an IPL, and each can have any pending interrupts cleared independently of the others.

The interrupt controller includes daisy-chaining functions in order to avoid contention when the EC000 core processor issues an interrupt acknowledge cycle. So if more than one interrupt source has the same IPL, they are daisy-chained in the following priority scheme:

$\overline{\text{IRQ7}}$ Input	Highest Priority
$\overline{\text{INT1}}$–$\overline{\text{INT8}}$ Inputs	•
Timer 1Interrupt	•
Timer 2 Interrupt	•
UART Interrupt	•
M-Bus Interrupt	Lowest Priority

The priority within the eight latched interrupt inputs is that $\overline{\text{INT1}}$ is the highest, and $\overline{\text{INT8}}$ is the lowest.

The block diagram of the interrupt controller is shown in Figure 5-4.

5.1.4.2 INTERRUPT VECTOR GENERATION. Pending interrupts are presented to the EC000 core processor in order of priority. The core responds to an interrupt request by initiating an interrupt acknowledge cycle to receive a vector number, which allows the core to locate the interrupt's service routine. The interrupt controller always provides the vector corresponding to the highest priority, unmasked, pending interrupt with the IPL priority equal to that echoed on A3–A1 during the IACK cycle.

The following procedure is used. The four most significant bits of the interrupt vector are programmed by the user in the PIVR. These four bits are concatenated with four bits generated by the interrupt controller to provide an 8-bit vector number to the core. The interrupt controller's encoding of the four low-order bits of the interrupt vector is shown in Table 5-4. An example vector calculation is shown following Table 5-4.

When the core initiates an interrupt acknowledge cycle for an interrupt and there is no interrupt pending, the interrupt controller encodes the error code binary 0000 onto the four low-order bits of the interrupt vector to indicate a spurious interrupt. As the interrupt controller responds to all interrupt acknowledge cycles, there will never be a bus error during such cycles, and the EC000 core vector number 24 (spurious interrupt by bus error) will never be used.

Figure 5-4 shows the interrupt controller logic block diagram.

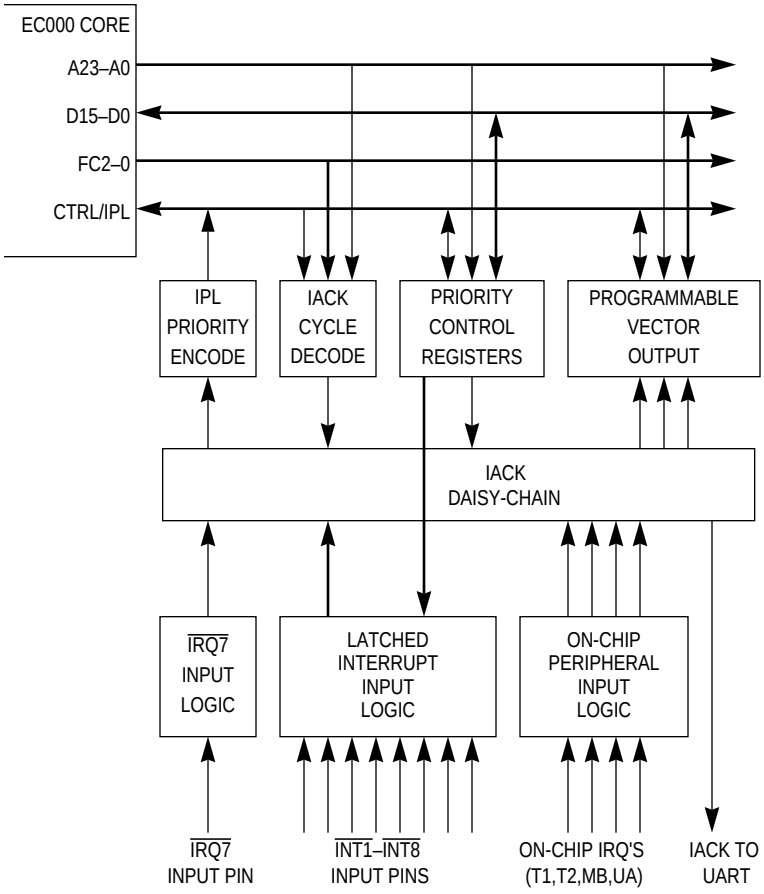


Figure 5-4. Interrupt Controller Logic Block Diagram

Table 5-4. Interrupt Vector Response

Vector	Interrupt Source	Priority if Same IPL Programmed
xxxx0000	Spurious Interrupt	
xxxx0001	$\overline{\text{IRQ7}}$ Input	(Highest)
xxxx0010	$\overline{\text{INT1}}$ Input	•
xxxx0011	$\overline{\text{INT2}}$ Input	•
xxxx0100	$\overline{\text{INT3}}$ Input	•
xxxx0101	$\overline{\text{INT4}}$ Input	•
xxxx0110	$\overline{\text{INT5}}$ Input	•
xxxx0111	$\overline{\text{INT6}}$ Input	•
xxxx1000	$\overline{\text{INT7}}$ Input	•
xxxx1001	$\overline{\text{INT8}}$ Input	•
xxxx1010	Timer 1	•
xxxx1011	Timer 2	•
xxxx1100	(Reserved for UART)*	•
xxxx1101	M-Bus Module	(Lowest)

1. Formulate 8-bit vector (source is TIMER 2):*

V7–V4				4-Bit Vector			
1	0	1	0	1	0	1	1

V7–V5 programmed by software in the programmable interrupt vector register 4-bit vector from Table 5-4.

2. Multiply by 4 to get address:

   1 0 1 0 1 0 1 1 0 0 = $2AC

   Note that $2AC is in the user interrupt vector area of the exception vector table. V7–V4 was purposely chosen to cause this.

3. Read 32-bit value at $2ac and jump:

$2AC	0007
$2AE	0302

Interrupt handler begins at $070302 (24-bit addresses are used on the MC68307)

* The UART module handles its own interrupt vector separately, it can be programmed to provide any vector by placing a value into the UART interrupt vector register (UIVR). For consistency with the other interrupt sources, this value can be programmed as xxxx1100.

5.1.4.3 $\overline{\text{IRQ7}}$ NON-MASKABLE INTERRUPT. The $\overline{\text{IRQ7}}$ input functions as a nonmaskable interrupt (NMI) which always generates a level 7 interrupt to the EC000 core processor. Priority level 7 cannot be disabled by the interrupt priority mask in the SR of the EC000 core processor.

The interrupt controller logic passes an interrupt from the $\overline{\text{IRQ7}}$ signal to the processor. When the processor responds with an interrupt acknowledge cycle for level 7, the interrupt controller issues the appropriate vector.

The $\overline{\text{IRQ7}}$ input is edge sensitive and subject to two clock falling edges of synchronization before being considered valid. If it is still asserted when a level 7 interrupt handler routine exits (thus bringing the processor priority level below 7) then it is ignored until it negates for one clock period before the next valid assertion. If the software needs to read the state of the $\overline{\text{IRQ7}}$ signal then the signal can be connected to an unused general-purpose input/ output pin, configured as an input.

An interlock is provided which prevents any interrupt including IRQ7 from reaching the EC000 core processor until the PIVR is first written with a vector range. This allows the user to ensure safety upon system startup before essential resources (e.g., chip selects for RAM which contain stack) have been set up.

5.1.4.4 GENERAL-PURPOSE INTERRUPT INPUTS. The eight general-purpose latched interrupt lines, $\overline{\text{INT1}}$–$\overline{\text{INT8}}$ share device pins with the high byte of port B I/O. To enable any or all of these lines as dedicated interrupt inputs, the corresponding bit or bits in the PBCNT must be set to one. As discussed already, the current state of the interrupt pin is always available to software by reading the PBDAT, so that an interrupt handler can quickly determine the source of the interrupt (it can also determine this by the vector number).

Before interrupts are enabled in system initialization software, the required IPL should be programmed into the latched interrupt control registers (LICR1 and LICR2). Each of these 16-bit registers contains the priority setting bits (IPL) for four of the general-purpose interrupt lines, along with pending interrupt reset (PIR) bits for each of them, allowing the interrupt response software to clear any pending latched interrupt conditions. Note that the PIR bits do not clear the source of interrupt; this must either happen automatically (by the interrupting device hardware) after the interrupt condition is presented to the MC68307, or software must explicitly address the device in question, and clear the condition manually. Note also that when the IPL is changed for these latched interrupts, the PIR bit must also be set. This avoids the possibility of any spurious interrupts occurring. Refer to **Section 5.2.4 Interrupt Control Registers** for further details and programming information.

If the IPL for any one general-purpose interrupt line is programmed as 000, then that line is effectively disabled. Another way to mask a particular interrupt line from causing interrupts, without having to reprogram its priority level in the LICRx, is to clear the relevant bit in the PBCNT.

When the EC000 runs an interrupt acknowledge cycle in response to the external interrupt condition at a certain IPL, the interrupt controller module provides the appropriate vector as discussed in **Section 5.1.4.2 Interrupt Vector Generation**. User software at the address

pointed to by the respective vector location should respond to the interrupt as appropriate, including setting the PIR bit for that general-purpose interrupt input in the relevant LICRx register, to indicate end-of-service.

Refer to **Section 5.1.4 Interrupt Processing** for a short discussion of daisy-chaining with reference to the external interrupt inputs.

NOTE

The latched interrupts need to be asserted for 2 clock falling edges before they are considered valid, and are latched.

5.1.4.5 PERIPHERAL INTERRUPT HANDLING. The MC68307 on-chip I/O peripherals, consisting of the two timer channels, the UART and the M-bus modules, are all capable of being a source of interrupts. The interrupt controller logic coordinates the reception of Interrupt Request signals from these three peripherals, the return of acknowledge information with associated daisy-chaining (refer to **Section 5.1.4 Interrupt Processing**), and the ultimate providing of vector information to the EC000 core processor.

The peripheral interrupt control register (PICR) allows the user to define which IPL each of these four peripheral sources will use. The PIVR and UIVR allow the user to define a particular vector number to be presented when the respective module receives an interrupt acknowledge from the processor via the interrupt controller logic. These interrupt vector registers are initialized upon cold reset with the uninitialized interrupt vector (hex $0F), and must be programmed with the required vector number for normal operation. It is important not to use reserved interrupt vector locations for this purpose, especially the ones used by the MBAR and SCR register group.

In the case of the UART, there are multiple sources within the UART module which can cause an interrupt, e.g., transmit ready, received character/buffer, break change of state. In order to determine which of these should be serviced, the software interrupt handler must read the relevant status register within the UART module. Note that the UART module has its own vector register, allowing it to issue any programmed vector in response to an interrupt acknowledge cycle.

In the case of the M-bus interface, there are multiple sources within the M-bus module which can cause an interrupt, e.g., byte transferred, slave Rx address match, arbitration lost. Again, the relevant status register within the M-bus interface module must be read to elaborate on the condition being reported.

Finally, in the case of the timer module, there are multiple sources which can cause an interrupt, e.g., compare or capture for each of the two functional timer channels. The timer event registers can be read to determine the cause of the condition.

5.1.5 Low-Power Sleep Logic

Various options for power-saving are available—turning off unused peripherals, reducing processor clock speed, disabling the processor altogether, or a combination of these. The SCR includes a clock divider enable bit (CDEN) which substitutes a divided-down version of the system clock as the clock input to the EC000 core. The division is by a power-of-2, as programmed in the CD2–0 bits, allowing a 16MHz system clock to be divided down to approximately 64kHz. There is also a low-power sleep mode enable (LPEN) bit which stops the clock to the EC000 processor altogether after using bus arbitration to turn off the external bus. Prior to this happening, the peripherals which are not required can be disabled or their clocks stopped by programming the appropriate registers, thus further reducing power consumption. The MC68307 crystal oscillator is not stopped by these internal modes. For lowest power operation, the external clock source can be slowed or removed altogether by the user.

NOTE

Because bus arbitration is used, the external address, data buses and control signals ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$) are tristated and so should have weak pullups to minimize system power consumption.

So, various options for power-saving are available—turning off unused peripherals, reducing processor clock speed, disabling the processor altogether, or a combination of these.

A wakeup from low-power sleep mode can be achieved by causing an interrupt at the interrupt controller Logic which continues to run throughout the period of processor sleep. Any interrupt source causes a wakeup of the EC000 core processor followed by processing of that interrupt when unmasked in the SR.

The wakeup operation involves the internal sleep/wake-up logic restarting the clock to the EC000 core processor and releasing the bus. Normal processing resumes with all register contents intact, i.e., the processor continues execution from the address following the low-power enable instruction sequence. Interrupt exception processing is initiated for the interrupt which caused the wakeup.

After a wakeup, the reset source bits in the SCR are set to a value which indicates a wakeup has happened. Normally, these bits show the source of the most recent reset of the core processor. A sleep/wakeup sequence does not cause ANY reset of the static core processor or the on-chip peripherals. User software can issue a RESET instruction if any external peripherals require reset after a period of being in sleep mode.

The on-chip peripherals can initiate a wake-up, for example, the timer can be set to wake-up after a certain elapsed time, or number of external events, or the UART can cause a wake-up on receiving serial data.

The clocks provided to the various internal modules can all be gated off, to further reduce power consumption (refer to **Section 5.2.1.2 System Control Register (SCR)** for details).

In the case of the UART, its clock is restarted automatically by a transition on the RxD pin, so that incoming data is clocked in. When the data has been completely received, then an interrupt from the UART can wake-up the core processor as described above. If the other on-chip modules are required to cause a wake-up (the timer and M-bus), then their clocks should not be gated off in this manner.

5.2 PROGRAMMING MODEL

The various modules in the MC68307, including the SIM, contain registers which are used to control the modules and provide status information from the modules. Most of these registers reside in one 4096-byte range of addresses in the memory map of the EC000 core processor. The only exceptions to this rule are the MBAR and the SCR. These reside in the initial memory map of the MC68307 overlaying Motorola-reserved locations of the exception vector table. Table 5-5 shows all MC68307 on-chip locations.

Table 5-5. MC68307 Configuration Memory Map

Address	System Configuration Registers
$0000F0	Reserved–No External Bus Access
$0000F2	Module Base Address Register (MBAR)
$0000F4	System Control Register (SCR)
$0000F6	System Control Register (SCR), continued
$0000F8	Reserved–No External Bus Access
$000FA	Reserved–No External Bus Access
$0000FC	Reserved–No External Bus Access
$0000FE	Reserved–No External Bus Access

Address	SIM Module–External Bus Interface Registers	
MBASE+$011	Do Not Access Byte $010	Port A Control Register (PACNT)
MBASE+$013	Do Not Access Byte $012	Port A Data Direction Register (PADDR)
MBASE+$015	Do Not Access Byte $014	Port A Data Register (PADAT)
MBASE+$016	Port B Control Register (PBCNT)	
MBASE+$018	Port B Data Direction Register (PBDDR)	
MBASE+$01A	Port B Data Register (PBDAT)	

Address	SIM Module–Interrupt Controller Registers	
MBASE+$020	Latched Interrupt Control Register 1 (LICR1)	
MBASE+$022	Latched Interrupt Control Register 2 (LICR2)	
MBASE+$024	Peripheral Interrupt Control Register (PICR)	
MBASE+$027	Do Not Access Byte $026	Programmable Interrupt Vector Reg (PIVR)

Address	SIM Module–Chip Select Registers
MBASE+$040	Base Register 0 (BR0)
MBASE+$042	Option Register 0 (OR0)
MBASE+$044	Base Register 1 (BR1)
MBASE+$046	Option Register 1 (OR1)
MBASE+$048	Base Register 2 (BR2)

Address	SIM Module–Chip Select Registers
MBASE+$04A	Option Register 2 (OR2)
MBASE+$04C	Base Register 3 (BR3)
MBASE+$04E	Option Register 3 (OR3)

Address	SIM Module–UART Module Registers	
MBASE+$101	Do Not Access Byte $100	UART Mode Register (UMR1, UMR2)
MBASE+$103	Do Not Access Byte $102	UART Status/Clock Select Reg(USR, UCSR)
MBASE+$105	Do Not Access Byte $104	UART Command Register (UCR)
MBASE+$107	Do Not Access Byte $106	(Read) UART Receive Buffer (UTB, URB)
MBASE+$107	Do Not Access Byte $106	(Write) UART Transmit Buffer (UTB, URB)
MBASE+$109	Do Not Access Byte $108	(Read) UART Input Port Change Register (UIPCR)
MBASE+$109	Do Not Access Byte $108	(Write) UART Control Reg (UACR)
MBASE+$10B	Do Not Access Byte $10A	(Read) UART Interrupt Status Reg (UISR)
MBASE+$10B	Do Not Access Byte $10A	(Write) UART Interrupt Mask Reg (UIMR)
MBASE+$10D	Do Not Access Byte $10C	Baud Rate Gen Prescaler msb (UBG1)
MBASE+$10F	Do Not Access Byte $10E	Baud Rate Gen Prescaler lsb (UBG2)
MBASE+$119	Do Not Access Byte $118	UART Interrupt Vector Register (UIVR)
MBASE+$11B	Do Not Access Byte $11A	UART Register Input Port (UIP)
MBASE+$11D	Do Not Access Byte $11C	UART Output Port Bit Set Cmd (UOP1)
MBASE+$11F	Do Not Access Byte $11E	UART Output Port Bit Reset Cmd (UOP0)

Address	SIM Module–Timer Module Registers	
MBASE+$120	Timer Mode Register 1 (TMR1)	
MBASE+$122	Timer Reference Register 1 (TRR1)	
MBASE+$124	Timer Capture Register 1 (TCR1)	
MBASE+$126	Timer Counter 1 (TCN1)	
MBASE+$129	Do Not Access Byte $128	Timer Event Register 1 (TER1)
MBASE+$12A	Watchdog Reference Register (WRR)	
MBASE+$12C	Watchdog Counter Register (WCR)	
MBASE+$130	Timer Mode Register 2 (TMR2)	
MBASE+$132	Timer Reference Register 2 (TRR1)	
MBASE+$134	Timer Capture Register 2 (TCR2)	
MBASE+$136	Timer Counter 2 (TCN2)	
MBASE+$139	Do Not Access Byte $138	Timer Event Register 2 (TER2)

Address	SIM Module–M-Bus Module Registers	
MBASE+$141	Do Not Access Byte $140	M-Bus Address register (MADR)
MBASE+$143	Do Not Access Byte $142	M-Bus Frequency Divider Register (MFDR)
MBASE+$145	Do Not Access Byte $144	M-Bus Control Register (MBCR)
MBASE+$147	Do Not Access Byte $146	M-Bus Status Register (MBSR)
MBASE+$149	Do Not Access Byte $148	M-Bus Data I/O Register (MBDR)

A software RESET instruction resets only the UART, timer, and M-bus registers. None of the system configuration registers are reset except by a cold reset.

5.2.1 System Configuration and Protection Registers

Four reserved long-word entries in the M68000 exception vector table (refer to Table 4-5) are used as addresses for internal system configuration registers. These entries are at locations $0F0, $0F4, $0F8, and $0FC. The first entry is the on-chip MBAR entry; the second is the on-chip SCR entry; the third and fourth entries are reserved for future use by Motorola. Note that some other manufacturers use these reserved locations for other purposes, for example, a dedicated set of autovector locations for on-chip peripherals. In the MC68307, this concept is replaced by programmable interrupt vectors, which can use any spare block of 16 vector locations, except for the reserved ones.

5.2.1.1 MODULE BASE ADDRESS REGISTER (MBAR). The MBAR can be read or written at any time. It is a 16-bit read-write location, and resides in the exception vector table, at address hex $0000F2, in supervisor data space. The MBAR cannot be accessed in user data space. The register consists of the high address bits, the compare function code bit, and the function code bits, as shown below. The register should only be accessed with Word size instructions. Upon a total system reset, the MBAR value may be read as $BFFF, but this value holds no meaning, and therefore the on-chip peripheral locations cannot be accessed at any address, until the MBAR is written by the user.

MBAR $0000F2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FC2–FC0			CFC	BA23–BA12											
RESET:															
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Read/Write														Supervisor only	

FC2–FC0—Function Codes

This field should be initialized to provide the required function code which is used when the on-chip registers are accessed. The primary function of this field is to allow the user to select whether or not the block of registers should be accessible in supervisor data space only (101B) or user data space only (001B). If both are required, then the CFC bit should be cleared to 0.

NOTE

Do not assign this field to the M68000 core interrupt acknowledge space (FC2–FC0 = 111).

CFC—Compare Function Codes

This bit allows the system configuration logic to determine whether or not the user wishes to restrict the on-chip peripheral registers to access by one function code alone, as specified by the FC2–FC0 bits.

0 = The FCx bits in the MBAR are ignored. Accesses to the MC68307 peripheral registers block occur without comparing the FCx bits.
1 = The FCx bits in the MBAR are compared. The address space compare logic uses the FCx bits to detect address matches.

BA23–BA12—Base Addresses

This field should be initialized to provide the base address for the block of on-chip peripheral registers. As only address bits from A12 upwards can be specified, the block of on-chip locations thus resides at an address which is a multiple of 4096. This block can be anywhere in the address space, although should not overlap with the other chip selected areas required in the user's application.

5.2.1.2 SYSTEM CONTROL REGISTER (SCR). The SCR can be read or written at any time by 8-bit, 16-bit or 32-bit transfers. It is a 32-bit read/write location, and resides in the exception vector table, at address hex $0000F4–$0000F7, in supervisor data space. The SCR cannot be accessed in user data space. The register consists of eight status bits (bits 31–24, also called the system status register), and 24 system control bits. The eight system status bits are normally 0 and are set to 1 by some event in the system. Writing a 0 to these locations has no effect; writing a 1 clears the status bit.

SCR $0000F4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC	WPV	HWT	—	RS1–0		—	—	ADCE	WPVE	EPCS	E8051	BUSW0	BUSW1	BUSW2	BUSW3
RESET:															
0	0	0	X	0	0	X	X	0	0	0	0	BUSW	1	1	1

Read/Write Supervisor only

SCR continued $0000F6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HWDE	HW2–0			UACW	UACD	TMCD	MBCD	LPEN	CDEN	CKD	EBUSW	—	CD2–0		
RESET:															
1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0

Read/Write Supervisor only

The meanings of the various bits within this register are described in **Section 5.2.1.3 System Status Register Bits Description** and **Section 5.2.1.4 System Control Register Bits Description.**

5.2.1.3 SYSTEM STATUS REGISTER BITS DESCRIPTION. The following paragraphs describe the system status register bits.

ADC—Address Decode Conflict

This bit is set when a conflict has occurred in the chip select logic because two or more chip select lines attempt assertion in the same bus cycle. This conflict may be caused by a programming error in which the user-allocated memory areas for each chip select overlap each other. Provided the ADCE bit is set, if ADC is set this causes $\overline{\text{BERR}}$ to be asserted. If this bit is already set when another address decode conflict occurs, $\overline{\text{BERR}}$ is still generated. The chip select logic protects the MC68307 from issuing two simultaneous chip selects by employing a priority system. Write a one to this location to clear the status bit. Writing a zero has no effect.

NOTE

Regardless of the state of the chip select programming, this bit is not set and $\overline{\text{BERR}}$ is not asserted for an address decode conflict occurring during access to a SCR. This is provided to guarantee access to the system configuration registers (MBAR and SCR) during initialization.

WPV—Write-Protect Violation

This bit is set when the EC000 core processor attempts to write to a location that has the RW bit set to zero (read only) in its associated chip select base register. Provided the WPVE bit is set, $\overline{\text{BERR}}$ is asserted on the bus cycle that sets this bit. If WPV and WPVE are both set when a second write protect violation occurs, $\overline{\text{BERR}}$ is still generated. The chip select is not asserted.

Write a one to this location to clear the status bit. Writing a zero has no effect.

HWT—Hardware Watchdog Timeout

This status bit is set during a bus error ($\overline{\text{BERR}}$ asserted) caused by the hardware watchdog timeout reaching its timeout period without a $\overline{\text{DTACK}}$ being received or generated internally. Note that the bus error occurs even if this status bit is already set.

Write a one to this location to clear the status bit. Writing a zero has no effect.

RS1–RS0 — Reset Source Indication

These bits record the source of the most recent reset condition or wakeup from sleep mode (not including a peripheral reset initiated by the processor running a RESET instruction). Writing to these bits has no effect on their set value. The valid combinations are as follows (RS1 first, then RS0):

00 = Cold reset (power-on)
01 = Cold reset (reset/halt input signals)
10 = Warm reset (software watchdog timeout)
11 = Wakeup from low-power sleep mode (no reset involved)

Bits 28, 25 and 24—Reserved by Motorola

These status bits are as yet undefined, and may be used for Motorola internal test or for future options. As such they may at times have a 1 or a 0 value. Do not rely on them being always zero when performing comparisons.

5.2.1.4 SYSTEM CONTROL REGISTER BITS DESCRIPTION. The following paragraphs describe the system control register bits.

ADCE—Address Decode Conflict Enable

This bit is used to enable or disable the automatic bus error condition ($\overline{BERR}$) which can be raised when the chip select logic reports an address conflict.

0 = $\overline{BERR}$ is not asserted by a conflict in the chip select logic when two or more chip select lines are programmed to overlap the same area.
1 = $\overline{BERR}$ is asserted by a conflict in the chip select logic when two or more chip select lines are programmed to overlap the same area.

After cold reset, this bit defaults to zero, so the automatic bus error is disabled.

NOTE

When the chip select logic reports an address conflict, the ADC bit is set, regardless of the value of ADCE bit.

WPVE—Write-Protect Violation Enable

This bit is used to enable or disable the automatic bus error condition ($\overline{BERR}$ asserted) which can be raised when the chip select logic reports a write-protect violation.

0 = $\overline{BERR}$ is not asserted when a write protect violation occurs.
1 = $\overline{BERR}$ is asserted when a write protect violation occurs.

After cold reset, this bit defaults to zero, so the automatic bus error is disabled.

NOTE

When the chip select logic reports a write-protect violation, the WPV bit is set, regardless of the value of the WPVE bit.

EPCS—Enable Peripheral Chip Selects

This bit enables or disables the expansion of chip select 2 and its output pin into 4 individual peripheral chip selects. For description of usage, refer to **Section 5.1.2.2 Peripheral Chip Selects**.

0 = The $\overline{CS2}$/$\overline{CS2A}$ output pin functions as a generic chip select, $\overline{CS2}$. The $\overline{CS2B}$/PA0, $\overline{CS2C}$/PA1, and $\overline{CS2D}$/PA2 pins function as general-purpose input/output PA0, PA1 and PA2, subject to correct programming of the PACNT.
1 = The $\overline{CS2}$/$\overline{CS2A}$ output pin functions as a the first of four peripheral chip select outputs, $\overline{CS2A}$. The $\overline{CS2B}$/PA0, $\overline{CS2C}$/PA1, and $\overline{CS2D}$/PA2 pins function as the other three peripheral chip select outputs, $\overline{CS2B}$, $\overline{CS2C}$, $\overline{CS2D}$, subject to correct programming of the PACNT. The block size for this chip select should be 64-Kbytes; each output signal selects a 16-Kbyte range within the block. If less than four peripheral chip selects are required, they can be individually enabled/disabled in PACNT.

E8051—Enable 8051-Compatible Bus

This bit enables or disables the expansion of chip select 3 into an 8051-compatible bus interface, with ALE, $\overline{RD}$ and $\overline{WR}$ pins indicating the timing of an 8-bit multiplexed lower address/data bus. For description of usage, refer to **Section 5.1.2.3 8051-Compatible Bus Chip Select**.

0 = The $\overline{CS3}$ output pin functions as a generic M68000-bus chip select.
1 = The $\overline{CS3}$ output pin functions as a chip select for the 8051-compatible bus. The address latch enable output and $\overline{RD}$/$\overline{WR}$ output strobes are active for this bus cycle. Each bus cycle has timing which is compatible with 8051 memory and peripherals, rather than M68000 ones. The lower 8 address lines function as a multiplexed address/data bus. The BUSW3 bit should be programmed for 8-bit bus width (value zero), as the 8051-compatible bus necessitates an 8-bit data-bus width.

BUSW0—Bus Width for Chip-Select 0

This bit defines the data bus width (port size) for the memory device selected by $\overline{CS0}$, which is usually the boot ROM device. BUSW0 controls the bus sizing logic whenever CS0 asserts on an address match.

0 = The bus width for CS0-controlled memory accesses is 8-bits.
1 = The bus width for CS0-controlled memory accesses is 16-bits.

After cold reset, this bit takes on the logic value which was present on the BUSW external pin during the reset, thus allowing the boot ROM data width to be selected.

BUSW1—Bus Width for Chip-Select 1

This bit defines the data bus width (port size) for the memory device selected by $\overline{CS1}$. BUSW1 controls the bus sizing logic whenever CS1 asserts on an address match.

0 = The bus width for CS1-controlled memory accesses is 8-bits.
1 = The bus width for CS1-controlled memory accesses is 16-bits.

After cold reset, this bit defaults to one, so the default width for CS1-controlled memory accesses is 16-bits.

BUSW2—Bus Width for Chip-Select 2

This bit defines the data bus width (port size) for the memory device selected by $\overline{CS2}$ and also the four peripheral chip select outputs ($\overline{CS2A}$, $\overline{CS2B}$, $\overline{CS2C}$, $\overline{CS2D}$) if they are enabled by the EPCS bit. BUSW2 controls the bus sizing logic whenever CS2 asserts on an address match.

0 = The bus width for CS2-controlled memory/peripheral accesses is 8-bits.
1 = The bus width for CS2-controlled memory/peripheral accesses is 16-bits.

After cold reset, this bit defaults to one, so the default width for CS2-controlled memory accesses is 16-bits.

BUSW3—Bus Width for Chip-Select 3

This bit defines the data bus width (port size) for the M68000 memory device selected by $\overline{CS3}$. BUSW3 controls the bus sizing logic whenever CS3 asserts on an address match.

Note that if the 8051-compatible bus is enabled by the E8051 bit, then CS3 accesses are required to be an 8-bit data-bus size, and so this bit should be cleared.

0 = The bus width for CS3-controlled memory accesses is 8-bits.
1 = The bus width for CS3-controlled memory accesses is 16-bits.

After cold reset, this bit defaults to one, so the default width for CS3-controlled M68000 memory accesses is 16-bits.

HWDE—Hardware Watchdog Enable

0 = The hardware watchdog is disabled.
1 = The hardware watchdog is enabled.

After cold reset, this bit defaults to one to enable the hardware watchdog.

HW2–HW0—Hardware Watchdog Count 2–0

000 = $\overline{\text{BERR}}$ is asserted after 128 clock cycles (8 µs, 16MHz clock)
001 = $\overline{\text{BERR}}$ is asserted after 256 clock cycles (16 µs, 16MHz clock)
010 = $\overline{\text{BERR}}$ is asserted after 512 clock cycles (32 µs, 16MHz clock)
011 = $\overline{\text{BERR}}$ is asserted after 1K clock cycles (64 µs, 16MHz clock)
100 = $\overline{\text{BERR}}$ is asserted after 2K clock cycles (128 µs, 16MHz clock)
101 = $\overline{\text{BERR}}$ is asserted after 4K clock cycles (256 µs, 16MHz clock)
110 = $\overline{\text{BERR}}$ is asserted after 8K clock cycles (512 µs, 16MHz clock)
111 = $\overline{\text{BERR}}$ is asserted after 16K clock cycles (1 ms, 16MHz clock)

After cold reset, these bits default to all ones; thus, $\overline{\text{BERR}}$ is asserted after 1 ms for a 16-MHz system clock.

NOTE

Successive timeouts of the hardware watchdog may vary slightly in length. The counter resolution is 16 clock cycles.

UACW—UART Clock Wakeup

Setting this bit allows a transition on the RxD pin of the UART to cause an automatic restart of the UART clock, allowing it to receive a character, even when the UART had its clock stopped by the setting of the UACD bit. If the UART is configured to cause an interrupt on Rx character or framing error, and the UART interrupt is enabled in the peripheral interrupt control register, then the reception of a character causes an interrupt and a wakeup of the EC000 core processor if it was in low-power mode (as enabled by the LPEN bit).

0 = A transition on RxD does not restart the UART clock if disabled by UACD.
1 = A transition on RxD restarts the UART clock if disabled by UACD.

After a cold reset, this bit is cleared.

Note that software should also handle framing errors if this feature is used, in order to avoid spurious transitions on RxD causing unnecessary power consumption. If interrupt on framing error is not enabled in this situation, then the UART module remains active even if the EC000 core is in low-power mode. The interrupt handler would normally set the UART and EC000 core back into low-power mode.

UACD—UART Clock Disable

Setting this bit disables the clock to the UART module immediately. As such it should only be done when no essential data is being transmitted or received by the UART channel.

0 = The clock to the UART module is active.
1 = The clock to the UART module is disabled.

After a cold reset, this bit is cleared, and so the UART clock is active.

Refer also to the description of the UART clock wakeup (UACW bit above).

TMCD—Timer Clock Disable

Setting this bit disables the clock to the timer module immediately. As such it should only be done when no software task depends on a timer channel timeout for completion, depending on the user's application.

0 = The clock to the timer module is active.
1 = The clock to the timer module is disabled.

After a cold reset, this bit is cleared, and so the timer clock is active.

MBCD—M-Bus Clock Disable

Setting this bit disables the clock to the M-bus module immediately. As such it should only be done when no essential data is being transmitted or received by the M-bus channel.

0 = The clock to the M-bus module is active.
1 = The clock to the M-bus module is disabled.

After a cold reset, this bit is cleared, and so the M-bus clock is active.

LPEN—Low-Power Sleep Mode Enable

Setting this bit allows the EC000 core processor to be placed into the low-power sleep mode, stopping its clock input and holding the processor in a known state ready for wakeup as described previously. The sequence for user software to stop the EC000 core clock

is first to set the LPEN bit and then to issue a STOP instruction immediately to place the processor in a known state.

0 =The CPU clock is to be maintained at the programmed frequency.
1 =The CPU should be shut down as the following STOP instruction is executed. The sleep/wake-up logic coordinates the re-awakening operation when it receives an interrupt from any source. The STOP instruction should unmask all interrupts so that the processor will exit from the STOP on wakeup.

After either cold reset or wakeup, this bit is cleared in hardware, and so the processor clock runs. Note that for minimum power consumption, any unused peripherals can have their clocks stopped as well, using the UACD, MBCD, CKD and TMCD bits. All these bits should be programmed in one write cycle. Refer to **Section 5.1.5 Low-Power Sleep Logic** for further details.

CDEN—Clock Divider Enable

Setting this bit allows the internal clock signal for the EC000 core processor to be reduced in frequency as programmed in the CD2–CD0 bits.

0 =The CPU clock is to be maintained at the full operating frequency of the system clock input.
1 =The CPU clock should be maintained with a reduced frequency as programmed in the clock divider bits (CD2–CD0).

After a cold reset, this bit is cleared, so the EC000 core runs at full speed. This bit is unaffected by a wake-up, hence if the EC000 core enters sleep mode with a reduced clock frequency it will wake-up with the same reduced clock frequency. Note that the LPEN bit always overrides the CDEN/CD2–CD0 setting.

CKD—Clock-Output Disable

This bit allows the MC68307 CLKOUT pin to be disabled after reset, if it is not required externally. Turning this driver off effectively saves power and reduces potential RFI problems.

0 = The CLKOUT pin is enabled. The system clock frequency is available.
1 = The CLKOUT pin is disabled, and is driven permanently high.

After cold reset, this bit defaults to zero, thus the CLKOUT pin is active.

EBUSW—Bus Width for Non-Chip-Select Cycles with External $\overline{\text{DTACK}}$

This bit defines the data bus width (port size) for bus cycles which do not match any of the programmable chip selects or internal locations, and therefore require an external $\overline{\text{DTACK}}$ to terminate the cycle.

0 = The bus width for non-chip-selected memory accesses is 8-bits.
1 = The bus width for non-chip-selected memory accesses is 16-bits.

After cold reset, this bit defaults to one, so the default width for non-chip-selected memory accesses is 16-bits.

Bit 3—Reserved by Motorola

This control bit is as yet undefined, and may be used for Motorola internal test or for future derivative products. Leave its value as zero always.

CD2–CD0—Low-Power Clock Divider Ratio

Program these bits to specify the frequency of clock signal for the EC000 core processor when CPU clock division is enabled by the CDEN bit.

Value	Division Ratio	Frequency from 16MHz EXTAL
000	÷2	~8MHz
001	÷4	~4MHz
010	÷8	~2MHz
011	÷16	~1MHz
100	÷32	~512kHz
101	÷64	~256kHz
110	÷128	~128kHz
111	÷256	~64kHz

After a cold reset , these bits are cleared, so the EC000 core would run at half the system clock frequency, if the CDEN bit was set. These bits are unaffected by wake-up. For main low-power sleep (LPEN) operation, the CD2–CD0 and CDEN bit values are ignored.

NOTE

The clock divider ratio should only be changed from one divider value to another when the CDEN bit is zero, i.e. when the CPU is operating at full speed. Changing from one reduced speed to another is not recommended.

5.2.2 Chip Select Registers

Each of the four chip select units has two registers that define its specific operation. These registers are a 16-bit base register (BR) and a 16-bit option register (OR) (e.g., BR0 and OR0). These registers may be modified by the EC000 core. The BR should normally be programmed after the OR since the BR contains the chip select enable bit. Programming both registers at once using a long word write is also recommended.

5.2.2.1 BASE REGISTERS (BR3–BR0). These 16-bit registers consist of a base address field, a read-write bit, a function code field and an enable bit.

BR0, BR1, BR2, BR3 MBASE+$040, $044, $048, $04C

15		13	12										2	1	0
FC2	FC1	FC0	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	RW	EN
RESET:															
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Read/Write Supervisor or User

FC2–FC0—Function Code Field

These bits are used to set the address space function code. The address compare logic uses these bits to determine whether an address match exists within its address space and, therefore, whether to assert the chip select line. Although the FC2–FC0 signals are

not available externally on the MC68307, they are still available internally for comparison purposes.

111 = Not supported; reserved. Chip select does not activate if this value is used
110 = Value may be used
• •
• •
000 = Value may be used

After system reset, the FCx field in BR3–BR0 defaults to supervisor program space (FC=110) to select a ROM device containing the reset vector. Because of the priority mechanism and the enable (EN) bit, only the $\overline{CS0}$ line is active after a system reset.

NOTE

The FCx bits can be masked and ignored by the chip select logic using the compare function code bit (CFC) in the option register (ORx).

BA23–BA13—Base Address

These bits are used to set the starting address of a particular address space. The address compare logic uses only A23–A13 to cause an address match within its block size.

After system reset, the base address defaults to zero to select a ROM device on which the reset vector resides. All base address values default to zero on system reset, but, because of the priority mechanism, only $\overline{CS0}$ is active.

NOTE

All address bits can be masked and ignored by the chip select logic through the base address mask in the option register (ORx).

RW—Read/Write

This bit, in conjunction with the mask read/write bit (MRW) in the option register (ORx), allows a chip select to assert for only read cycles, only write cycles, or both types of cycle.

0 = The chip select line is asserted for read operations only.
1 = The chip select line is asserted for write operations only.

After system reset, this bit defaults to zero (read-only operation).

NOTE

This bit can be masked and ignored by the read-write compare logic, as determined by the mask read/write bit (MRW) in the option register (ORx). The line is then asserted for both read and write cycles.

On write protect violation cycles (RW=0 and MRW=1), a bus error exception ($\overline{BERR}$) is generated if the write protect violation enable bit (WPVE) in the SCR is set, and the WPV status bit in the SCR is set so that the exception handler can determine the reason for Bus Error.

EN—Enable

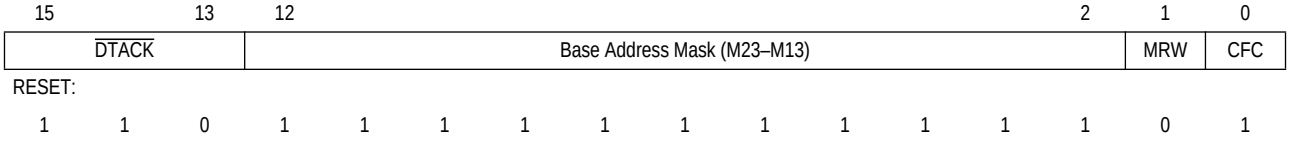
This bit enables or disables one of the chip select outputs and its associated $\overline{\text{DTACK}}$ generator. This bit can be set in the same write cycle as the setting of other fields.

0 = The chip select line is disabled.
1 = The chip select line is enabled.

After system reset, all chip selects are enabled, however only chip select 0 asserts due to the priority mechanism. The chip select does not require disabling before changing its parameters, although care should be taken if altering the configuration of the chip select that is currently asserting on the instruction stream, e.g., to relocate ROM to a higher address. It is best to ensure that both base register and option register are written in one long word cycle.

5.2.2.2 OPTION REGISTERS (OR3–OR0). These four 16-bit registers consist of a base address mask field, a read/write mask bit, a compare function code bit, and a $\overline{\text{DTACK}}$ generation field.

OR0, OR1, OR2, OR3 MBASE+$042, $046, $04A, $04E



Read/Write Supervisor or User

$\overline{\text{DTACK}}$ Field

These bits are used to determine whether $\overline{\text{DTACK}}$ is generated internally with a programmable number of wait states or externally by the peripheral. With internal $\overline{\text{DTACK}}$ generation, zero to six wait states can be automatically inserted before the $\overline{\text{DTACK}}$ pin is asserted as an output (Refer to Table 5-6).

Chip select 3 when used in the 8051-compatible bus mode, has a minimum of 6 and a maximum of 12 wait-states. The meaning of the $\overline{\text{DTACK}}$ bits changes accordingly as shown in the table.

Table 5-6. $\overline{\text{DTACK}}$ Field Encoding

Bits			M68000-Bus Cycles Description	8051-Compatible Bus Cycles Description
15	14	13		
0	0	0	No Wait State	6 Wait States
0	0	1	1 Wait State	7 Wait States
0	1	0	2 Wait States	8 Wait States
0	1	1	3 Wait States	9 Wait States
1	0	0	4 Wait States	10 Wait States
1	0	1	5 Wait States	11 Wait States
1	1	0	6 Wait States	12 Wait States
1	1	1	External $\overline{\text{DTACK}}$	External $\overline{\text{DTACK}}$

When all the bits in this field are set to one, $\overline{\text{DTACK}}$ must be generated externally, and the MC68307 waits for $\overline{\text{DTACK}}$ (input) to terminate its bus cycle.

After system reset, the bits of the $\overline{\text{DTACK}}$ field default to six wait states (M68000 cycles) or 12 wait states (8051-cycles).

The $\overline{\text{DTACK}}$ generator uses the MC68307 internal processor clock to generate the programmable number of wait states. The number of wait states remains constant as programmed even if the processor clock speed is reduced using sleep modes.

NOTE

Do not assert $\overline{\text{DTACK}}$ externally when it is programmed to be generated internally. The signal is bidirectional, and outputs as an indication of the internally generated signal.

M23–M13—Base Address Mask

These bits are used to set the block size of a particular chip select line. The address compare logic uses only the address bits that are not masked (i.e., mask bit set to one) to detect an address match within its block size.

0 = The address bit in the corresponding base register (BR) is masked; the address compare logic does not use this address bit. The corresponding external address line value is a don't care in the comparison.

1 = The address bit in the corresponding BR is not masked; the address compare logic uses this address bit.

For example, for a 64-Kbyte block, this field should be M13–M15 = 0 with the rest of the base address mask bits (M23–M16) equal to one.

After system reset, the bits of the base address mask field default to ones, (selecting the smallest block size of 8 Kbytes) to allow CS0 to select the ROM device containing the reset vector.

MRW—Mask Read/Write

This bit is used to disable or enable the comparison of read or write cycle during the chip select matching process.

0 = The read/write bit (RW) in the base register (BR) is masked. The chip select is asserted for both read and write operations.

1 = The RW bit in the BR is not masked. The chip select is asserted for read-only or write-only operations as programmed by the corresponding RW bit in BR3–BR0.

After system reset, this bit defaults to zero.

CFC—Compare Function Code

0 = The function code 2-0 bits (FC2–FC0) in the base register (BR) are ignored. The chip select is asserted without comparing the FCx bits.

1 = The FCx bits on the BR are compared. The address space compare logic uses the FCx bits to assert the $\overline{CSx}$ line.

After system reset, this bit defaults to one.

NOTE

Even when CFC=0, if the function code lines are internally generated as “111” (CPU space cycle), the chip select is not asserted.

5.2.3 External Bus Interface Control Registers

These consist only of the control and data registers associated with the external general-purpose/dedicated I/O signals. All other configuration of the external bus interface is contained within either the system configuration or the chip select logic blocks.

5.2.3.1 PORT A CONTROL REGISTER (PACNT). This 8-bit read/write register is used by the user to specify the function of the I/O lines in port A, the 8-bit I/O port, as general-purpose I/O, or dedicated I/O for the on-chip peripherals.

From a cold reset, these register bits are all cleared, thus configuring all port A I/O lines as general-purpose inputs. Bootstrap software must write an appropriate value into PACNT to reflect the system hardware setup, before any of the peripherals which need port A pins can be used properly. Pullup resistors may be required if the dedicated peripheral outputs (which share port A pins) are used, as these signals would otherwise momentarily float at cold reset until configured in the PACNT register.

PACNT MBASE+$011

7							0
CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

CA7–CA0—Port A Pin Assignment Control

These bits are used to determine whether the corresponding port A input/output line are configured as general-purpose functions or dedicated functions.

0 = The corresponding port A I/O bit is to be a *general-purpose* I/O bit.

1 = The corresponding port A I/O bit is to be a *dedicated* I/O bit.

5.2.3.2 PORT A DATA DIRECTION REGISTER (PADDR). This 8-bit read/write register is used by the user to program the general-purpose I/O lines in port A, the 8-bit I/O port, as inputs or outputs.

From a cold reset, these register bits are all cleared, configuring all port A I/O lines as general-purpose inputs. Bootstrap software must write an appropriate value into PADDR if any of the port A lines are required as outputs. Pullup resistors may be required to solve the problem of port A I/O lines perhaps floating at cold reset, until PADDR is written. They may also be required if the peripheral chip select feature (which shares port A pins) is used, as these signals would otherwise momentarily float at cold reset until configured.

PADDR MBASE+$013

7							0
DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

DA7–DA0—Data Direction Field A

These bits are used to determine whether the corresponding port A input/output line are inputs or latched outputs. As such, the DDR bit for any one port A line is ignored unless that port A line is configured as general-purpose I/O in the PACNT. If a particular general-purpose I/O line has its direction changed from an input to an output, the initial data which appears on that pin is the last data written to the latch by the PADAT register.

0 = The corresponding port A I/O bit is to be an *input*.
1 = The corresponding port A I/O bit is to be an *output*.

5.2.3.3 PORT A DATA REGISTER (PADAT). This 8-bit read/write register is used by the user to read or write the logic states of the general-purpose I/O lines in port A, the 8-bit I/O port.

From a cold reset, these register bits are all cleared, so when any port A lines are configured as outputs, a logic zero appears on those pins, unless PADAT is written with an initial data value to be written.

PADAT MBASE+$015

7							0
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

PA7–PA0—Port A Data Field

If the corresponding port A bit is configured as a general-purpose output in the PADDR and PACNT, then writing a certain logic value to that bit in the PADAT register enables that logic level to appear on the output pin. Data written is latched internally, even for pins

which are not currently configured as outputs. If a port line is reconfigured from being an input to being an output, then the most recently latched value appears on that pin.

If the corresponding port A bit is configured as a general-purpose input in the PADDR and PACNT, then the current status of that pin can be read by reading PADAT and examining the value in that bit position. Reading PADAT bits for pins configured as general-purpose outputs simply returns the output value as stored in the latch. Reading PADAT bits for pins configured as dedicated inputs/outputs for on-chip peripherals returns the current state of the pin, whether it is an input or an output.

1 = The corresponding port A bit (input or output) currently holds a logic 1 (high) value (read cycle) OR a logic 1 is to be stored to that bit (write cycle).
0 = The corresponding port A bit (input or output) currently holds a logic 0 (low) value (read cycle) OR a logic 0 is to be stored to that bit (write cycle).

5.2.3.4 PORT B CONTROL REGISTER (PBCNT). This 16-bit read/write register is used by the user to specify the function of the I/O lines in port B, the 16-bit I/O port, as general-purpose I/O, or dedicated I/O for the on-chip peripherals.

From a cold reset, these register bits are all cleared, thus configuring all port B I/O lines as general-purpose inputs. Bootstrap software must write an appropriate value into PBCNT to reflect the system hardware setup, before any of the peripherals which need port B pins can be used properly. Pullup resistors may be required if the dedicated peripheral outputs (which share port B pins) are used, as these signals would otherwise momentarily float at cold reset until configured in the PBCNT register.

PBCNT MBASE+$016

15															0
CB15	CB14	CB13	CB12	CB11	CB10	CB9	CB8	CB7	CB6	CB5	CB4	CB3	CB2	CB1	CB0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Read/Write Supervisor or User

CB15–CB0—Port B Pin Assignment Control

These bits are used to determine whether the corresponding port B input/output line are configured as general-purpose functions or dedicated functions.

0 = The corresponding port B I/O bit is to be a *general-purpose* I/O bit.
1 = The corresponding port B I/O bit is to be a *dedicated* I/O bit.

5.2.3.5 PORT B DATA DIRECTION REGISTER (PBDDR). This 16-bit read/write register is used by the user to program the general-purpose I/O lines in port B, the 16-bit I/O port, as inputs or outputs.

From a cold reset, these register bits are all cleared, configuring all port B I/O lines as general-purpose inputs. Bootstrap software must write an appropriate value into PBDDR if

any of the port B lines are required as outputs. Pullup resistors may be required to solve the problem of port B I/O lines perhaps floating at cold reset, until PBDDR is written.

PBDDR MBASE+$018

15															0
DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Read/Write Supervisor or User

Bits 15–0—Data Direction Field B (DB15–DB0)

These bits are used to determine whether the corresponding port B input/output line are inputs or latched outputs. As such, the DDR bit for any one port B line is ignored unless that port B line is configured as a general-purpose output in the PBCNT. If a particular general-purpose I/O line has its direction changed from an input to an output, the initial data which appears on that pin is the last data written to the latch by the PBDAT register.

0 = The corresponding port B I/O bit is to be an *input*.
1 = The corresponding port B I/O bit is to be an *ouput*.

5.2.3.6 PORT B DATA REGISTER (PBDAT). This 16-bit read/write register is used by the user to read or write the logic states of the general-purpose I/O lines in port B, the 8-bit I/O port.

From a cold reset, these register bits are all cleared, so when any port B lines are configured as outputs, a logic zero appears on those pins, unless PBDAT is written with an initial data value to be written.

PBDAT MBASE+$01A

15															0
PB15	PB14	PB13	PB12	PB11	PB10	PB9	PB8	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Read/Write Supervisor or User

PB7–PB0—Port B Data Field

If the corresponding port B bit is configured as a general-purpose output in the PBDDR, then writing a certain logic value to that bit in the PBDAT register enables that logic level to appear on the output pin. Data written is latched internally, even for pins which are not currently configured as outputs. As already mentioned, if a port line is reconfigured from being an input to being an output, then the most recently latched value appears on that pin.

If the corresponding port B bit is configured as a general-purpose input in the PBDDR, then the current status of that pin can be read by reading PBDAT and examining the value in that bit position. Reading PBDAT bits for pins configured as general-purpose outputs simply returns the output value as stored in the latch. Reading PBDAT bits for pins con-

figured as dedicated inputs/outputs for on-chip peripherals returns the current state of the pin, whether it is an input or an output.

1 = The corresponding Port B bit (input or output) currently holds a logic 1 (high) value (read cycle) OR a logic 1 is to be stored to that bit (write cycle).

0 = The corresponding Port B bit (input or output) currently holds a logic 0 (low) value (read cycle) OR a logic 0 is to be stored to that bit (write cycle).

5.2.4 Interrupt Control Registers

The following paragraphs describe the interrupt control registers.

5.2.4.1 LATCHED INTERRUPT CONTROL REGISTERS 1,2 (LICR1,LICR2). These registers control the interrupt priorities for the external general-purpose latched interrupt input signals, and also allow software to reset any pending interrupts from these lines. There are eight general-purpose latched interrupt inputs altogether, each has four bits assigned to it in these registers. The registers can be read or written at any time. When read, the data returned is the last value that was written to the register, with the exception of the reset bits, which are transitory functions. The registers can be accessed by either word (16-bit) or byte (8-bit) data transfer instructions. An 8-bit write to one half of a register leaves the other half intact.

LICR1 MBASE+$020

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIR1	INT1IPL(2–0)			PIR2	INT2IPL(2–0)			PIR3	INT3IPL(2–0)			PIR4	INT4IPL(2–0)		
RESET:															
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

Read/Write Supervisor or User

LICR2 MBASE+$022

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIR5	INT5IPL(2–0)			PIR6	INT6IPL(2–0)			PIR7	INT7IPL(2–0)			PIR8	INT8IPL(2–0)		
RESET:															
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

Read/Write Supervisor or User

PIR1–PIR8—Pending Interrupt Reset 1–8

These bits allow the user to clear any pending interrupt on the interrupt input latch for the specified input ($\overline{\text{INT1}}$–$\overline{\text{INT8}}$). The action of writing a logic 1 to one of these bit positions clears the latch, so that the interrupt input must be toggled before another interrupt is latched. It is not necessary to subsequently write a logic 0 to 'release' the reset condition, this is done internally.

When read, these bits indicate the current value of the latched interrupt, a 1 indicating active, a zero inactive.

These bits also enable a new value of the IPL field to be set. Each interrupt channel ignores the value written to its INTxIPL(2–0) field UNLESS the corresponding PIRx bit is

set. To avoid altering the IPL field values when only a pending interrupt reset is required, use an AND.B, AND.W or AND.L operation with an immediate value containing ones for the PIRx bit positions required to be set, with the corresponding IPL fields set to 111.

1 = The corresponding $\overline{\text{INT1}}$–$\overline{\text{INT8}}$ interrupt latch is cleared, so that a pending interrupt on that line is discarded. The new IPL value for that interrupt source is stored (write cycle).

0 = The corresponding $\overline{\text{INT1}}$–$\overline{\text{INT8}}$ interrupt latch is unaffected. Any pending interrupt on that line remains pending. The IPL value for that source remains unchanged (write cycle).

INT1IPL(2–0)–INT8IPL(2–0)—Interrupt Priority Level 1–8

These bits allow the user to specify the IPL for the corresponding general-purpose latched interrupt input line. When an interrupt occurs, the interrupt controller logic asserts the correct priority code on the EC000 interrupt level inputs, and respond to the subsequent acknowledge cycle by outputting the correct vector. The IPL for any of the 8 external interrupt sources $\overline{\text{INT1}}$–$\overline{\text{INT8}}$ can only be changed if the corresponding PIRx bit is also written as 1.

000 = The corresponding $\overline{\text{INT1}}$–$\overline{\text{INT8}}$ interrupt line is inhibited and cannot generate interrupts. Its state can still be read via the port B registers.

001–111= The corresponding $\overline{\text{INT1}}$–$\overline{\text{INT8}}$ interrupt line is enabled, and can generate an interrupt to the EC000 core processor with the indicated priority level.

NOTE

Do not write a 1 to a PIR bit without also supplying the correct IPL2–IPL0 value for that interrupt channel. The IPL2–IPL0 value is written, and if ‘000’ is supplied, then that channel is effectively disabled. Use the AND method as described above.

5.2.4.2 PERIPHERAL INTERRUPT CONTROL REGISTER (PICR). This register controls the interrupt priorities for the interrupt signals from the MC68307 internal I/O modules, in the same way as the latched interrupt control registers do so for the external latched interrupt input lines The modules which can provide interrupts to the EC000 core processor are: the M-bus interface module, the timer module (both channels), and the UART interface module. Each of these sources has four bits assigned to it in the peripheral interrupt control register. The PICR can be read or written at any time. When read, the data returned is the last value that was written to the register. This register can be accessed by either word (16-bit) or byte (8-bit) data transfer instructions. An 8-bit write to one half of the register leaves the other half intact.

PICR MBASE+$024

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	T1IPL(2–0)			—	T2IPL(2–0)			—	UAIPL(2–0)			—	MBIPL(2–0)		
RESET:															
X	0	0	0	X	0	0	0	X	0	0	0	X	0	0	0

Read/Write Supervisor or User

Interrupt Priority Level Timer 1, Timer 2, UART, MBUS, T1IPL2–0, T2IPL2–0, MBIPL2–0, UAIPL2–0

These bits allow the user to specify the IPL for the corresponding on-chip peripheral module interrupt input line (Timer 1, Timer 2, UART and M-bus respectively). When an interrupt occurs, the interrupt controller logic asserts the correct priority code on the EC000 interrupt level inputs, and respond to the subsequent acknowledge cycle by coordinating the return of the appropriate vector, as programmed into the programmable interrupt vector register or UART interrupt vector register.

000 = The corresponding interrupt source is inhibited and cannot generate interrupts.

001 - 111 =The corresponding interrupt source is enabled, and can generate an interrupt to the EC000 core processor with the indicated priority level.

The remaining four bits (Bits 14, 11, 7, 3) are reserved for future implementation, and should always be written as one. When read, their value should be ignored.

5.2.4.3 PROGRAMMABLE INTERRUPT VECTOR REGISTER (PIVR). This register specifies the vector numbers which is returned by the interrupt controller in response to Interrupt Acknowledge cycles for the various peripherals and discrete interrupt sources. The high four bits of the vector number are programmed in the PIVR, and the low four bits are provided by the interrupt controller depending on the highest priority source which is currently active for the specific IPL being responded to in the current acknowledge cycle.

PIVR MBASE+$027

7	6	5	4	3	2	1	0
IV7	IV6	IV5	IV4	—			
RESET:							
0	0	0	0	1	1	1	1

Read/Write Supervisor or User

IV7–IV4—Interrupt Vector Numbers 7–4

These bits provide the high four bits of the interrupt vector for interrupt acknowledge cycles from all sources except the UART, which provides its own, unique vector number. The UART has a separate vector register for historical reasons, due to its compatibility with the MC68681 DUART.

Bits 3–0—Unused

These bits are ignored on a write cycle, and return 1 on a read cycle. The interrupt controller provides the appropriate encoding on these four bits of the vector (Refer to **Section 5.1.4.2 Interrupt Vector Generation**).

After cold reset, this register contains the value $0F, although no interrupts are ever propagated to the CPU until the PIVR is first programmed (not even level 7 interrupts). Only write to the PIVR after the CPU stack pointer has been set up to point to valid addressable RAM, i.e., after the chip selects for RAM have been set up.

5.3 MC68307 INITIALIZATION PROCEDURE

The following paragraphs describe the initialization procedure.

5.3.1 Startup—Cold Reset

Upon cold reset (power-on reset, $\overline{\text{RSTIN}}$ asserted, $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ pins asserted or software watchdog timeout), all external accesses in the first 8K bytes of the memory address space asserts chip select 0, which should be used to enable a boot ROM device. Before the chip selects can be set up any other way, the MC68307 on-chip register block must be located in its desired memory map slot by programming the module base address register in supervisor data space at address $0000F2. Note that accessing this on-chip register does not in any way access the ROM on chip select 0, whose address range includes the MBAR address.

5.3.2 SIM Configuration

The cold reset routine should configure all of the following SIM functions: chip selects, I/O pin function definition and initial value, interrupt controller IPL levels and vector numbers. These are the “system integration” features which would not normally be changed subsequently, although there is nothing to prevent this if required.

SECTION 6
DUAL TIMER MODULE

6.1 OVERVIEW

The MC68307 includes three timer units: two identical general-purpose timers and a software watchdog timer.

Each general-purpose timer consists of a timer mode register (TMR), a timer capture register (TCR), a 16-bit timer counter (TCN), a timer reference register (TRR), and a timer event register (TER). The TMR contains the prescaler value programmed by the user. The software watchdog timer, which has a watchdog reference register (WRR) and a watchdog counter (WCN), uses a fixed prescaler value. The timer block diagram is shown in Figure 6-1.

The two identical general-purpose 16-bit timer units have the following features:

- Maximum Period of 16 Seconds (at 16.67 MHz)
- 60-ns Resolution (at 16.67 MHz)
- Programmable Sources for the Clock Input, Including External Clock
- Input Capture Capability with Programmable Trigger Edge on Input Pins
- Output Compare with Programmable Mode for the Output Pins
- Two Timers Externally Cascadable to Form a 32-Bit Timer
- Free Run and Restart Modes

The watchdog timer has the following features:

- 16-bit Counter and Reference Register
- Maximum Period of 16.78 Seconds (at 16 MHz)
- 0.5 ms Resolution (at 16 MHz)
- Timeout Causes Reset of the MC68307 (CPU and Peripherals).

6.2 MODULE OPERATION

The following paragraphs describe the operation of the dual timer module.

6.2.1 General-Purpose Timer Units

The clock input to the prescaler may be selected from the system clock (divided by 1 or by 16) or from the corresponding timer input (TIN1 or TIN2) pin. TIN is internally synchronized to the internal clock. The clock input source is selected by the ICLK bits of the corresponding

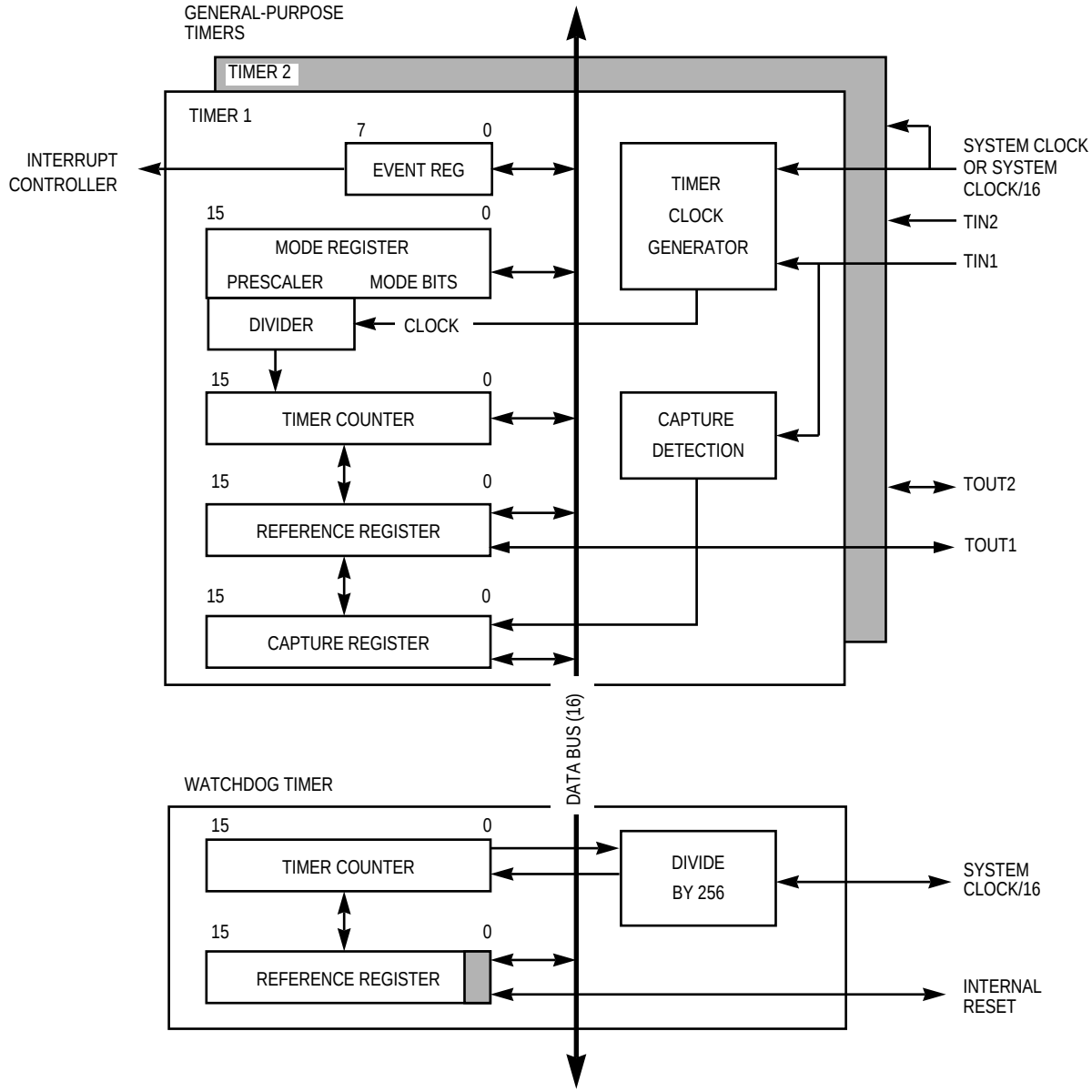


Figure 6-1. Timer Block Diagram

TMR. The prescaler is programmed to divide the clock input by values from 1 to 256. The output of the prescaler is used as an input to the 16-bit counter.

The maximum resolution of the timer is one system clock cycle (60ns at 16.67 MHz). The maximum period (when the reference value is all ones) is 268,435,456 cycles (16.78 seconds at 16.00 MHz).

Each timer may be configured to count until a reference is reached and then either start a new time count immediately or continue to run. The free run/restart (FRR) bit of the corresponding TMR selects each mode. Upon reaching the reference value, the corresponding TER bit is set, and an interrupt is issued if the output reference interrupt enable (ORI) bit in TMR is set.

Each timer may output a signal on the timer output (TOUT1 or TOUT2) pin when the reference value is reached, as selected by the output mode (OM) bit of the corresponding TMR. This signal can be an active-low pulse or a toggle of the current output, under program control. The output can also be used as an input to the other timer, resulting in a 32-bit timer.

Each timer has a 16-bit TCR, which is used to latch the value of the counter when a defined transition (of TIN1 or TIN2) is sensed by the corresponding input capture edge detector. The type of transition triggering the capture is selected by the capture edge (CE) bits in the corresponding TMR. Upon a capture or reference event, the corresponding TER bit is set, and a maskable interrupt is issued.

6.2.2 Software Watchdog Timer

A software watchdog timer is used to protect against system failures by providing a means to escape from unexpected input conditions, external events, or programming errors. The third 16-bit timer block is used for this purpose. Once started, the software watchdog timer must be cleared by software on a regular basis so that it never reaches its timeout value. Upon reaching the timeout value, the assumption is made that a system failure has occurred, and the software watchdog logic initiates a reset of the MC68307.

The software watchdog timer counts from zero to a maximum of 32768 (16.67 seconds at 16.00 MHz) with a resolution or step size of 8192 clock periods (0.5 ms at 16.00 MHz). This timer uses a 16-bit counter with an 8-bit prescaler value.

The software watchdog timer uses the system clock divided by 16 as the input to the prescaler. The prescaler circuitry divides the clock input by a fixed value of 256. The output of this prescaler circuitry is connected to the input of the 16-bit counter. Since the least significant bit of the WCN is not used in the comparison with the WRR reference value, the effective value of the prescaler is 512.

The timer counts until the reference value is reached and then starts a new time count immediately. Upon reaching the reference value, the counter asserts an internal output to the MC68307 reset logic. The reset source bits (RS1-RS0) in the system control register (SCR) are updated with the cause of reset being indicated as software watchdog. Refer to **Section 5.2.1.2 System Control Register (SCR)** for further details of the RSx bits.

The value of the timer can be read any time.

6.3 PROGRAMMING MODEL

The following paragraphs describe the programming model of the dual timer module.

6.3.1 General Purpose Timer Units

The Timer registers may be modified at any time by the user.

6.3.1.1 TIMER MODE REGISTER (TMR1, TMR2). TMR1 and TMR2 are identical 16-bit memory-mapped registers. They are cleared by reset.

TMR1 MBASE+$120
TMR2 MBASE+$130

15 … 8	7	6	5	4	3	2	1	0
PRESCALER VALUE (PS7–0)	CE1–CE0		OM	ORI	FRR	ICLK1–ICLK0		RST
RESET:								
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0

Read/Write — Supervisor or User

PS—Prescaler Value

The prescaler is programmed to divide the clock input by values from 1 to 256. The value 00000000 divides the clock by 1; the value 11111111 divides the clock by 256.

CE1–CE0—Capture Edge and Enable Interrupt

- 00 = Disable interrupt on capture event
- 01 = Capture on rising edge only and enable interrupt on capture event
- 10 = Capture on falling edge only and enable interrupt on capture event
- 11 = Capture on any edge and enable interrupt on capture event

OM—Output Mode

- 0 = Active-low pulse for one CLKO clock cycle (60ns at 16.67 MHz)
- 1 = Toggle output

ORI—Output Reference Interrupt Enable

- 0 = Disable interrupt for reference reached (does not affect interrupt on capture function)
- 1 = Enable interrupt upon reaching the reference value

FRR—Free Run/Restart

- 0 = Free run: timer count continues to increment after the reference value is reached.
- 1 = Restart: timer count is reset immediately after the reference value is reached.

ICLK1–ICLK0—Input Clock Source for the Timer

- 00 = Stop count
- 01 = Master system clock
- 10 = Master system clock divided by 16. Note that this clock source is not synchronized to the timer; thus, successive timeouts may vary slightly in length.
- 11 = Corresponding TIN pin, TIN1 or TIN2 (falling edge)

RST—Reset Timer

This bit performs a software reset of the timer similar to that of an external reset, although while this bit is zero, the other register values can still be written. Effectively, a transition of this bit from one to zero is what resets the register values. The counter/timer/prescaler is not clocked unless the timer is enabled.

0 = Reset timer (software reset)
1 = Enable timer

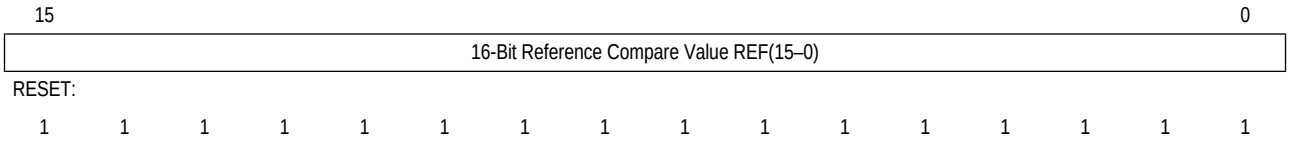
NOTE

After reset, the TOUT signal begins in a high state, but is not available externally until the port A control register (PACNT) is configured for this function.

6.3.1.2 TIMER REFERENCE REGISTERS (TRR1, TRR2). Each TRRx is a 16-bit register containing the reference value that is compared with the free-running TCN as part of the output compare function. TRR1 and TRR2 are memory-mapped read-write registers.

TRR1 and TRR2 are set to all ones by reset. The reference value is not matched until TCN increments to equal TRR.

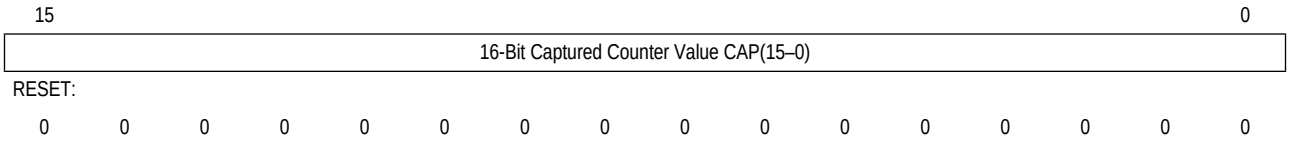
TRR1 MBASE+$122
TRR2 MBASE+$132



Read/Write Supervisor or User

6.3.1.3 TIMER CAPTURE REGISTERS (TCR1, TCR2). Each TCRx is a 16-bit register used to latch the value of the TCN during a capture operation when an edge occurs on the respective TIN1 or TIN2 pin, as programmed in the TMR. TCR1 and TCR2 appear as memory-mapped read-only registers to the user and are cleared at reset.

TCR1 MBASE+$124
TCR2 MBASE+$134

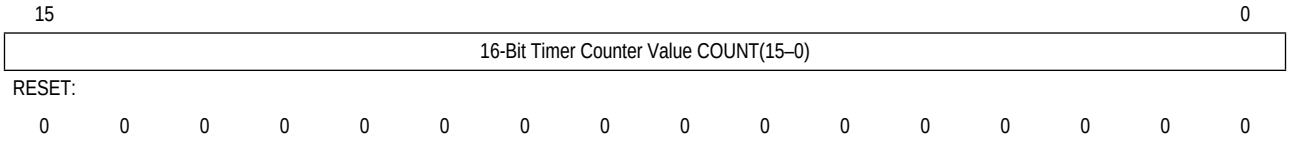


Read-only Supervisor or User

6.3.1.4 TIMER COUNTER (TCN1, TCN2). TCN1 and TCN2 are 16-bit up-counters. Each is memory-mapped and can be read by the user at any time. A read cycle to TCN1 and TCN2 yields the current value of the timer and does not affect the counting operation.

A write cycle to TCN1 and TCN2 causes it and the corresponding prescaler to be reset.

TCN1 MBASE+$126
TCN2 MBASE+$136



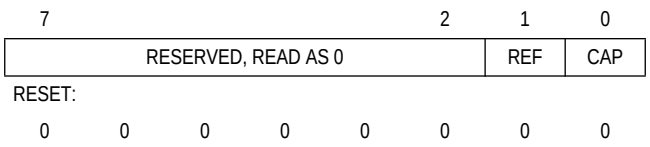
Read/Write Supervisor or User

6.3.1.5 TIMER EVENT REGISTERS (TER1, TER2). Each TERx is an 8-bit register used to report events recognized by any of the timers. On recognition of an event, the timer sets the appropriate bit in the TER, regardless of the corresponding interrupt enable bits (ORI and CE) in the TMR.

TER1 and TER2, which appear to the user as memory-mapped registers, may be read at any time.

A bit is cleared by writing a one to that bit (writing a zero does not affect a bit's value). More than one bit may be cleared at a time. Both bits must be cleared before the timer negates the IRQ to the interrupt controller. This register is cleared at reset.

TER1 MBASE+$129
TER2 MBASE+$139



Read/Write Supervisor or User

Bits 7–2—Reserved by Motorola.

These bits are currently zero when read.

REF—Output Reference Event

The counter has reached the TRR value. The ORI bit in the TMR is used to enable the interrupt request caused by this event. Write a one to this bit to clear the event condition.

CAP—Capture Event

The counter value has been latched into the TCR. The CE bits in the TMR are used to enable the interrupt request caused by this event. Write a one to this bit to clear the event condition.

6.3.2 Software Watchdog Timer

The software watchdog timer module has an 8-bit prescaler that is not accessible to the user, a read-only 16-bit watchdog counter register WCR, and a WRR.

6.3.2.1 WATCHDOG REFERENCE REGISTER (WRR). The WRR is a 16-bit register containing the reference value for the watchdog timeout. The WRR appears as a memory-mapped read-write register to the user.

WRR MBASE+$12A

15															0
15-Bit Reference Value REF(15–1)															EN
RESET:															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Read/Write Supervisor or User

Reset initializes the register to $FFFF, enabling the watchdog timer and setting it to the maximum timeout period. This causes a timeout to occur if there is an error in the boot program.

Reference Value

This 15-bit field should be written with the limit value for the corresponding WCR count bits. It is 15-bits wide because the least-significant bit of the WCR is not compared.

EN — Enable Watchdog

1 = The watchdog timer is enabled, software should periodically write to the WCR location, so that the counter never reaches the above reference value.
0 = The watchdog timer is disabled, and does not count.

6.3.2.2 WATCHDOG COUNTER REGISTER (WCR). The WCR is a 16-bit up-counter, appears as a memory-mapped register and may be read at any time. Clearing EN in the WRR causes the counter to be reset and disables the count operation.

WCR MBASE+$12C

15															0
16-Bit Counter Value COUNT(15–0)															
RESET:															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Read/Write Supervisor or User

A read cycle to WCR causes the current value of the watchdog timer to be read. Reading the watchdog timer does not affect the counting operation.

A write cycle to WCR causes the counter and prescaler to be reset. A write cycle should be executed on a regular basis so that the watchdog timer is never allowed to reach the reference value during normal program operation.

6.4 TIMER PROGRAMMING EXAMPLES

The programming examples below are written in M68000 assembly language. They refer to symbolic names of MC68307 registers and bit fields within those registers. For access to on-chip locations, the addressing mode Offset_From(A5) is used. Below is a list of the bit definitions used in the examples.

```
* TMRx bit definitions (non-zero cases only)

RST    EQU    $0001       TMRx RST    ; enable timer
ICLK01 EQU    $0002       TMRx ICLK   ; use system clock
ICLK10 EQU    $0004       TMRx ICLK   ; use system clock divide by 16
ICLK11 EQU    $0006       TMRx ICLK   ; use TINx as clock
FRR    EQU    $0008       TMRx FRR    ; restart timer on ref. compare
ORI    EQU    $0010       TMRx ORI    ; enable interrupt on ref. compare
OM     EQU    $0020       TMRx OM     ; toggle TOUTx
CE01   EQU    $0040       TMRx CE     ; capture on +ve edge clk + interrupt
CE10   EQU    $0080       TMRx CE     ; capture on -ve edge clk + interrupt
CE11   EQU    $00C0       TMRx CE     ; capture on both clk edges + interrupt

* TERx bit masks

REF    EQU    $02                     ; isolate TERx REF bit
CAP    EQU    $01                     ; isolate TERx CAP bit
```

The examples assume that the timer is in a reset state prior to execution of the code sequence, that is, either this is the first timer access after a system reset or the timer mode register RST bit has had a programmed transition from 1 to 0.

6.4.1 Initialization and Reference Compare Function

This code sequence starts both timers in an output reference compare mode, outputting a repeating pulse on each channel's TOUT pin after a preset period. The reference value and prescale value have been made different in each case to demonstrate programming of different periods.

Interrupts are disabled in this example. If the ORI bit were also set, interrupts would be enabled, one for each corresponding TOUTx pulse.

TOUTx is also set to output to a single clock pulse on each match. The alternate mode is toggle mode, accomplished by also setting the OM bit of TMRx.

Note that the timer mode registers can be programmed in one operation. There is no need to have separate writes to take the timer out of reset, to program the various modes and then to finally start the clock.

```
* Enable timer output pins in port A control register

       OR.B   #$18,PACNT(A5)          ; TOUT1, 2 pins67 dedicated

* Setup timer 1 and timer 2 to run independently of each other
* Both counters reset (and repeat forever) after reference reached
* Both disabled interrupts for reference
```

```
* Active-low pulses on TOUT1 and TOUT2
* Both disabled interrupts on capture

* TOUT1 pulse generated every (2 x 9) CPU clocks with no interrupts
* Timer 1 prescaler: divide by 2, reference: divide by 9

       MOVE.W #8,TRR1(A5)             ; reference value 8
       MOVE.W #RST+ICLK01+FRR+$0100,TMR1(A5); value=$010B

* TOUT2 pulse generated every (3 x 7) CPU clocks with no interrupts
* Timer 2 prescalar: divide by 3, reference: divide by 7

       MOVE.W #6,TRR2(A5)                   ; reference value 6
       MOVE.W #RST+ICLK01+FRR+$0200,TMR2(A5); value=$020B
```

6.4.2 Event Counting Function and Interrupts

A variation on the reference compare function demonstrated above is the following example of event counting with an interrupt to the CPU after N events have been counted.

Note that the reference register is programmed with the compare value before the timer is started, otherwise a spurious compare may occur. Although the timer channel is in reset (RST bit = 0), other registers can be accessed.

```
* Enable timer 1 input pin in port B control register

       OR.W   #$0040,PBCNT(A5)              ; TIN1 pin dedicated

* Set up TIN1 as clock source;
* Interrupt generated every 4 TIN1 clocks, so reference value = 3
* Counter reset after reference reached
* Enable interrupt for reference
* Active-low pulse on TOUT1
* Disable interrupt on capture
* Prescaler: divide by 1

       MOVE.W #3,TRR1(A5)                   ; reference value 3
       MOVE.W #RST+ICLK11+FRR+ORI,TMR1(A5)  ; value=$001F
       STOP   $2000                         ; wait for IRQ
....   ....                           ....

INT_T1;ANDI.B #REF,TER1(A5)                 ; clear REF event bit
....   ....                                 ; Process the interrupt
       RTE
```

6.4.3 Input Capture Function

This example provides an interrupt routine that reads the captured timer value each time the TIN2 pin has a transition from 0 to 1. The timer is programmed to be free-running (FRR bit not set) and so the reference value is ignored.

Note that the interrupt service routine does not check which event bit is set as the cause of the interrupt. A more robust system would do this, and indeed a timer channel could have both interrupt sources enabled.

```
* Enable timer 2 input pin in port B control register

       OR.W   #$0080,PBCNT(A5)             ; TIN2 pin dedicated

* Capture on TIN2 +ve edge using system clock as clock source
* Get captured timer word in register D5
* Counter free-running
* Disable interrupt for reference
* Enable interrupt on capture on +ve TIN2 edge
* Prescaler: divide by 1

       MOVE.W #RST+ICLK01+CE01,TMR2(A5); value=$0043
....   ....                            ....

INT_T2:ANDI.B #CAP,TER2(A5)            ; clear CAP event bit
       MOVE.W TCR2(A5),D5              ; Process the interrupt
       RTE
```

6.4.4 Watchdog Usage Example

The first example shows how to disable the watchdog after system reset if it is not required. Note that if a RESET instruction is executed, the timer module is reset, and so the watchdog is enabled again.

```
* Disable watchdog timer forever

       MOVE.W #$0000,WRR(A5)           ; clear WRR EN bit
```

The second example shows setting the watchdog timer to timeout every 10mS unless it is periodically reset by software. The watchdog is clocked by the system clock divided by 16, and there is a fixed 8-bit prescaler, so the resulting clock count value increments every 0.24576mS for a 16.67 MHz system clock.

The watchdog reference register is programmed to match on a compare value of 40, to give an approximate 10mS timeout. Note that the least significant bit of the counter is not compared and the corresponding reference register bit must be 1 to enable the watchdog.

```
* Program watchdog for 10mS timeout

       MOVE.W #$0029,WRR(A5)           ; enable = 1, timeout = 40
....   ....                            ....
       MOVE.W D0,WCR                   ; tickle watchdog counter
```

Note that it does not matter what data value is written to the WCR to keep it from timing out. The example uses D0 as the source, since it results in the fastest executing instruction. (The CLR instruction on a 68000 reads the location before clearing it.)

SECTION 7
M-BUS INTERFACE MODULE

Motorola bus (in short: M-bus) is a two-wire, bidirectional serial bus which provides a simple, efficient method of data exchange between devices. It is compatible with the widely-used I^2C bus standard. This two wire bus minimizes the interconnection between devices and eliminates the need for an address decoder.

This bus is suitable for applications which need occasional communications in a short distance among a number of devices. It also provides flexibility that allows additional devices to be connected to the bus for further expansion and system developing.

The interface is designed to operate up to 100 Kb/s with maximum bus loading and timing. The device is capable of operating at higher baud rates, with reduced bus loading. The maximum communication length and number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

M-bus system is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters attempt to control the bus simultaneously. This feature provides the capability for complex applications with multi-processor control. It may also be used for rapid testing and alignment of end products via external connections to an assembly-line computer.

The M-bus module has the following key features:

- Compatible with I^2C Bus Standard
- Multi-Master Operation
- Software Programmable for One of 32 Different Serial Clock Frequencies
- Software Selectable Acknowledge Bit
- Interrupt Driven Byte By Byte Data Transfer
- Arbitration Lost Interrupt with Automatic Mode Switching from Master to Slave
- Calling Address Identification Interrupt
- Start and Stop Signal Generation/Detection
- Repeated START Signal Generation
- Acknowledge Bit Generation/Detection
- Bus Busy Detection

A block diagram of the complete M-bus module is shown in Figure 7-1.

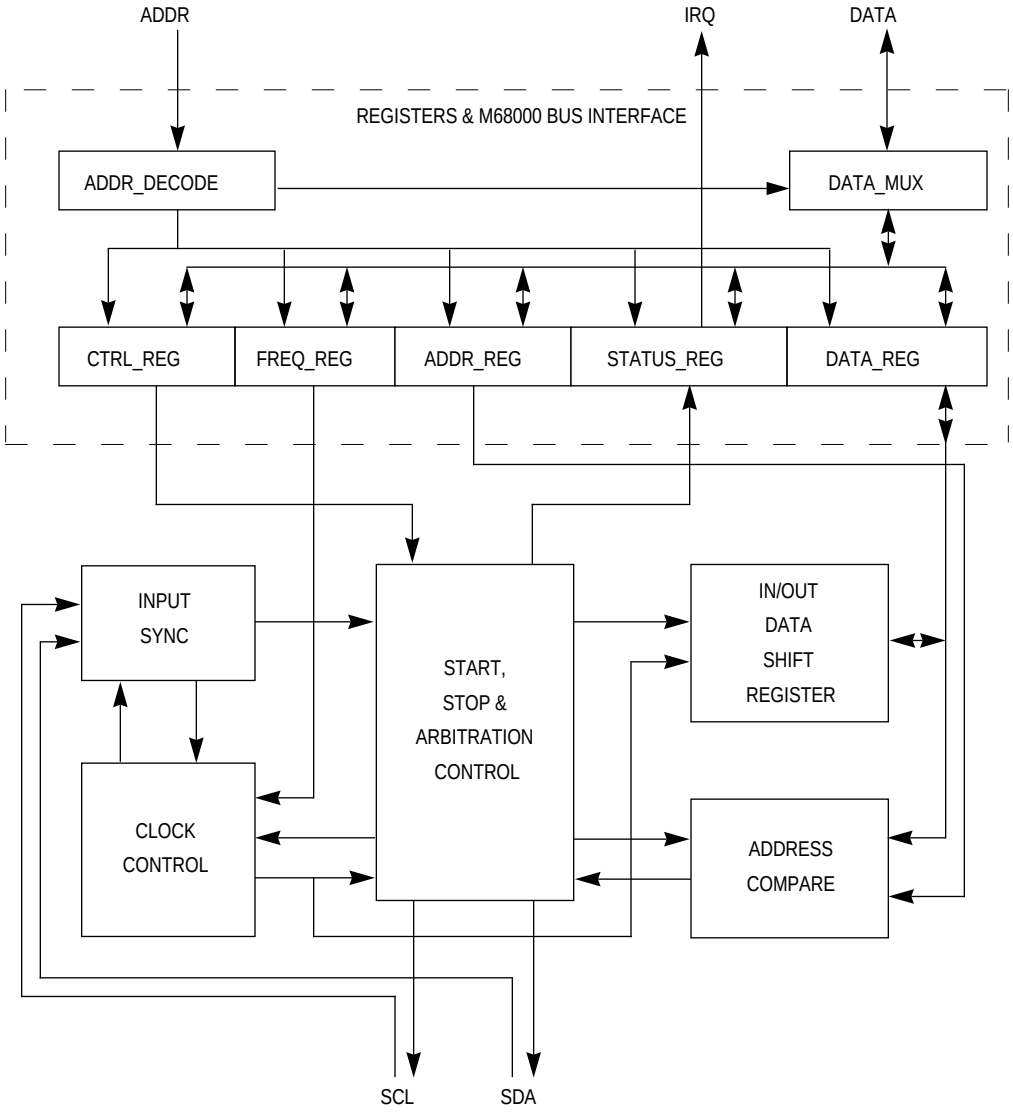


Figure 7-1. M-Bus Interface Block Diagram

7.1 M-BUS SYSTEM CONFIGURATION

The M-bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. The logical AND function is exercised on both lines with pull-up resistors.

7.2 M-BUS PROTOCOL

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. They are described briefly in the following sections and illustrated in Figure 7-2.

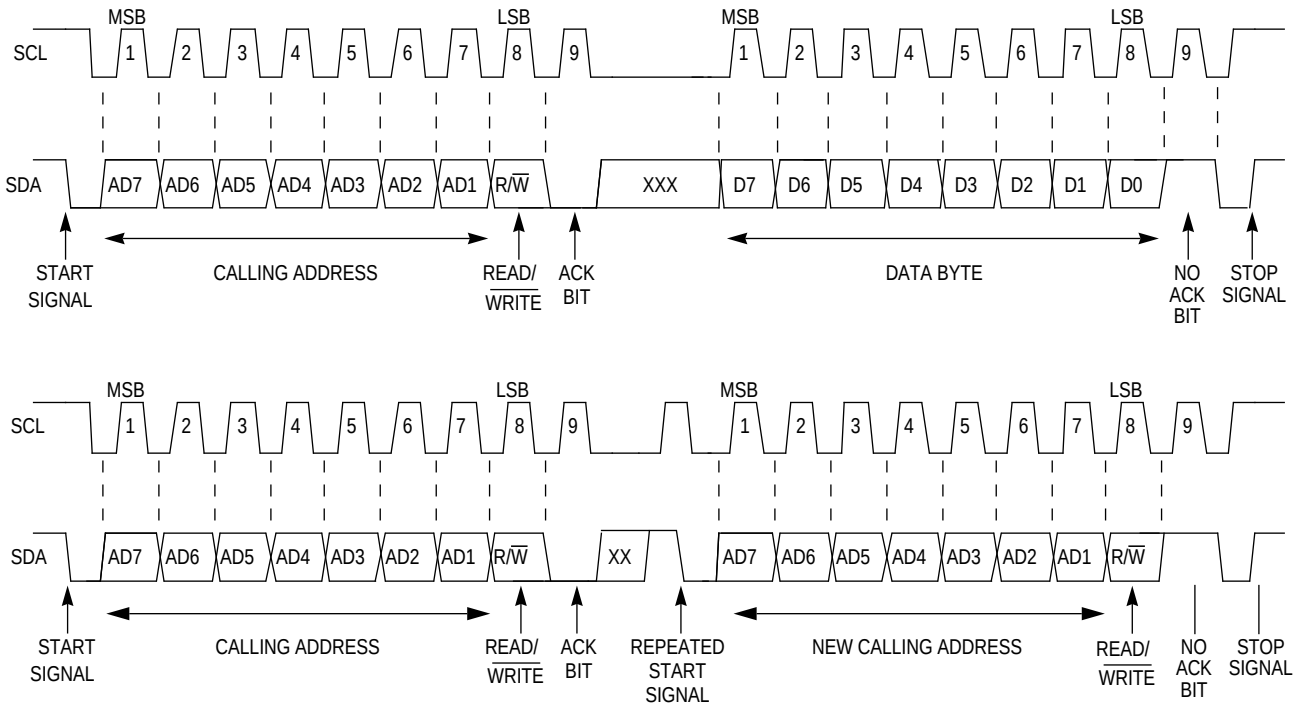


Figure 7-2. M-Bus Transmission Signals

7.2.1 START Signal

When the bus is free, i.e., no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 7-2, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and wakes up all slaves.

7.2.2 Slave Address Transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven bits long calling address followed by a R/$\overline{W}$ bit. The R/$\overline{W}$ bit tells the slave the desired direction of data transfer.

Only the slave with a calling address that matches the one transmitted by the master responds by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (refer to Figure 7-2).

7.2.3 Data Transfer

Once successful slave addressing is achieved, the data transfer can proceed byte by byte in a direction specified by the R/$\overline{W}$ bit previously sent by the calling master.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in Figure 7-2. There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte has to be followed by an acknowl-

edge bit, which is signalled from the receiving device by pulling the SDA low at the 9th clock. So one complete data byte transfer needs 9 clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means 'end of data' to the slave, so that the slave should release the SDA line for the master to generate STOP or START signal.

7.2.4 Repeated START Signal

As shown in Figure 7-2, a repeated START signal is to generate a START signal without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/ receive mode) without releasing the bus.

7.2.5 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL at logical high. (Refer to Figure 7-2).

7.2.6 Arbitration Procedure

M-bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure, a bus master loses arbitration if it transmits logic “1” while another master transmits logic “0”. The losing masters immediately switch over to slave receive mode and stop driving SDA output. In this case the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate lost of arbitration.

7.2.7 Clock Synchronization

Since wire-AND logic is performed on SCL line, a high-to-low transition on SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change from low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (Refer to Figure 7-3). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and is pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

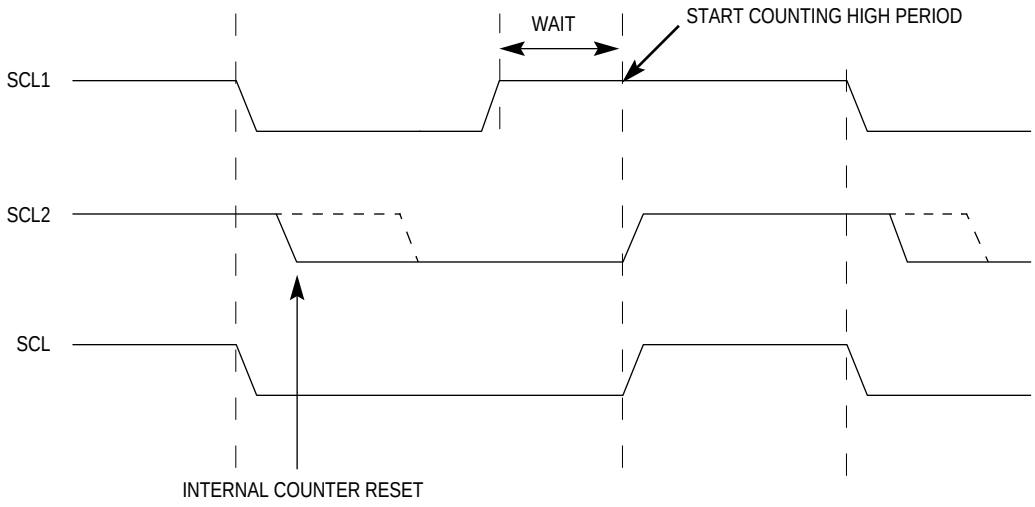


Figure 7-3. M-Bus Clock Synchronization

7.2.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

7.2.9 Clock Stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

7.3 PROGRAMMING MODEL

There are five registers used in the M-bus interface and the internal configuration of these registers is discussed in the following paragraphs. A block diagram of the M-bus module is shown in Figure 7-1.

7.3.1 M-Bus Address Register (MADR)

MADR MBASE+$141

7						1	0
ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	—
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

ADR7–ADR1—Slave Address

This field contains the specific slave address to be used by the M-bus module.

Bit 0—Reserved by Motorola.

7.3.2 M-Bus Frequency Divider Register (MFDR)

MFDR MBASE+$143

7	6	5	4	3	2	1	0
—	—	—	MBC4	MBC3	MBC2	MBC1	MBC0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

Bits 7–5—Reserved by Motorola.

MBC4–MBC0—M-Bus Clock Rate 4–0

This field is used to prescale the clock for bit rate selection. Due to the potential slow rise and fall times of the SCL and SDA signals the bus signals are sampled at the prescaler frequency. This sampling incurs an overhead of 6 clocks per SCL pulse. The serial bit clock frequency is equal to the CPU clock divided by the divider shown in Table 7-1 plus the sampling overhead of 6 clocks per cycle.

For a 16.67 MHz internal operating frequency, the serial bit clock frequency of M-bus ranges from 3825 Hz to 595 kHz.

Table 7-1. M-Bus Prescaler values

MBC4–MBC0	Divider	MBC4–MBC0	Divider
00000	22	10000	352
00001	24	10001	384
00010	28	10010	448
00011	34	10011	544
00100	44	10100	704
00101	48	10101	768
00110	56	10110	896
00111	68	10111	1088
01000	88	11000	1408
01001	96	11001	1536
01010	112	11010	1792
01011	136	11011	2176
01100	176	11100	2816
01101	192	11101	3072
01110	224	11110	3584
01111	272	11111	4352

7.3.3 M-Bus Control Register (MBCR)

MBCR MBASE+$145

7	6	5	4	3	2	1	0
MEN	MIEN	MSTA	MTX	TXAK	RSTA	—	—
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

MEN—M-Bus Enable

This bit controls the software reset of the entire M-bus module.

1 = The M-bus module is enabled. This bit must be set before any other MBCR bits have any effect.

0 = The module is reset and disabled. This is the power-on reset situation. When low the interface is held in reset but registers can still be accessed.

If the M-bus module is enabled in the middle of a byte transfer the interface behaves as follows. Slave mode ignores the current transfer on the bus and start operating whenever a subsequent start condition is detected. Master mode is not aware that the bus is busy, hence if a start cycle is initiated then the current bus cycle may become corrupt. This would ultimately result in either the current bus master or the M-bus module losing arbitration, after which bus operation would return to normal.

MIEN—M-Bus Interrupt Enable

1 = Interrupts from the M-bus module are enabled. An M-bus interrupt occurs provided the MIF bit in the status register is also set.
0 = Interrupts from the M-bus module are disabled. Note that this does not clear any currently pending interrupt condition.

MSTA—Master/Slave Mode Select Bit

Upon reset, this bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus, and the master mode is selected. When this bit is cleared, a STOP signal is generated and the operation mode changes from master to slave.

MSTA is cleared without generating a STOP signal when the master loses arbitration.

1 = Master mode
0 = Slave mode

MTX—Transmit/Receive Mode Select Bit

This bit selects the direction of master and slave transfers. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode this bit should be set according to the type of transfer required. Hence for address cycles this bit is always high.

1 = Transmit
0 = Receive

TXAK—Transmit Acknowledge Enable

This bit specifies the value driven onto SDA during acknowledge cycles for both master and slave receivers.

1 = No acknowledge signal response is sent (i.e., acknowledge bit = 1)
0 = An acknowledge signal is sent out to the bus at the 9th clock bit after receiving one byte data.

RSTA—Repeat Start

Writing a 1 to this bit generates a repeated START condition on the bus, provided we are the current bus master. This bit is always read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, results in loss of arbitration.

1 = Generate repeat start cycle.
0 = Do not generate repeat start cycle.

Bits 1–0—Reserved by Motorola.

7.3.4 M-Bus Status Register (MBSR)

This status register is read-only with exception of the MIF and MAL bits, which are software clearable. All bits are cleared upon reset except the MCF and RXAK bits.

MBSR MBASE+$147

7	6	5	4	3	2	1	0
MCF	MAAS	MBB	MAL	—	SRW	MIF	RXAK
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

MCF—Data Transferring Bit

While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer.

1 = Transfer complete
0 = Transfer in progress

MAAS—Addressed As a Slave Bit

When its own specific address (M-bus address register) is matched with the calling address, this bit is set. The CPU is interrupted provided the MIEN is set. Then the CPU needs to check the SRW bit and set its TX/RX mode accordingly.

Writing to the M-bus control register clears this bit.

1 = Addressed as a slave
0 = Not addressed as a slave

MBB—Bus Busy Bit

This bit indicates the status of the bus. When a START signal is detected, the MBB is set. If a STOP signal is detected, it is cleared.

1 = Bus is busy
0 = Bus is idle

MAL—Arbitration Lost

The arbitration lost bit (MAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances:

1. SDA sampled as low when the master drives a high during an address or data transmit cycle.
2. SDA sampled as a low when the master drives a high during the acknowledge bit of a data receive cycle.
3. A start cycle is attempted when the bus is busy.
4. A repeated start cycle is requested in slave mode.
5. A stop condition is detected when the master did not request it.

This bit must be cleared by software, writing a low to this bit clears it.

Bit 3—Reserved by Motorola.

SRW—Slave Read/Write

When MAAS is set this bit indicates the value of the R/$\overline{W}$ command bit of the calling address sent from master. This bit is only valid when a complete transfer has occurred and no other transfers have been initiated. Checking this bit, the CPU can select slave transmit/receive mode according to the command of master.

1 = Slave transmit, master reading from slave.
0 = Slave receive, master writing to slave.

MIF—M-Bus Interrupt

The MIF bit is set when an interrupt is pending, which causes a processor interrupt request provided MIEN is set. MIF is set when one of the following events occurs:

1. Complete one byte transfer (set at the falling edge of the 9th clock).
2. Receive a calling address that matches its own specific address in slave receive mode.
3. Arbitration lost.

This bit must be cleared by software, writing a low to it, in the interrupt routine.

RXAK—Received Acknowledge

The value of SDA during the acknowledge bit of a bus cycle. If the RXAK is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock.

1 = No acknowledge received
0 = Acknowledge received

7.3.5 M-Bus Data I/O Register (MBDR)

MBDR MBASE+$149

7							0
D7	D6	D5	D4	D3	D2	D1	D0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

In master transmit mode, data written into the MBDR is sent to the bus automatically. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same function is available after it is addressed.

7.4 M-BUS PROGRAMMING EXAMPLES

The following paragraphs provide programming examples for the M-bus module.

7.4.1 Initialization Sequence

Reset puts the M-bus control register to its default status. Before the interface can be used to transfer serial data, an initialization procedure must be carried out:

1. Update MFDR, select division ratio required to obtain SCL frequency from system clock.

2. Update MADR to define its own slave address.
3. Set MEN bit of MBCR to enable the M-bus interface system.
4. Modify the bits of MBCR to select master/slave mode, transmit/receive mode, interrupt enable or not.

7.4.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' mode. If the device is connected to a multi-master bus system, the state of the M-bus busy bit (MBB) must be tested to check whether the serial bus is free.

If the bus is free (MBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB set to indicate the direction of transfer required from the slave.

The bus free time (i.e. the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period it may be necessary to wait until the M-bus is busy after writing the calling address to the MBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of a program which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
CHFLAG  BTST.B      #5,MBSR          ; CHECK THE MBB BIT OF THE
        BNE.S       CHFLAG           ; STATUS REGISTER. IF IT IS
                                     ; SET, WAIT UNTIL IT IS CLEAR
TXSTART BSET.B      #4,MBCR          ; SET TRANSMIT MODE
        BSET.B      #5,MBCR          ; SET MASTER MODE
                                     ; i.e. GENERATE START CONDITION
        MOVE.B      CALLING,MBDR     ; TRANSMIT THE CALLING
                                     ; ADDRESS, D0=R/W
MBFREE  BTST.B      #5,MBSR          ; CHECK THE MBB BIT OF THE
        BEQ.S       MBFREE           ; STATUS REGISTER. IF IT IS
                                     ; CLEAR, WAIT UNTIL IT IS SET
```

7.4.3 Post-Transfer Software Response

Transmission or reception of a byte sets the data transferring bit (MCF) to 1, which indicates one byte communication is finished. Also the M-bus interrupt bit (MIF) is set, an interrupt is generated if the interrupt function is enable during initialization by setting the MIEN bit. Software must clear the MIF bit in the interrupt routine first. The MCF bit is cleared by reading from the MBDR in receive mode or writing to MBDR in transmit mode.

Software may service the M-bus I/O in the main program by monitoring the MIF bit if the interrupt function is disabled. Note that polling should monitor the MIF bit rather than the MCF bit since their operation is different when arbitration is lost.

Note that when an interrupt occurs at the end of the address cycle the master is always in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/$\overline{W}$ bit in MBDR, then the MTX bit should be toggled at this stage.

During slave mode address cycles (MAAS=1) the SRW bit in the status register is read to determine the direction of the subsequent transfer and the MTX bit is programmed accordingly. For slave mode data cycles (MAAS=0) the SRW bit is not valid, the MTX bit in the control register should be read to determine the direction of the current transfer.

The following is an example of a software response by a 'master transmitter' in the interrupt routine (refer to Figure 7-4).

```
ISR             BCLR.B      #1,MBSR             ; CLEAR THE MIF FLAG
                BTST.B      #5,MBCR             ; CHECK THE MSTA FLAG,
                BEQ.S       SLAVE               ; BRANCH IF SLAVE MODE
                BTST.B      #4,MBCR             ; CHECK THE MODE FLAG,
                BEQ.S       RECEIVE             ; BRANCH IF IN RECEIVE MODE
                BTST.B      #0,MBSR             ; CHECK ACK FROM RECEIVER
                BNE.B       END                 ; IF NO ACK, END OF TRANSMISSION
TRANSMIT        MOVE.B      DATABUF,MBDR        ; TRANSMIT NEXT BYTE OF DATA
```

7.4.4 Generation of STOP

A data transfer ends with a STOP signal generated by the master device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following is an example showing how a stop condition is generated by a master transmitter.

```
MASTX           BTST.B      #0,MBSR             ; IF NO ACK, BRANCH TO END
                BNE.B       END
                MOVE.B      TXCNT,D0            ; GET VALUE FROM THE
                                                ; TRANSMITTING COUNTER
                BEQ.S       END                 ; IF NO MORE DATA, BRANCH TO
                                                ; END
                MOVE.B      DATABUF,MBDR        ; TRANSMIT NEXT BYTE OF DATA
                SUBI.B      #1,TXCNT            ; DECREASE THE TXCNT
                BRA.S       EMASTX              ; EXIT
END             BCLR        #5,MBCR             ; GENERATE A STOP CONDITION
EMASTX          RTE                             ; RETURN FROM INTERRUPT
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data which can be done by setting the transmit acknowledge bit (TXAK) before reading the 2nd last byte of data. Before reading the last byte of data, a STOP signal must be generated first. The following is an example showing how a STOP signal is generated by a master receiver.

```
MASR            SUBI.B      #1,RXCNT
                BEQ.S       ENMASR              ; LAST BYTE TO BE READ
                MOVE.B      RXCNT,D1            ; CHECK SECOND LAST BYTE
                SUBI.B      #1,D1               ; TO BE READ
                BNE.S       NXMAR               ; NOT LAST ONE OR SECOND LAST
LAMAR           BSET.B      #3,MBCR             ; SECOND LAST, DISABLE ACK
                                                ; TRANSMITTING
                BRA         NXMAR
```

```
ENMASR      BCLR.B    #5,MBCR          ; LAST ONE, GENERATE 'STOP'
                                       ; SIGNAL
NXMAR       MOVE.B    MBDR,RXBUF       ; READ DATA AND STORE
            RTE
```

7.4.5 Generation of Repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generate a STOP signal. A program example is as shown.

```
RESTART     BSET.B    #2,MBCR          ; ANOTHER START (RESTART)
            MOVE.B    CALLING,MBDR     ; TRANSMIT THE CALLING
                                       ; ADDRESS, D0=R/W-
```

7.4.6 Slave Mode

In slave interrupt service routine, the master addressed as slave bit (MAAS) should be tested to check if a calling of its own address has just been received (refer to Figure 7-4). If MAAS is set, software should set the transmit/receive mode select bit (MTX bit of MBCR) according to the R/$\overline{W}$ command bit (SRW). Writing to the MBCR clears the MAAS automatically. Note that the only time MAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred; interrupts resulting from subsequent data transfers clear MAAS. A data transfer may now be initiated by writing information to MBDR, for slave transmits, or dummy reading from MBDR, in slave receive mode. The slave drives SCL low between byte transfers, SCL is released when the MBDR is accessed in the required mode.

In slave transmitter routine, RXAK must be tested before transmitting next byte of data. When RXAK is set, this signals 'end of data' from the master receiver, which must then switch from transmitter mode to receiver mode by software. This is followed by a dummy read, which releases the SCL line so that the master can generate a STOP signal.

7.4.7 Arbitration Lost

Only one master can engage the bus at any one time. Those devices wishing to engage the bus, but having lost arbitration, switch immediately to slave receive mode by hardware. Their data output to the SDA line is stopped, but the SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with MAL=1 and MSTA=0. If one master attempts to start transmission while the bus is being engaged by another master, the hardware inhibits the transmission; the MSTA bit is cleared without generating a STOP condition; an interrupt to CPU is generated and MAL is set to indicate that the attempt to engage the bus failed. In these cases, the slave service routine should test MAL first; MAL should be cleared by software if it is set.

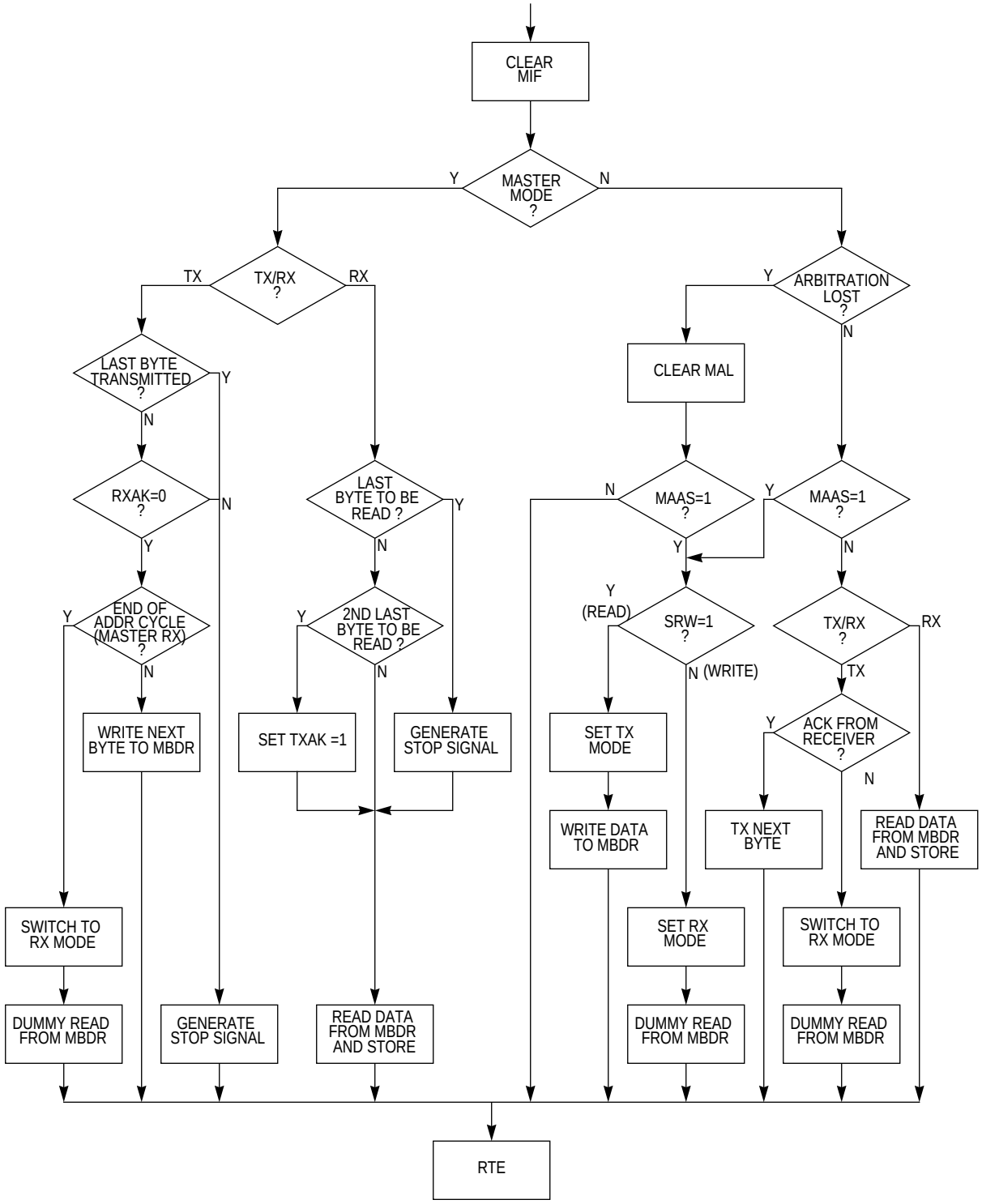


Figure 7-4. Flow-Chart of Typical M-Bus Interrupt Routine

SECTION 8
SERIAL MODULE

The MC68307 serial module is a universal asynchronous/synchronous receiver/transmitter that interfaces directly to the CPU. The serial module, shown in Figure 8-1, consists of the following major functional areas:

- Serial Communication Channel
- Sixteen Bit Timer for Baud Rate Generation
- Internal Channel Control Logic
- Interrupt Control Logic

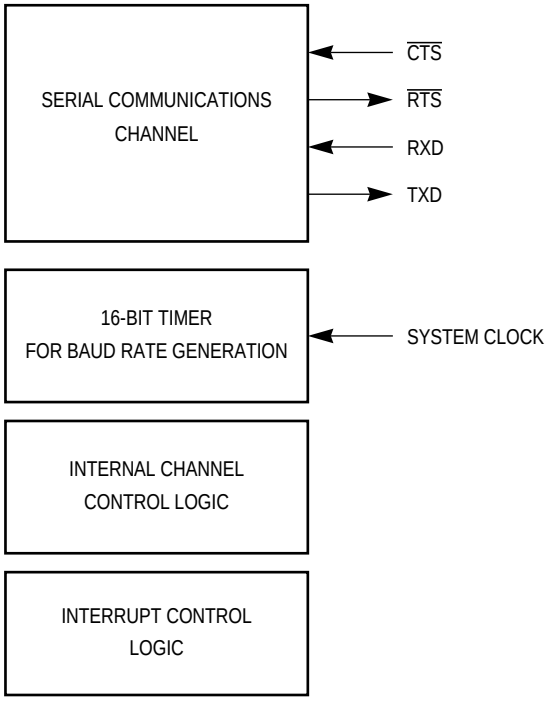


Figure 8-1. Simplified Block Diagram

8.1 MODULE OVERVIEW

Features of the serial module are as follows:

- Full-Duplex Asynchronous/Synchronous Receiver/Transmitter Channel
- Quadruple-Buffered Receiver
- Double-Buffered Transmitter
- Independently Programmable Baud Rate for Receiver and Transmitter Selectable from:
  — Timer-Generated Baud Rate Up to 260 kbaud
- Programmable Data Format:
  — Five to Eight Data Bits Plus Parity
  — Odd, Even, No Parity, or Force Parity
  — Nine-Sixteenths to Two Stop Bits Programmable in One-Sixteenth Bit Increments
- Programmable Channel Modes:
  — Normal (Full Duplex)
  — Automatic Echo
  — Local Loopback
  — Remote Loopback
- Automatic Wakeup Mode for Multidrop Applications
- Eight Maskable Interrupt Conditions
- Parity, Framing, and Overrun Error Detection
- False-Start Bit Detection
- Line-Break Detection and Generation
- Detection of Breaks Originating in the Middle of a Character
- Start/End Break Interrupt/Status

8.1.1 Serial Communication Channel

The communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter using an operating frequency derived from the system clock.

The transmitter accepts parallel data from the bus, converts it to a serial bit stream, inserts the appropriate start, stop, and optional parity bits, then outputs a composite serial data stream on the channel transmitter serial data output (TxD). Refer to **Section 8.3.2.1 Transmitter** for additional information.

The receiver accepts serial data on the channel receiver serial data input (RxD), converts it to parallel format, checks for a start bit, stop bit, parity (if any), or break condition, and transfers the assembled character onto the bus during read operations. Refer to **Section 8.3.2.2 Receiver** for additional information.

8.1.2 Baud Rate Generator Logic

The Baud rate Generator logic consists of an internal UART clock input which is not available as an external pin, but is a derivative of the IMBP system clock, a baud-rate generator (BRG) and a programmable 16-bit timer. The clock serves as the basic timing reference for the baud-rate generator and other internal circuits.

The 16-bit timer is used to produce a 16X clock for any baud rate by counting down the system clock, it acts as a programmable divider. This feature is especially useful for non-standard system clock frequencies and baud rates.

8.1.3 Baud Rate Generator/Timer

The 16-bit timer is used as a baud rate generator, and provides a synchronous clock mode of operation when used as a divide-by-1 clock and an asynchronous clock mode when used as a divide-by-16 clock. This allows flexible baud rates for various system clock rates of the MC68307, the divisor value being directly programmable.

8.1.4 Interrupt Control Logic

An internal interrupt request signal ($\overline{IRQ}$) is provided to notify the MC68307 interrupt controller of an interrupt condition. The output is the logical NOR of all (up to eight) unmasked interrupt status bits in the interrupt status register (UISR).

The interrupt level of the serial module is programmed in the MC68307 interrupt controller external to the serial module. When an interrupt at this level is acknowledged, the serial module provides an automatic vector if UART is highest priority at this level.

8.1.5 Comparison of Serial Module to MC68681

The MC68307 is code compatible with the MC68681, except that only channel A is implemented, and the timer/counter is used as a baud rate clock, dividing the MC68307 clock frequency. The input and output port lines (IPx and OPx) are not implemented except for $\overline{CTS}$ and $\overline{RTS}$ functions.

8.2 SERIAL MODULE SIGNAL DEFINITIONS

The following paragraphs contain a brief description of the serial module signals. Figure 8-2 shows both the external and internal signal groups.

NOTE

The terms *assertion* and *negation* are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term *assert* or *assertion* indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term *negate* or *negation* indicates that a signal is inactive or false.

8.2.1 Transmitter Serial Data Output (TxD)

This signal is the transmitter serial data output. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal on the falling edge of the clock source, with the least significant bit transmitted first.

Note that in order to use the TxD signal, the MC68307 port B control register must be set up to enable the corresponding I/O pin for this function. By default this signal functions as port B bit 2.

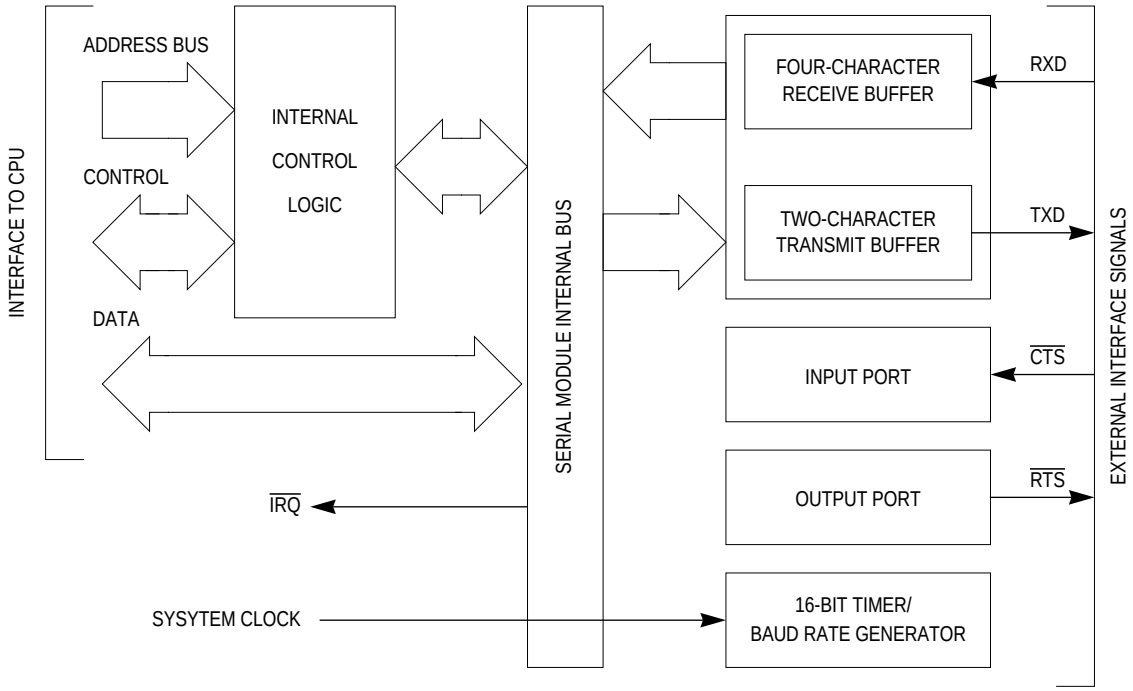


Figure 8-2. External and Internal Interface Signals

8.2.2 Receiver Serial Data Input (RxD)

This signal is the receiver serial data input. Data received on this signal is sampled on the rising edge of the clock source, with the least significant bit received first.

Note that in order to use the RxD, the MC68307 port B control register must be set up to enable the corresponding I/O pin for this function. By default this signal functions as port B bit 3.

8.2.3 Request-To-Send ($\overline{\text{RTS}}$)

This active-low output signal can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send ($\overline{\text{CTS}}$) input of a transmitter, this signal can be used to control serial data flow.

8.2.4 Clear-To-Send ($\overline{CTS}$)

This active-low input is the clear-to-send input. It can generate an interrupt on change-of-state.

Note that in order to use the $\overline{RTS}$ or $\overline{CTS}$ signals, the MC68307 port B control register must be set up to enable the corresponding I/O pins for these functions. By default these signals function as port B bits 4 and 5 respectively.

8.3 OPERATION

The following paragraphs describe the operation of the baud rate generator, transmitter and receiver, and other functional operating modes of the serial module.

8.3.1 Baud Rate Generator/Timer

The baud rate generator (see Figure 8-3) consists of a 16-bit timer clocked by the MC68307 system clock. Baud rates are selected by programming a divide value into the 16-bit timer.

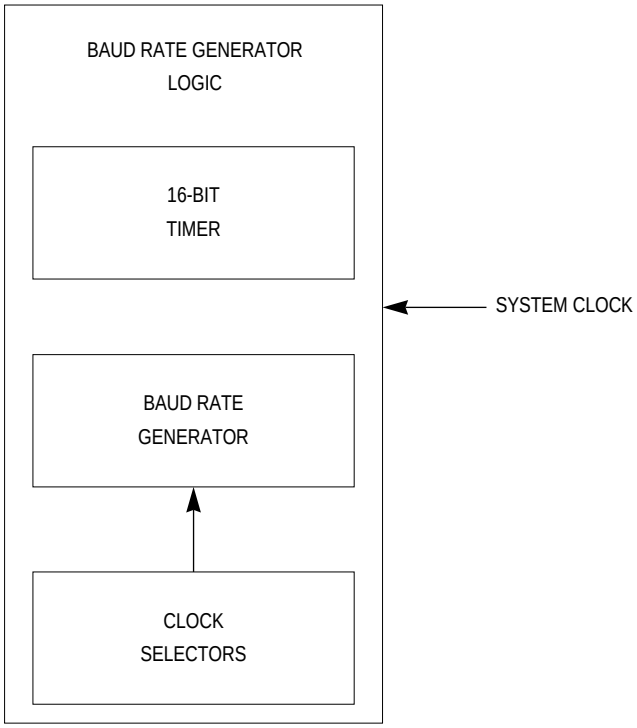


Figure 8-3. Baud Rate Generator Block Diagram

8.3.2 Transmitter and Receiver Operating Modes

The functional block diagram of the transmitter and receiver, including command and operating registers, is shown in Figure 8-4. The paragraphs that follow contain descriptions for both these functions in reference to this diagram. For detailed register information, refer to **Section 8.4 Register Description and Programming**

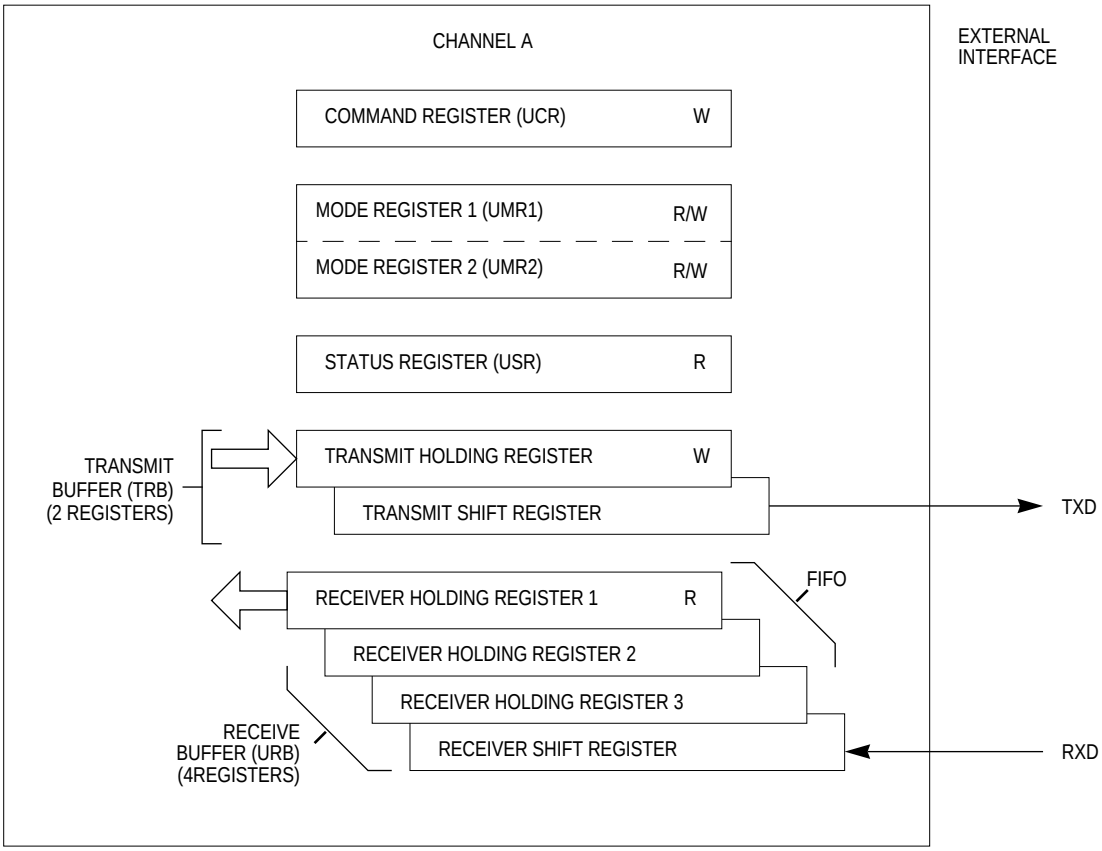


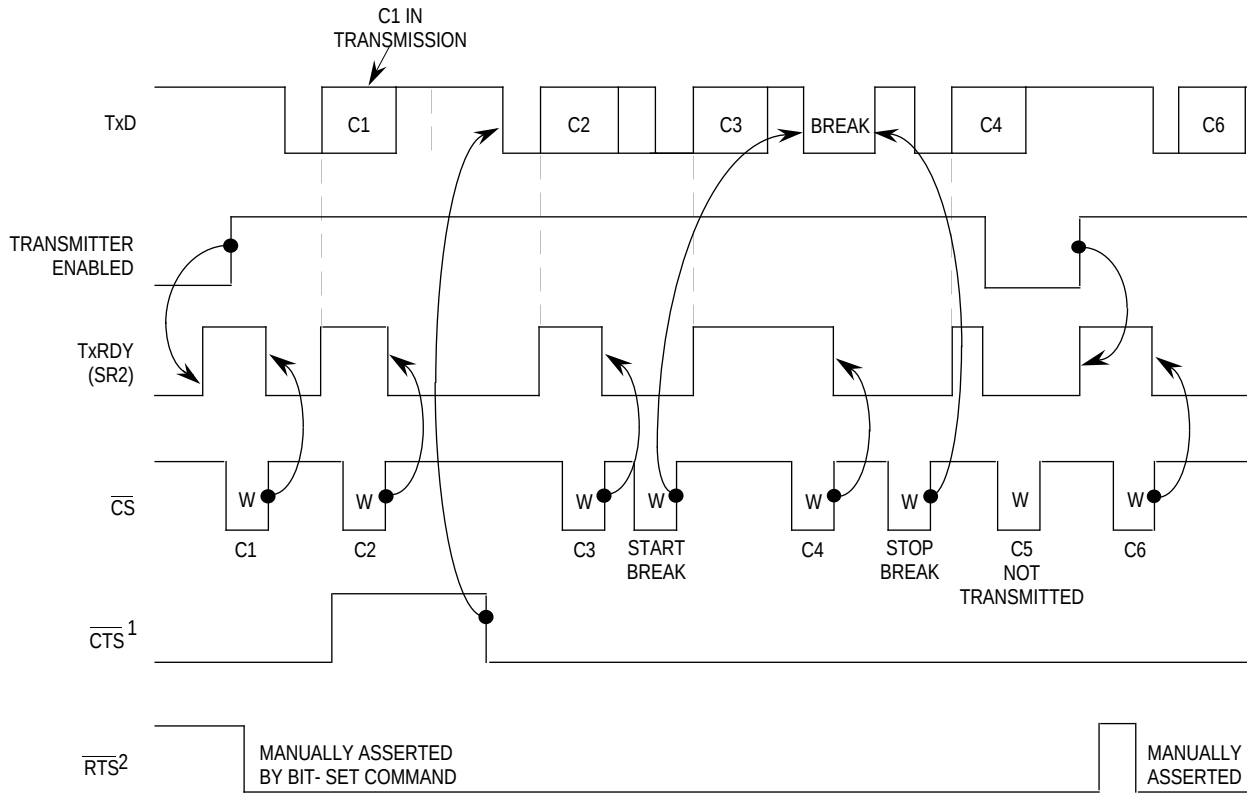
Figure 8-4. Transmitter and Receiver Functional Diagram

8.3.2.1 TRANSMITTER. The transmitter is enabled through its command register (UCR) located within the serial module. The serial module signals the CPU when it is ready to accept a character by setting the transmitter-ready bit (TxRDY) in the UART status register (USR). Functional timing information for the transmitter is shown in Figure 8-5.

The transmitter converts parallel data from the CPU to a serial bit stream on TxD. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The least significant bit is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

Following transmission of the stop bits, if a new character is not available in the transmitter holding register, the TxD output remains high ('mark' condition), and the transmitter empty bit (TxEMP) in the USR is set. Transmission resumes and the TxEMP bit is cleared when the CPU loads a new character into the transmitter buffer (UTB). If a disable command is sent to the transmitter, it continues operating until the character in the transmit shift register, if any, is completely sent out. If the transmitter is reset through a software command, operation ceases immediately (refer to **Section 8.4.1.5 Command Register (UCR)**). The transmitter is re-enabled through the UCR to resume operation after a disable or software reset.

If clear-to-send operation is enabled, $\overline{\text{CTS}}$ must be asserted for the character to be transmitted. If $\overline{\text{CTS}}$ is negated in the middle of a transmission, the character in the shift register



NOTES:
1. TIMING SHOWN FOR UMR2(4) = 1
2. TIMING SHOWN FOR UMR2(5) = 1
3. C_N = TRANSMIT CHARACTER
4. W = WRITE

Figure 8-5. Transmitter Timing Diagram

is transmitted, and TxD remains in the 'mark' state until $\overline{CTS}$ is asserted again. If the transmitter is forced to send a continuous low condition by issuing a send break command, the state of $\overline{CTS}$ is ignored by the transmitter.

The transmitter can be programmed to automatically negate the request-to-send ($\overline{RTS}$) output upon completion of a message transmission. If the transmitter is programmed to operate in this mode, $\overline{RTS}$ must be manually asserted before a message is transmitted. In applications in which the transmitter is disabled after transmission is complete and $\overline{RTS}$ is appropriately programmed, $\overline{RTS}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually re-enabled by reasserting $\overline{RTS}$ before the next message is to be sent.

8.3.2.2 RECEIVER. The receiver is enabled through its UCR located within the serial module. Functional timing information for the receiver is shown in Figure 8-6. The receiver looks for a high-to-low (mark-to-space) transition of the start bit on RxD. When a transition is detected, the state of RxD is sampled each 16× clock for eight clocks, starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If RxD is sampled high, the start bit is invalid, and the search for the valid start bit begins again. If RxD is still low, a valid start bit is assumed, and the receiver continues to sample the input at one-bit time intervals, at the theoretical center of the bit, until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the RxD input is sampled on the rising edge of the programmed clock source. The least significant bit is received first. The data is then transferred to a receiver holding register, and the RxRDY bit in the USR is set. If the character length is less than eight bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, the receiver immediately looks for the next start bit. However, if a non-zero character is received without a stop bit (framing error) and RxD remains low for one-half of the bit period after the stop bit is sampled, the receiver operates as if a new start bit is detected. The parity error (PE), framing error (FE), overrun error (OE), and received break (RB) conditions (if any) set error and break flags in the USR at the received character boundary and are valid only when the RxRDY bit in the USR is set.

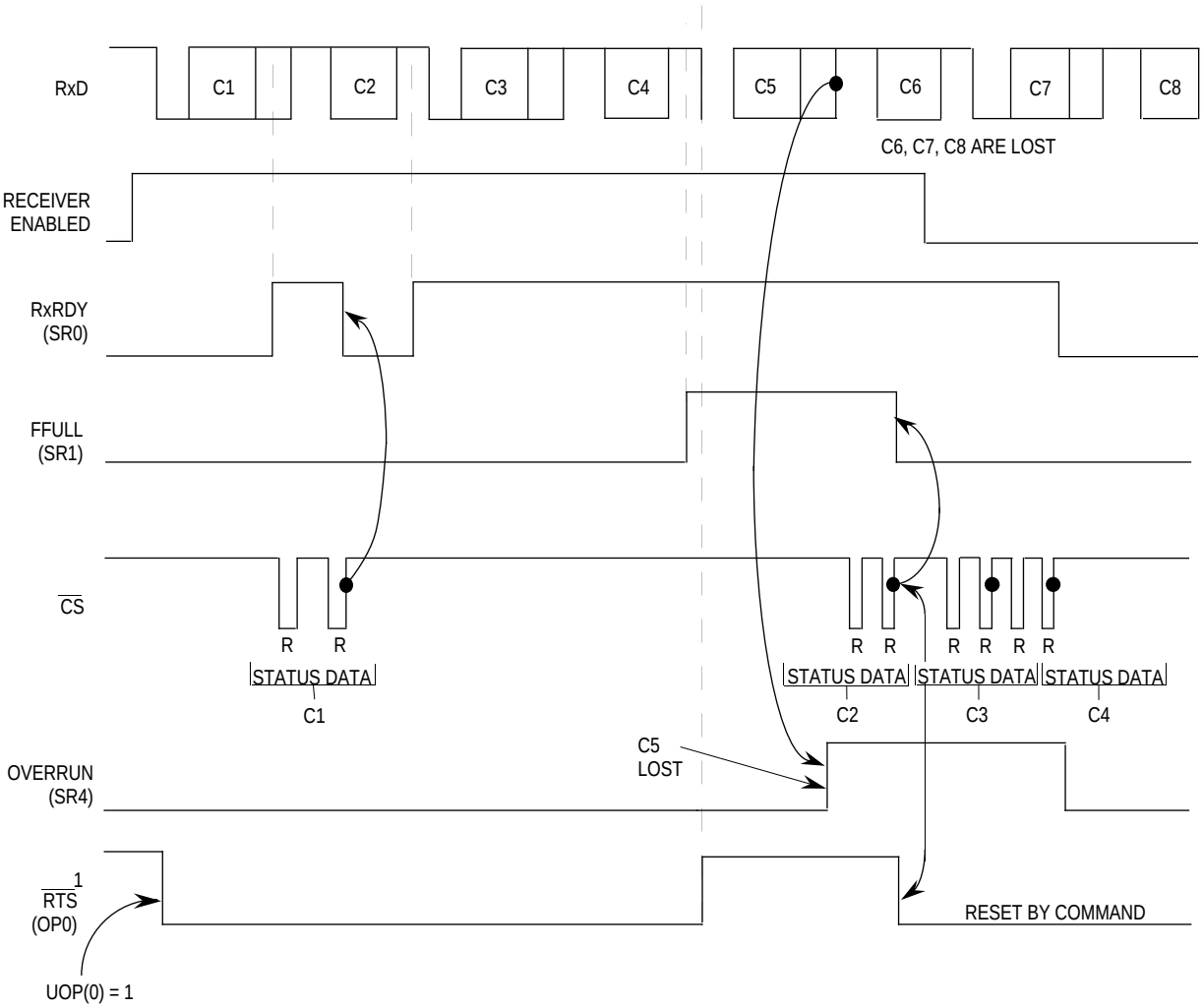
If a break condition is detected (RxD is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register, and the RB and RxRDY bits in the USR are set. The RxD signal must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. When the break begins in the middle of a character, the receiver places the damaged character in the receiver first-in-first-out (FIFO) stack and sets the corresponding error conditions and RxRDY bit in the USR. Then, if the break persists until the next character time, the receiver places an all-zero character into the receiver FIFO and sets the corresponding RB and RxRDY bits in the USR.

8.3.2.3 FIFO STACK. The FIFO stack is used in the UART's receiver buffer logic. The stack consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the RxD (refer to Figure 8-4). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Thus, data flowing from the receiver to the CPU is quadruple buffered.

In addition to the data byte, three status bits, PE, FE, and RB, are appended to each data character in the FIFO; OE is not appended. By programming the ERR bit in the channel's mode register (UMR1), status is provided in character or block modes.

The RxRDY bit in the USR is set whenever one or more characters are available to be read by the CPU. A read of the receiver buffer produces an output of data from the top of the FIFO stack. After the read cycle, the data at the top of the FIFO stack and its associated status bits are 'popped', and new data can be added at the bottom of the stack by the receiver shift



NOTES:
1. Timing shown for MR1(7) = 1
2. Timing shown for MR1(6) = 0
3. R = Read
4. C_N = Received Character

Figure 8-6. Receiver Timing Diagram

register. The FIFO-full status bit (FFULL) is set if all three stack positions are filled with data. Either the RxRDY or FFULL bit can be selected to cause an interrupt.

In the character mode, status provided in the USR is given on a character-by-character basis and thus applies only to the character at the top of the FIFO. In the block mode, the status provided in the USR is the logical OR of all characters coming to the top of the FIFO stack since the last reset error command. A continuous logical OR function of the corresponding status bits is produced in the USR as each character reaches the top of the FIFO stack. The block mode is useful in applications where the software overhead of checking each character's error cannot be tolerated. In this mode, entire messages are received, and only one data integrity check is performed at the end of the message. This mode allows a data-reception speed advantage, but does have a disadvantage since each character is not individually checked for error conditions by software. If an error occurs within the message,

the error is not recognized until the final check is performed, and no indication exists as to which character in the message is at fault.

In either mode, reading the USR does not affect the FIFO. The FIFO is 'popped' only when the receive buffer is read. The USR should be read prior to reading the receive buffer. If all three of the FIFO's receiver holding registers are full when a new character is received, the new character is held in the receiver shift register until a FIFO position is available. If an additional character is received during this state, the contents of the FIFO are not affected. However, the character previously in the receiver shift register is lost, and the OE bit in the USR is set when the receiver detects the start bit of the new overrunning character.

To support control flow capability, the receiver can be programmed to automatically negate and assert $\overline{\text{RTS}}$. When in this mode, $\overline{\text{RTS}}$ is automatically negated by the receiver when a valid start bit is detected and the FIFO stack is full. When a FIFO position becomes available, $\overline{\text{RTS}}$ is asserted by the receiver. Using this mode of operation, overrun errors are prevented by connecting the $\overline{\text{RTS}}$ to the $\overline{\text{CTS}}$ input of the transmitting device.

Note that in order to use the $\overline{\text{RTS}}$ or $\overline{\text{CTS}}$ signals, the MC68307 port B control register must be set up to enable the corresponding I/O pins for these functions. By default these signals function as port B bits 4 and 5 respectively.

If the FIFO stack contains characters and the receiver is disabled, the characters in the FIFO can still be read by the CPU. If the receiver is reset, the FIFO stack and all receiver status bits, corresponding output ports, and interrupt request are reset. No additional characters are received until the receiver is re-enabled.

8.3.3 Looping Modes

The UART can be configured to operate in various looping modes as shown in Figure 8-7. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs with further information available in **Section 8.4 Register Description and Programming**.

The UART's transmitter and receiver should both be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

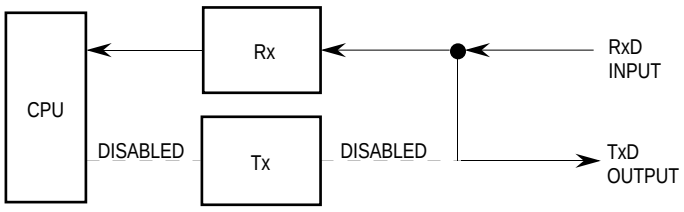
8.3.3.1 AUTOMATIC ECHO MODE. In this mode, the UART automatically retransmits the received data on a bit-by-bit basis. The local CPU-to-receiver communication continues normally, but the CPU-to-transmitter link is disabled. While in this mode, received data is clocked on the receiver clock and retransmitted on TxD. The receiver must be enabled, but the transmitter need not be enabled.

Since the transmitter is not active, the TxEMP and TxRDY bits in USR are inactive, and data is transmitted as it is received. Received parity is checked, but not recalculated for transmission. Character framing is also checked, but stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.

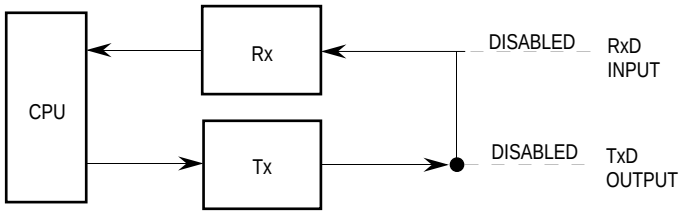
8.3.3.2 LOCAL LOOPBACK MODE. In this mode, TxD is internally connected to RxD. This mode is useful for testing the operation of a local serial module channel by sending data to the transmitter and checking data assembled by the receiver. In this manner, correct channel operations can be assured. Also, both transmitter and CPU-to-receiver communications continue normally in this mode. While in this mode, the RxD input data is ignored, the TxD is held marking, and the receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be enabled.

8.3.3.3 REMOTE LOOPBACK MODE. In this mode, the channel automatically transmits received data on the TxD output on a bit-by-bit basis. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote channel. While in this mode, the receiver clock is used for the transmitter.

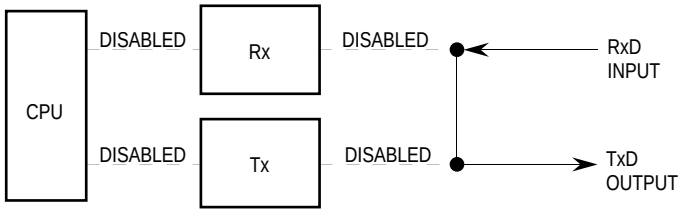
Since the receiver is not active, received data cannot be read by the CPU, and the error status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.



(a) Automatic Echo



(b) Local Loopback



(c) Remote Loopback

Figure 8-7. Looping Modes Functional Diagram

8.3.4 Multidrop Mode

The UART can be programmed to operate in a wakeup mode for multidrop or multiprocessor applications. Functional timing information for the multidrop mode is shown in Figure 8-8. The mode is selected by setting bits 3 and 4 in mode register 1 (UMR1). This mode of operation allows the master station to be connected to several slave stations (maximum of 256). In this mode, the master transmits an address character followed by a block of data characters targeted for one of the slave stations. The slave stations have their channel receivers disabled. However, they continuously monitor the data stream sent out by the master station. When an address character is sent by the master, the slave receiver channel notifies its respective CPU by setting the RxRDY bit in the USR and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Slave stations not addressed continue to monitor the data stream for the next address character. Data fields in the data stream are separated by an address character. After a slave receives a block of data, the slave station's CPU disables the receiver and initiates the process again.

A transmitted character from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. The A/D bit identifies the type of character being transmitted to the slave station. The character is interpreted as an address character if the A/D bit is set or as a data character if the A/D bit is cleared. The polarity of the A/D bit is selected by programming bit 2 of UMR1. UMR1 should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RxRDY bit and loads the character into the receiver holding register FIFO stack provided the received A/D bit is a one (address tag). The character is discarded if the received A/D bit is a zero (data tag). If the receiver is enabled, all received characters are transferred to the CPU via the receiver holding register stack during read operations.

In either case, the data bits are loaded into the data portion of the stack while the A/D bit is loaded into the status portion of the stack normally used for a parity error (USR bit 5). Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may still contain error detection and correction information. One way to provide error detection, if 8-bit characters are not required, is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

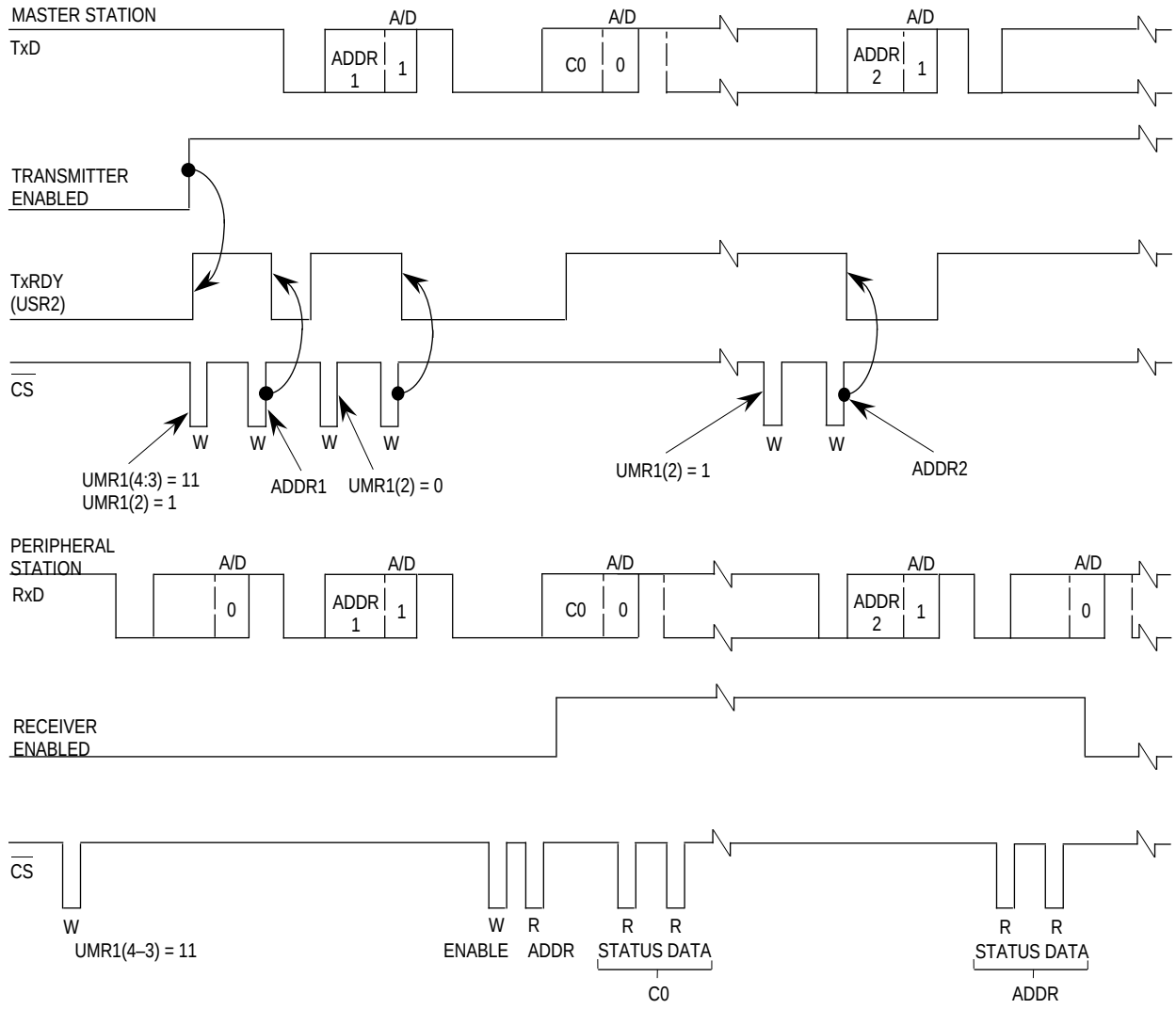


Figure 8-8. Multidrop Mode Timing Diagram

8.3.5 Bus Operation

This section describes the operation of the bus during read, write, and interrupt acknowledge cycles to the serial module. All serial module registers must be accessed as bytes.

8.3.5.1 READ CYCLES. The serial module is accessed by the CPU with zero wait states, as the MC68307 system clock is also used for the serial module. The serial module responds to reads with byte data on D7–D0. Reserved registers return logic zero during reads.

8.3.5.2 WRITE CYCLES. The serial module is accessed by the CPU with zero wait states. The serial module accepts write data on D7–D0. Write cycles to read-only registers and reserved registers complete in a normal manner without exception processing; however, the data is ignored.

8.3.5.3 INTERRUPT ACKNOWLEDGE CYCLES. The serial module is capable of arbitrating for interrupt servicing and supplying the interrupt vector when it has successfully won arbitration. The vector number must be provided if interrupt servicing is necessary; thus, the interrupt vector register (UIVR) must be initialized. If the UIVR is not initialized, a spurious interrupt exception is taken if interrupts are generated. This works in conjunction with the MC68307 interrupt controller, which allows a programmable IPL for the interrupt.

8.4 REGISTER DESCRIPTION AND PROGRAMMING

This section contains a detailed description of each register and its specific function as well as flowcharts of basic serial module programming.

8.4.1 Register Description

The operation of the serial module is controlled by writing control bytes into the appropriate registers. A list of serial module registers and their associated addresses is shown in Table 8-1.

NOTE

All serial module registers are only accessible as bytes. The contents of the mode registers (UMR1 and UMR2), clock-select register (UCSR), and the auxiliary control register (UACR) bit 7 should only be changed after the receiver/transmitter is issued a software RESET command—i.e., channel operation must be disabled. Care should also be taken if the register contents are changed during receiver/transmitter operations, as undesirable results may be produced.

In the registers discussed in the following pages, the numbers above the register description represent the bit position in the register. The register description contains the mnemonic for the bit. The values shown below the register description are the values of those register bits after a hardware reset. A value of U indicates that the bit value is unaffected by reset. The read/write status is shown in the last line.

Table 8-1. Serial Module Programming Model

Address	Register Read (R/W = 1)	Register Write (R/W = 0)
MBASE+$101	MODE REGISTER (UMR1, UMR2)	MODE REGISTER (UMR1, UMR2)
MBASE+$103	STATUS REGISTER (USR)	CLOCK-SELECT REGISTER (UCSR)
MBASE+$105	DO NOT ACCESS[1]	COMMAND REGISTER (UCR)
MBASE+$107	RECEIVER BUFFER (URB)	TRANSMITTER BUFFER (UTB)
MBASE+$109	INPUT PORT CHANGE REGISTER (UIPCR)	AUXILIARY CONTROL REGISTER (UACR)
MBASE+$10B	INTERRUPT STATUS REGISTER (UISR)	INTERRUPT MASK REGISTER (UIMR)
MBASE+$10D	BAUD RATE GENERATOR PRESCALE MSB (UBG1)	
MBASE+$10F	BAUD RATE GENERATOR PRESCALE LSB (UBG2)	
	DO NOT ACCESS[1]	
MBASE+$119	INTERRUPT VECTOR REGISTER (UIVR)	INTERRUPT VECTOR REGISTER (UIVR)
MBASE+$11B	INPUT PORT REGISTER (UIP)	DO NOT ACCESS[1]
MBASE+$11D	DO NOT ACCESS[1]	OUTPUT PORT BIT SET CMD (UOP1)[2]
MBASE+$11F	DO NOT ACCESS[1]	OUTPUT PORT BIT RESET CMD (UOP0)[2]

NOTES
1. This address is used for factory testing and should not be read. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.
2. Address-triggered commands.

8.4.1.1 MODE REGISTER 1 (UMR1). UMR1 controls some of the serial module configuration. This register can be read or written at any time. It is accessed when the mode register pointer points to UMR1. The pointer is set to UMR1 by RESET or by a set pointer command, using the control register. After reading or writing UMR1, the pointer points to UMR2.

UMR1 MBASE + $101

7	6	5	4	3	2	1	0
RxRTS	RxIRQ	ERR	PM1	PM0	PT	B/C1	B/C0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

RxRTS—Receiver Request-to-Send Control

1 = Upon receipt of a valid start bit, $\overline{RTS}$ is negated if the UART's FIFO is full. $\overline{RTS}$ is reasserted when the FIFO has an empty position available.

0 = The receiver has no effect on $\overline{RTS}$.

This feature can be used for flow control to prevent overrun in the receiver by using the $\overline{RTS}$ output to control the $\overline{CTS}$ input of the transmitting device. If both the receiver and transmitter are programmed for $\overline{RTS}$ control, $\overline{RTS}$ control is disabled for both since this configuration is incorrect. See **Section 8.4.1.2 Mode Register 2 (UMR2)** for information on programming the transmitter $\overline{RTS}$ control.

RxIRQ—Receiver Interrupt Select

1 = FFULL is the source that generates IRQ.
0 = RxRDY is the source that generates IRQ.

ERR—Error Mode

This bit controls the meaning of the three FIFO status bits (RB, FE, and PE) in the USR.

1 = Block mode—The values in the channel USR are the accumulation (i.e., the logical OR) of the status for all characters coming to the top of the FIFO since the last reset error status command for the channel was issued. Refer to **Section 8.4.1.5 Command Register (UCR)** for more information on serial module commands.
0 = Character mode—The values in the channel USR reflect the status of the character at the top of the FIFO.

NOTE

ERR = 0 must be used to get the correct A/D flag information when in multidrop mode.

PM1–PM0—Parity Mode

These bits encode the type of parity used for the channel (see Table 8-2). The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. These bits can alternatively select multidrop mode for the channel.

PT—Parity Type

This bit selects the parity type if parity is programmed by the parity mode bits, and if multidrop mode is selected, it configures the transmitter for data character transmission or address character transmission. Table 8-2 lists the parity mode and type or the multidrop mode for each combination of the parity mode and the parity type bits.

Table 8-2. PMx and PT Control Bits

PM1	PM0	Parity Mode	PT	Parity Type
0	0	With Parity	0	Even Parity
0	0	With Parity	1	Odd Parity
0	1	Force Parity	0	Low Parity
0	1	Force Parity	1	High Parity
1	0	No Parity	X	No Parity
1	1	Multidrop Mode	0	Data Character
1	1	Multidrop Mode	1	Address Character

B/C1–B/C0—Bits per Character

These bits select the number of data bits per character to be transmitted. The character length listed in Table 8-3 does not include start, parity, or stop bits.

Table 8-3. B/Cx Control Bits

B/C1	B/C0	Bits/Character
0	0	Five Bits
0	1	Six Bits
1	0	Seven Bits
1	1	Eight Bits

8.4.1.2 MODE REGISTER 2 (UMR2). UMR2 controls some of the serial module configuration. It is accessed when the mode register pointer points to UMR2, which occurs after any access to UMR1. Accesses to UMR2 do not change the pointer.

UMR2 MBASE + $101

7	6	5	4	3	2	1	0
CM1	CM0	TxRTS	TxCTS	SB3	SB2	SB1	SB0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

CM1–CM0—Channel Mode

These bits select a channel mode as listed in Table 8-4. See **Section 8.3.3 Looping Modes** for more information on the individual modes.

Table 8-4. CMx Control Bits

CM1	CM0	Mode
0	0	Normal
0	1	Automatic Echo
1	0	Local Loopback
1	1	Remote Loopback

TxRTS—Transmitter Ready-to-Send

This bit controls the negation of the $\overline{\text{RTS}}$ signal.

1 = In applications where the transmitter is disabled after transmission is complete, setting this bit causes the particular OP bit to be cleared automatically one bit time after the characters, if any, in the channel transmit shift register and the transmitter holding register are completely transmitted, including the programmed number of stop bits. This feature is used to automatically terminate transmission of a message. If both the receiver and the transmitter in the same channel are programmed for $\overline{\text{RTS}}$ control, $\overline{\text{RTS}}$ control is disabled for both since this is an incorrect configuration.

0 = The transmitter has no effect on $\overline{\text{RTS}}$.

TxCTS—Transmitter Clear-to-Send

1 = Enables clear-to-send operation. The transmitter checks the state of the $\overline{CTS}$ input each time it is ready to send a character. If $\overline{CTS}$ is asserted, the character is transmitted. If $\overline{CTS}$ is negated, the channel TxD remains in the high state, and the transmission is delayed until $\overline{CTS}$ is asserted. Changes in $\overline{CTS}$ while a character is being transmitted do not affect transmission of that character. If both TxCTS and TxRTS are enabled, TxCTS controls the operation of the transmitter.

0 = The $\overline{CTS}$ has no effect on the transmitter.

SB3–SB0—Stop-Bit Length Control

These bits select the length of the stop bit appended to the transmitted character as listed in Table 8-5. Stop-bit lengths of nine-sixteenth to two bits, in increments of one-sixteenth bit, are programmable for character lengths of six, seven, and eight bits. For a character length of five bits, one and one-sixteenth to two bits are programmable in increments of one-sixteenth bit. In all cases, the receiver only checks for a high condition at the center of the first stop-bit position—i.e., one bit time after the last data bit or after the parity bit, if parity is enabled.

If an external 1× clock is used for the transmitter, UMR2 bit 3 = 0 selects one stop bit, and UMR2 bit 3 = 1 selects two stop bits for transmission.

Table 8-5. SBx Control Bits

SB3	SB2	SB1	SB0	Length 6-8 Bits	Length 5 Bits
0	0	0	0	0.563	1.063
0	0	0	1	0.625	1.125
0	0	1	0	0.688	1.188
0	0	1	1	0.750	1.250
0	1	0	0	0.813	1.313
0	1	0	1	0.875	1.375
0	1	1	0	0.938	1.438
0	1	1	1	1.000	1.500
1	0	0	0	1.563	1.563
1	0	0	1	1.625	1.625
1	0	1	0	1.688	1.688
1	0	1	1	1.750	1.750
1	1	0	0	1.813	1.813
1	1	0	1	1.875	1.875
1	1	1	0	1.938	1.938
1	1	1	1	2.000	2.000

8.4.1.3 STATUS REGISTER (USR). The USR indicates the status of the characters in the FIFO and the status of the transmitter and receiver.

USR MBASE + $103

7	6	5	4	3	2	1	0
RB	FE	PE	OE	TxEMP	TxRDY	FFULL	RxRDY
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor or User

RB—Received Break

1 = An all-zero character of the programmed length has been received without a stop bit. The RB bit is only valid when the RxRDY bit is set. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until RxD returns to the high state for at least one-half bit time, which is equal to two successive edges of the internal or external 1× clock or 16 successive edges of the external 16× clock.
The received break circuit detects breaks that originate in the middle of a received character. However, if a break begins in the middle of a character, it must persist until the end of the next detected character time.

0 = No break has been received.

FE—Framing Error

1 = A stop bit was not detected when the corresponding data character in the FIFO was received. The stop-bit check is made in the middle of the first stop-bit position. The bit is valid only when the RxRDY bit is set.

0 = No framing error has occurred.

PE—Parity Error

1 = When the with parity or force parity mode is programmed in the UMR1, the corresponding character in the FIFO was received with incorrect parity. When the multidrop mode is programmed, this bit stores the received A/D bit. This bit is valid only when the RxRDY bit is set.

0 = No parity error has occurred.

OE—Overrun Error

1 = One or more characters in the received data stream have been lost. This bit is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, framing error status, and parity error, if any, are lost. This bit is cleared by the reset error status command in the UCR.

0 = No overrun has occurred.

TxEMP—Transmitter Empty

1 = The transmitter has underrun (both the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission.

0 = The transmitter buffer is not empty. Either a character is currently being shifted out, or the transmitter is disabled. The transmitter is enabled/disabled by programming the TCx bits in the UCR.

TxRDY—Transmitter Ready

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted.

0 = The transmitter holding register was loaded by the CPU, or the transmitter is disabled.

FFULL—FIFO Full

1 = A character has been received and is waiting in the receiver buffer FIFO.

0 = The FIFO is not full, but may contain up to two unread characters.

RxRDY—Receiver Ready

1 = One or more characters has been received and is waiting in the receiver buffer FIFO.

0 = The CPU has read the receiver buffer, and no characters remain in the FIFO after this read.

8.4.1.4 CLOCK-SELECT REGISTER (UCSR). The UCSR selects the baud rate clock for the receiver and transmitter. The baud rates listed in Table 8-6 and Table 8-7 are only valid if an appropriate frequency is used for the MC68307 system clock, for example, 3.6864 MHz. This is not generally the case, and so a more generic method of baud rate generation is provided by the 16-bit timer. In order to use this mode, program the UCSR to the value $DD.

UCSR MBASE + $103

7	6	5	4	3	2	1	0
RCS3	RCS2	RCS1	RCS0	TCS3	TCS2	TCS1	TCS0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor or User

RCS3–RCS0—Receiver Clock Select

These bits select the baud rate clock for the receiver from a set of baud rates listed in Table 8-6. The baud rate set selected depends upon the auxiliary control register (UACR) bit 7. Set 1 is selected if UACR bit 7 = 0, and set 2 is selected if UACR bit 7 = 1. The receiver clock is always 16 times the baud rate shown in this list.

Table 8-6. RCSx Control Bits

RCS3	RCS2	RCS1	RCS0	Set 1	Set 2
0	0	0	0	50	75
0	0	0	1	110	110
0	0	1	0	134.5	134.5
0	0	1	1	200	150
0	1	0	0	300	300
0	1	0	1	600	600
0	1	1	0	1200	1200
0	1	1	1	1050	2000
1	0	0	0	2400	2400
1	0	0	1	4800	4800
1	0	1	0	7200	1800
1	0	1	1	9600	9600
1	1	0	0	38.4k	19.2k
1	1	0	1	TIMER	TIMER
1	1	1	0	—	—
1	1	1	1	—	—

NOTE: These values are only valid for a 3.6864 MHz MC68307 clock.

TCS3–TCS0—Transmitter Clock Select

These bits select the baud rate clock for the transmitter from a set of baud rates listed in Table 8-7. The baud rate set selected depends upon UACR bit 7. Set 1 is selected if UACR bit 7 = 0, and set 2 is selected if UACR bit 7 = 1. The transmitter clock is always 16 times the baud rate shown in this list.

Table 8-7. TCSx Control Bits

TCS3	TCS2	TCS1	TCS0	Set 1	Set 2
0	0	0	0	50	75
0	0	0	1	110	110
0	0	1	0	134.5	134.5
0	0	1	1	200	150
0	1	0	0	300	300
0	1	0	1	600	600
0	1	1	0	1200	1200
0	1	1	1	1050	2000
1	0	0	0	2400	2400
1	0	0	1	4800	4800
1	0	1	0	7200	1800
1	0	1	1	9600	9600
1	1	0	0	38.4k	19.2k
1	1	0	1	TIMER	TIMER
1	1	1	0	—	—
1	1	1	1	—	—

NOTE: These values are only valid for a 3.6864 MHz MC68307 clock.

8.4.1.5 COMMAND REGISTER (UCR). The UCR is used to supply commands to the UART. Multiple commands can be specified in a single write to the UCR if the commands are not conflicting—e.g., reset transmitter and enable transmitter commands cannot be specified in a single command.

UCR MBASE + $105

7	6	5	4	3	2	1	0
—	MISC2	MISC1	MISC0	TC1	TC0	RC1	RC0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor or User

MISC3–MISC0—Miscellaneous Commands

These bits select a single command as listed in Table 8-8.

Table 8-8. MISCx Control Bits

MISC2	MISC1	MISC0	Command
0	0	0	No Command
0	0	1	Reset Mode Register Pointer
0	1	0	Reset Receiver
0	1	1	Reset Transmitter
1	0	0	Reset Error Status
1	0	1	Reset Break-Change Interrupt
1	1	0	Start Break
1	1	1	Stop Break

Reset Mode Register Pointer—The reset mode register pointer command causes the mode register pointer to point to UMR1.

Reset Receiver—The reset receiver command resets the receiver. The receiver is immediately disabled, the FFULL and RxRDY bits in the USR are cleared, and the receiver FIFO pointer is reinitialized. All other registers are unaltered. This command should be used in lieu of the receiver disable command whenever the receiver configuration is changed because it places the receiver in a known state.

Reset Transmitter—The reset transmitter command resets the transmitter. The transmitter is immediately disabled, and the TxEMP and TxRDY bits in the USR are cleared. All other registers are unaltered. This command should be used in lieu of the transmitter disable command whenever the transmitter configuration is changed because it places the transmitter in a known state.

Reset Error Status—The reset error status command clears the RB, FE, PE, and OE bits (in the USR). This command is also used in the block mode to clear all error bits after a data block is received.

Reset Break-Change Interrupt—The reset break-change interrupt command clears the delta break (DBx) bits in the UISR.

Start Break—The start break command forces TxD low. If the transmitter is empty, the start of the break conditions can be delayed up to one bit time. If the transmitter is active,

the break begins when transmission of the character is complete. If a character is in the transmitter shift register, the start of the break is delayed until the character is transmitted. If the transmitter holding register has a character, that character is transmitted after the break. The transmitter must be enabled for this command to be accepted. The state of the $\overline{CTS}$ input is ignored for this command.

Stop Break—The stop break command causes TxD to go high (mark) within two bit times. Characters stored in the transmitter buffer, if any, are transmitted.

TC1–TC0—Transmitter Commands

These bits select a single command as listed in Table 8-9.

Table 8-9. TCx Control Bits

TC1	TC0	Command
0	0	No Action Taken
0	1	Enable Transmitter
1	0	Disable Transmitter
1	1	Do Not Use

No Action Taken—The no action taken command causes the transmitter to stay in its current mode. If the transmitter is enabled, it remains enabled; if disabled, it remains disabled.

Transmitter Enable—The transmitter enable command enables operation of the channel's transmitter. The TxEMP and TxRDY bits in the USR are also set. If the transmitter is already enabled, this command has no effect.

Transmitter Disable—The transmitter disable command terminates transmitter operation and clears the TxEMP and TxRDY bits in the USR. However, if a character is being transmitted when the transmitter is disabled, the transmission of the character is completed before the transmitter becomes inactive. If the transmitter is already disabled, this command has no effect.

Do Not Use—Do not use this bit combination because the result is indeterminate.

RC1–RC0—Receiver Commands

These bits select a single command as listed in Table 8-10.

Table 8-10. RCx Control Bits

RC1	RC0	Command
0	0	No Action Taken
0	1	Enable Receiver
1	0	Disable Receiver
1	1	Do Not Use

No Action Taken—The no action taken command causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.

Receiver Enable—The receiver enable command enables operation of the channel's receiver. If the serial module is not in multidrop mode, this command also forces the receiver

into the search-for-start-bit state. If the receiver is already enabled, this command has no effect.

Receiver Disable—The receiver disable command disables the receiver immediately. Any character being received is lost. The command has no effect on the receiver status bits or any other control register. If the serial module is programmed to operate in the local loop-back mode or multidrop mode, the receiver operates even though this command is select-ed. If the receiver is already disabled, this command has no effect.

Do Not Use—Do not use this bit combination because the result is indeterminate.

8.4.1.6 RECEIVER BUFFER (URB). The receiver buffer contains three receiver holding registers and a serial shift register. The RxD pin is connected to the serial shift register. The holding registers act as a FIFO. The CPU reads from the top of the stack while the receiver shifts and updates from the bottom of the stack when the shift register has been filled (see Figure 8-4).

URB MBASE + $107

7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor or User

RB7–RB0—These bits contain the character in the receiver buffer.

8.4.1.7 TRANSMITTER BUFFER (UTB). The transmitter buffer consists of two registers, the transmitter holding register and the transmitter shift register (see Figure 8-3). The holding register accepts characters from the bus master if the TxRDY bit in the channel's USR is set. A write to the transmitter buffer clears the TxRDY bit, inhibiting any more characters until the shift register is ready to accept more data. When the shift register is empty, it checks to see if the holding register has a valid character to be sent (TxRDY bit cleared). If there is a valid character, the shift register loads the character and reasserts the TxRDY bit in the USR. Writes to the transmitter buffer when the channel's USR TxRDY bit is clear and when the transmitter is disabled have no effect on the transmitter buffer.

UTB MBASE + $107

7	6	5	4	3	2	1	0
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor or User

TB7–TB0—These bits contain the character in the transmitter buffer.

8.4.1.8 INPUT PORT CHANGE REGISTER (UIPCR). The UIPCR shows the current state and the change-of-state for the $\overline{\text{CTS}}$ pin.

UIPCR MBASE + $109

7	6	5	4	3	2	1	0
0	0	0	COS	1	1	1	$\overline{\text{CTS}}$
RESET:							
0	0	0	0	0	1	1	$\overline{\text{CTS}}$

Read Only Supervisor or User

Bits 7, 6, 5, 3, 2, 1—Reserved by Motorola.

COS—Change-of-State

1 = A change-of-state (high-to-low or low-to-high transition), lasting longer than 25–50 µs has occurred at the corresponding IPx input. When these bits are set, the UACR can be programmed to generate an interrupt to the CPU.
0 = No change-of-state has occurred since the last time the CPU read the UIPCR. A read of the UIPCR also clears the UISR COS bit.

$\overline{\text{CTS}}$—Current State

Starting two serial clock periods after reset, the $\overline{\text{CTS}}$ bit reflects the state of the $\overline{\text{CTS}}$ pin. If the $\overline{\text{CTS}}$ pin is detected as asserted at that time, the COS bit is set, which initiates an interrupt if the IEC bit of the UACR register is enabled.

1 = The current state of the $\overline{\text{CTS}}$ input is logic one.
0 = The current state of the $\overline{\text{CTS}}$ input is logic zero.

8.4.1.9 AUXILIARY CONTROL REGISTER (UACR). The UACR selects which baud rate is used and controls the handshake of the transmitter/receiver.

UACR MBASE + $109

7	6	5	4	3	2	1	0
BRG	CTMS2	CTMS1	CTMS0	—	—	—	IEC
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor or User

BRG—Baud Rate Generator Set Select

1 = Set 2 of the available baud rates is selected.
0 = Set 1 of the available baud rates is selected. Refer to **Section 8.4.1.4 Clock-select Register (UCSR)** for more information on the baud rates.

CTMS2–0—Timer Mode and Source Select

Table 8-11 shows the timer mode and source select bit fields.

Table 8-11. Timer Mode and Source Select Bits

CTMS2	CTMS1	CTMS0	Mode Command	Clock Source Select Command
0	1	1	Counter	Crystal or External Clock

NOTE: Other values are invalid on the MC68307 and should not be used.

IEC—Input Enable Control

1 = UISR bit 7 is set and an interrupt is generated when the COS bit in the UIPCR is set by an external transition on the $\overline{CTS}$ input (if bit 7 of the interrupt mask register (UIMR) is set to enable interrupts).
0 = Setting the corresponding bit in the UIPCR has no effect on UISR bit 7.

8.4.1.10 INTERRUPT STATUS REGISTER (UISR). The UISR provides status for all potential interrupt sources. The contents of this register are masked by the UIMR. If a flag in the UISR is set and the corresponding bit in UIMR is also set, the internal interrupt output is asserted. If the corresponding bit in the UIMR is cleared, the state of the bit in the UISR has no effect on the output.

NOTE

The UIMR does not mask reading of the UISR. True status is provided regardless of the contents of UIMR. The contents of UISR are cleared when the serial module is reset.

UISR MBASE + $10B

7	6	5	4	3	2	1	0
COS	—	—	—	—	DB	RxRDY	TxRDY
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor or User

COS—Change-of-State

1 = A change-of-state has occurred at the $\overline{CTS}$ input and has been selected to cause an interrupt by programming bit 0 of the UACR.
0 = COS bit in the UIPCR is not selected.

DB—Delta Break

1 = The receiver has detected the beginning or end of a received break.
0 = No new break-change condition to report. Refer to **Section 8.4.1.5 Command Register (UCR)** for more information on the reset break-change interrupt command.

RxRDY—Receiver Ready or FIFO Full

The function of this bit is programmed by UMR1 bit 6. It is a duplicate of either the FFULL or RxRDY bit of USR.

TxRDY—Transmitter Ready

This bit is the duplication of the TxRDY bit in USR.

1 = The transmitter holding register is empty and ready to be loaded with a character.
0 = The transmitter holding register was loaded by the CPU, or the transmitter is disabled. Characters loaded into the transmitter holding register when TxRDY=0 are not transmitted.

8.4.1.11 INTERRUPT MASK REGISTER (UIMR). The UIMR selects the corresponding bits in the UISR that cause an interrupt. If one of the bits in the UISR is set and the corresponding bit in the UIMR is also set, the internal interrupt output is asserted. If the corresponding bit in the UIMR is zero, the state of the bit in the UISR has no effect on the interrupt output. The UIMR does not mask the reading of the UISR.

UIMR MBASE + $10B

7	6	5	4	3	2	1	0
COS	—	—	—	—	DB	FFULL	TxRDY
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor or User

COS—Change-of-State

1 = Enable interrupt
0 = Disable interrupt

DB—Delta Break

1 = Enable interrupt
0 = Disable interrupt

FFULL—FIFO Full

1 = Enable interrupt
0 = Disable interrupt

TxRDY—Transmitter Ready

1 = Enable interrupt
0 = Disable interrupt

8.4.1.12 TIMER UPPER PRELOAD REGISTER (UBG1). This register holds the eight most significant bits of the preload value to be used by the timer in order to provide a given baud rate. The minimum value that can be loaded on the concatenation of UBG1 with UBG2 is $0002. This register is write only and cannot be read by the CPU.

8.4.1.13 TIMER UPPER PRELOAD REGISTER (UBG2). This register holds the eight least significant bits of the preload value to be used by the timer in order to provide a given baud rate. The minimum value that can be loaded on the concatenation of UBG1 with UBG2 is $0002. This register is write only and cannot be read by the CPU.

8.4.1.14 INTERRUPT VECTOR REGISTER (UIVR). The UIVR contains the 8-bit vector number of the internal interrupt. See **Section 5.1.4.2 Interrupt Vector Generation**,

UIVR MBASE + $119

7	6	5	4	3	2	1	0
IVR7	IVR6	IVR5	IVR4	IVR3	IVR2	IVR1	IVR0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor or User

IVR7–IVR0—Interrupt Vector Bits

This 8-bit number indicates the offset from the base of the vector table where the address of the exception handler for the specified interrupt is located. The UIVR is reset to $0F, which indicates an uninitialized interrupt condition. See **Section 5.1.4.2 Interrupt Vector Generation** for more information.

8.4.1.15 INPUT PORT REGISTER (UIP). The UIP register shows the current state of the $\overline{CTS}$ input.

UIP MBASE + $11B

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	$\overline{CTS}$
RESET:							
1	1	1	1	1	1	1	1

Read Only Supervisor or User

$\overline{CTS}$—Current State

1 = The current state of the $\overline{CTS}$ input is logic one.
0 = The current state of the $\overline{CTS}$ input is logic zero.

The information contained in this bit is latched and reflects the state of the input pin at the time that the UIP is read.

NOTE

This bit has the same function and value as the UIPCR bit 0.

8.4.1.16 OUTPUT PORT DATA REGISTERS (UOP1, UOP0). The $\overline{\text{RTS}}$ output is set by performing a bit set command (writing to UOP1) and is cleared by performing a bit reset command (writing to UOP0).

Bit Set

UOP1 MBASE + $11D

7	6	5	4	3	2	1	0
							$\overline{\text{RTS}}$
RESET:							
—	—	—	—	—	—	—	0

Write Only Supervisor or User

$\overline{\text{RTS}}$—Output Port Parallel Output

1 = A write cycle to the OP bit set command address sets all OP bits corresponding to one bits on the data bus.
0 = These bits are not affected by writing a zero to this address.

NOTE

The output port bits are inverted at the pins, so the $\overline{\text{RTS}}$ set bit provides an asserted $\overline{\text{RTS}}$ pin.

Bit Reset

UOP0 MBASE + $11F

7	6	5	4	3	2	1	0
							$\overline{\text{RTS}}$
RESET:							
—	—	—	—	—	—	—	—

Write Only Supervisor or User

$\overline{\text{RTS}}$—Output Port Parallel Output

1 = A write cycle to the OP bit reset command address clears all OP bits corresponding to one bits on the data bus.
0 = These bits are not affected by writing a zero to this address.

8.4.2 Programming

The basic interface software flowchart required for operation of the serial module is shown in Figure 8-9. The routines are divided into three categories:

- Serial Module Initialization
- I/O Driver
- Interrupt Handling

8.4.2.1 SERIAL MODULE INITIALIZATION. The serial module initialization routines consist of SINIT and CHCHK. SINIT is called at system initialization time to check UART operation. Before SINIT is called, the calling routine allocates two words on the system stack. Upon return to the calling routine, SINIT passes information on the system stack to reflect the status of the UART. If SINIT finds no errors, the receiver and transmitter are enabled. The CHCHK routine performs the actual checks as called from the SINIT routine. When called, SINIT places the UART in the local loopback mode and checks for the following errors:

- Transmitter Never Ready
- Receiver Never Ready
- Parity Error
- Incorrect Character Received

8.4.2.2 I/O DRIVER EXAMPLE. The I/O driver routines consist of INCH and OUTCH. INCH is the terminal input character routine and gets a character from the receiver. OUTCH is used to send a character to the transmitter.

8.4.2.3 INTERRUPT HANDLING. The interrupt handling routine consists of SIRQ, which is executed after the serial module generates an interrupt caused by a change-in-break (beginning of a break). SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

8.5 SERIAL MODULE INITIALIZATION SEQUENCE

If the serial capability of the MC68307 is being used, the following steps are required to properly initialize the serial module.

NOTE

The serial module registers can be accessed by word or byte operations, but only the data byte D7–D0 is valid.

Command Register (UCR)

- Reset the Receiver and Transmitter.

Interrupt Vector Register (UIVR)

- Program the Vector Number for a Serial Module Interrupt.

Interrupt Mask Register (UIMR)

- Enable the Desired Interrupt Sources.

Auxiliary Control Register (UACR)

- Select Baud Rate Set (BRG bit).
- Initialize the Input Enable Control (IEC bit).
- Select Timer Mode and Clock Source if Necessary.

Clock Select Register (UCSR)

- Select the Receiver and Transmitter Clock. Use Timer as Source if Required.

Mode Register 1 (UMR1)

- If Desired, Program Operation of Receiver Ready-to-Send (RxRTS Bit).
- Select Receiver-Ready or FIFO-Full Notification (R/F Bit).
- Select Character or Block Error Mode (ERR Bit).
- Select Parity Mode and Type (PM and PT Bits).
- Select Number of Bits Per Character (B/Cx Bits).

Mode Register 2 (UMR2)

- Select the Mode of Operation (CMx bits).
- If Desired, Program Operation of Transmitter Ready-to-Send (TxRTS Bit).
- If Desired, Program Operation of Clear-to-Send (TxCTS Bit).
- Select Stop-Bit Length (SBx Bits).

Command Register (UCR)

- Enable the Receiver and Transmitter.

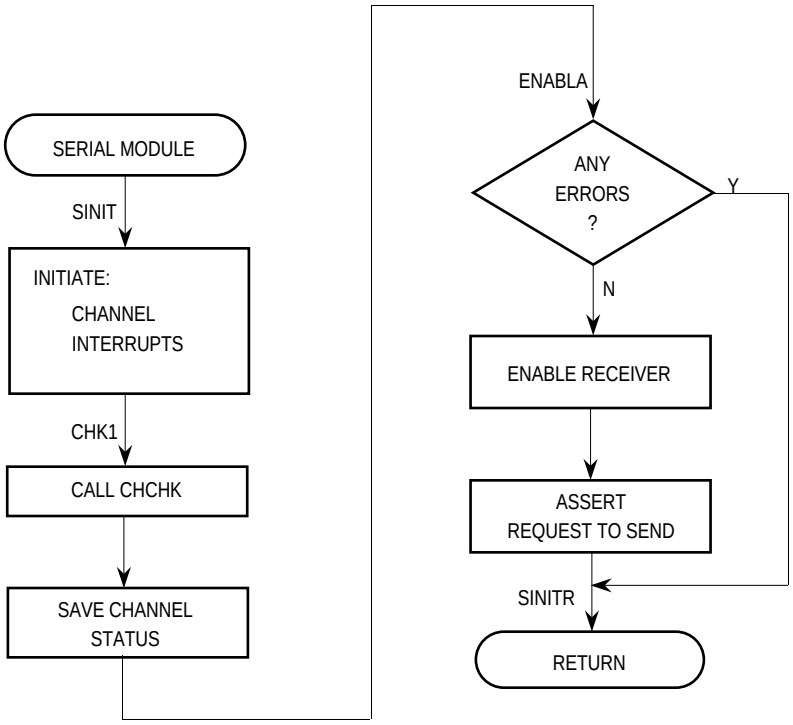


Figure 8-9. Serial Mode Programming Flowchart (1 of 5)

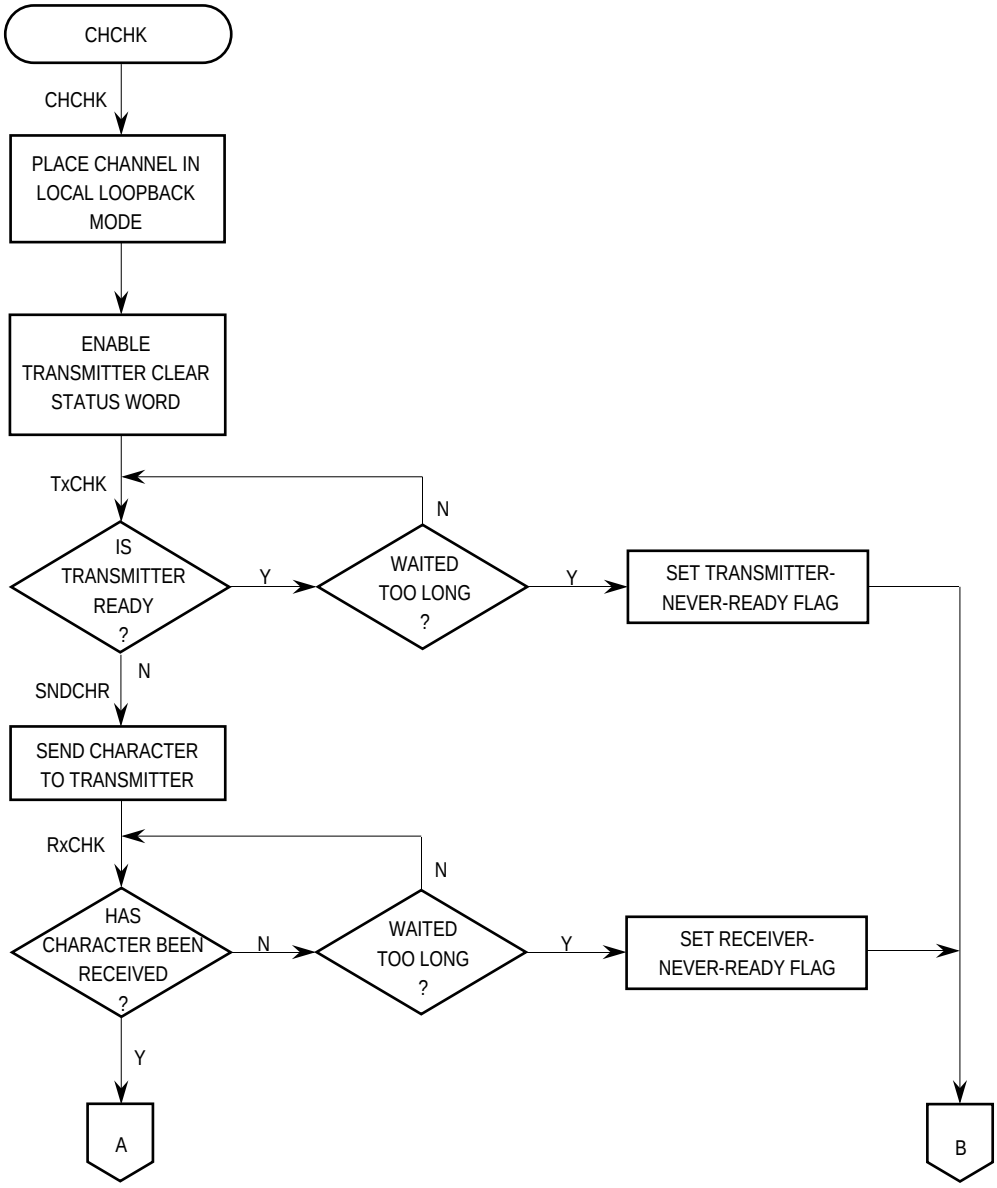


Figure 8-9. Serial Mode Programming Flowchart (2 of 5)

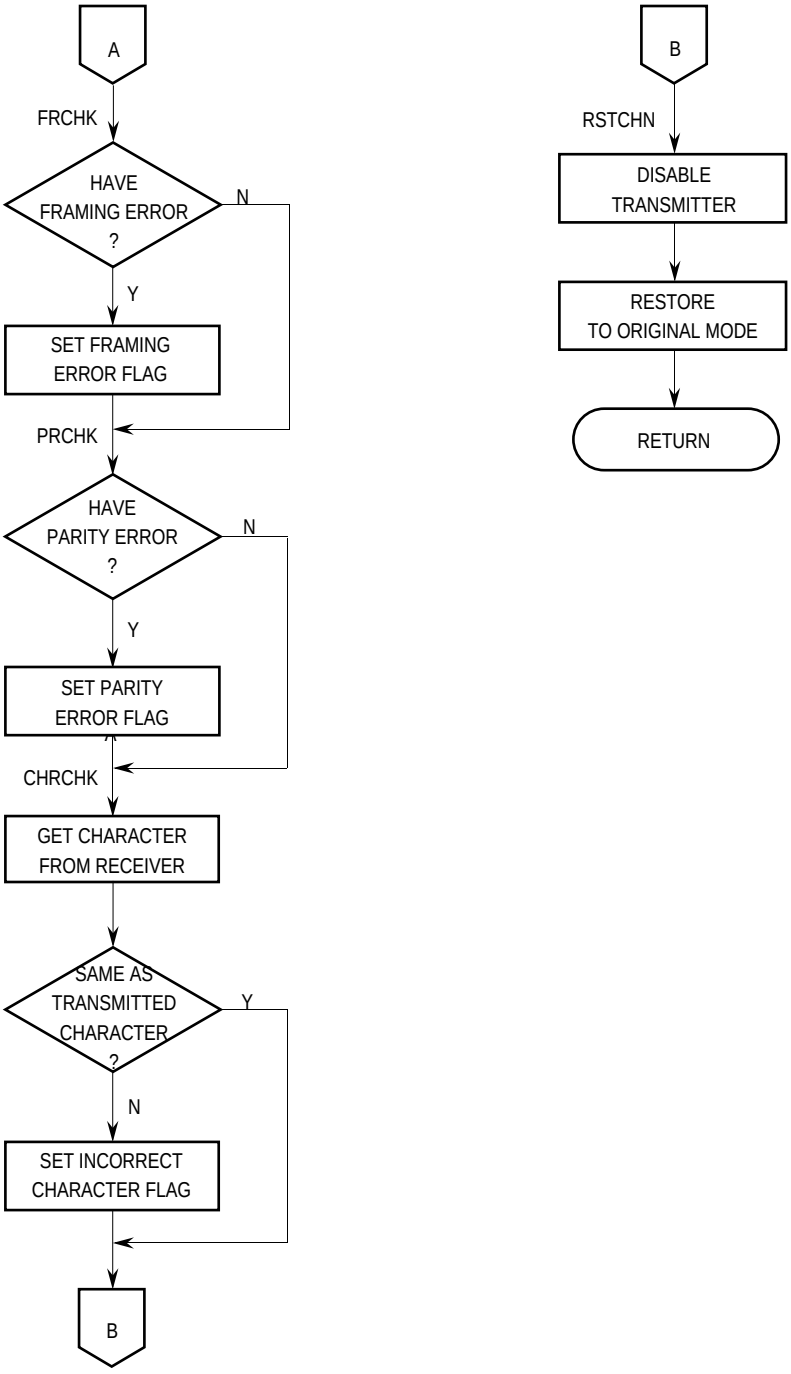


Figure 8-9. Serial Mode Programming Flowchart (3 of 5)

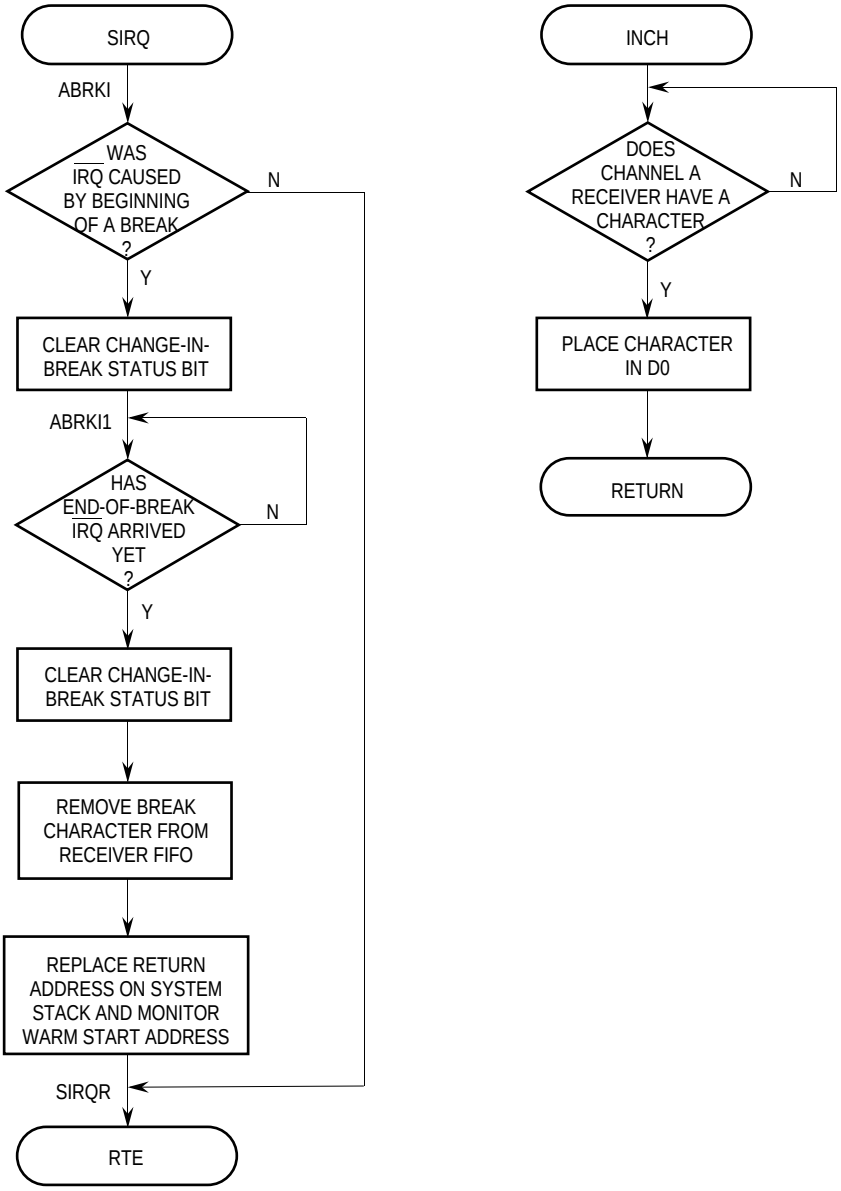


Figure 8-9. Serial Mode Programming Flowchart (4 of 5)

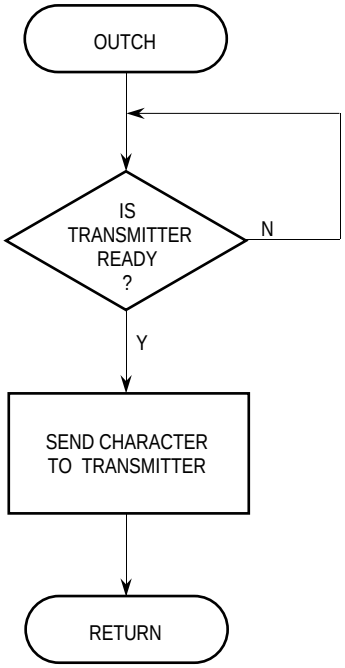


Figure 8-9. Serial Mode Programming Flowchart (5 of 5)

SECTION 9
IEEE 1149.1 TEST ACCESS PORT

The MC68307 includes dedicated user-accessible test logic that is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to development of this standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The MC68307 implementation supports circuit-board test strategies based on this standard.

The test logic includes a test access port (TAP) consisting of five dedicated signal pins, a 16-state controller, an instruction register, and four test data registers. A boundary scan register links all device signal pins into a single shift register. The test logic, implemented using static logic design, is independent of the device system logic. The MC68307 implementation provides the following capabilities:

1. Perform boundary scan operations to test circuit-board electrical continuity
2. Sample the MC68307 system pins during operation and transparently shift out the result in the boundary scan register
3. Bypass the MC68307 for a given circuit-board test by effectively reducing the boundary scan register to a single bit
4. Disable the output drive to pins during circuit-board testing
5. Drive output pins to stable levels

NOTE

Certain precautions must be observed to ensure that the IEEE 1149.1 test logic does not interfere with non-test operation. See **Section 9.6 Non-IEEE 1149.1 Operation** for details.

9.1 OVERVIEW

NOTE

This description is not intended to be used without the supporting IEEE 1149.1 document.

The discussion includes those items required by the standard and provides additional information specific to the MC68307 implementation. For internal details and applications of the standard, refer to the IEEE 1149.1 document.

An overview of the MC68307 implementation of IEEE 1149.1 is shown in Figure 9-1. The MC68307 implementation includes a 16-state controller, a 4-bit instruction register, and four

test registers (a 1-bit bypass register, a 117-bit boundary scan register, a 6-bit module mode register, and a 32-bit ID register). This implementation includes a dedicated TAP consisting of the following signals:

- TCK —test clock input to synchronize the test logic (with pulldown).
- TMS —test mode select input (with an internal pullup resistor) that is sampled on the rising edge of TCK to sequence the TAP controller's state machine.
- TDI —test data input (with an internal pullup resistor) that is sampled on the rising edge of TCK.
- TDO —three-state test data output that is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.

No TRST pin is provided since the TAP pin is reset by an internal power-on reset circuit.

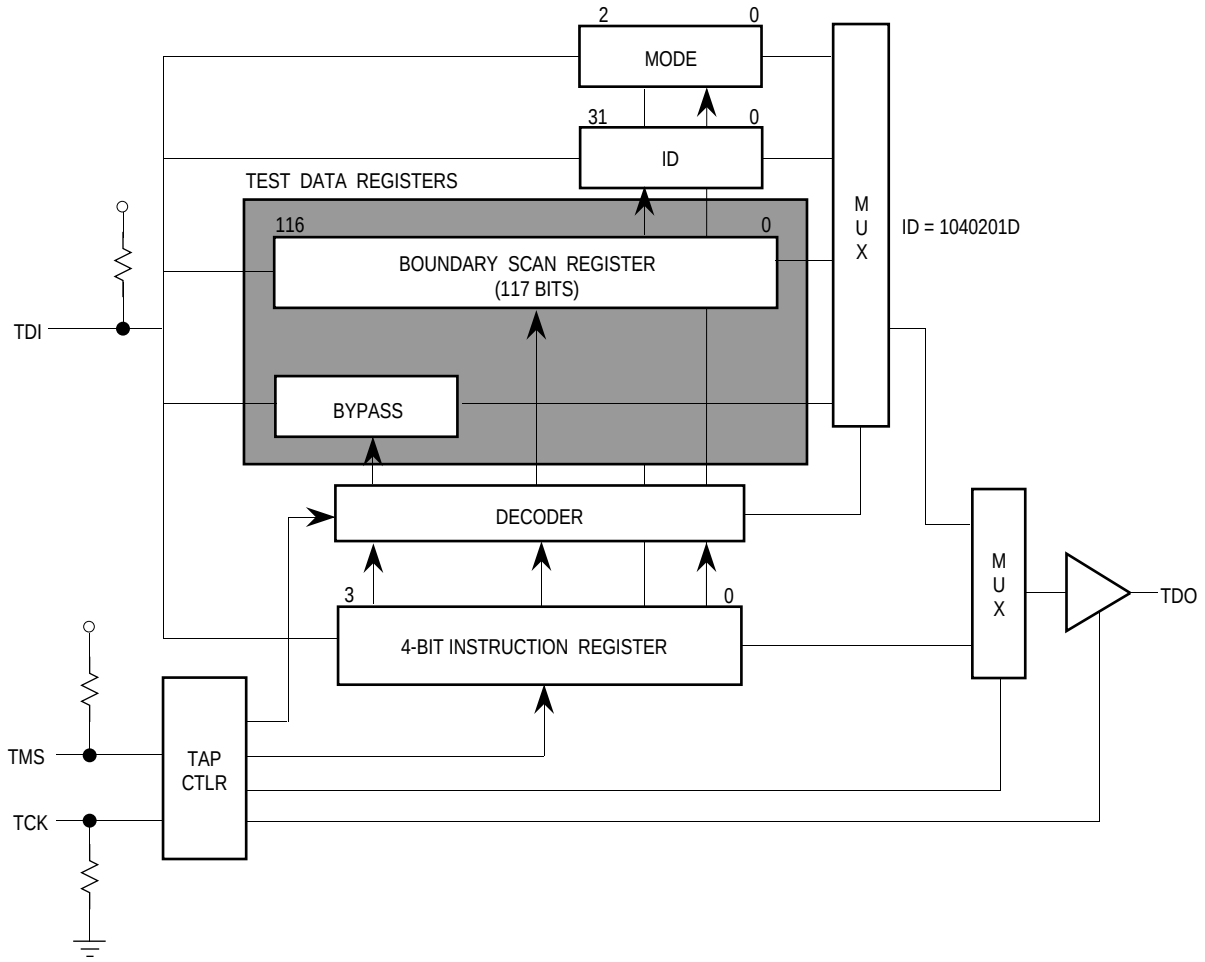


Figure 9-1. Test Access Port Block Diagram

9.2 TAP CONTROLLER

The TAP controller is responsible for interpreting the sequence of logical values on the TMS signal. It is a synchronous state machine that controls the operation of the JTAG logic. The state machine is shown in Figure 9-2; the value shown adjacent to each arc represents the value of the TMS signal sampled on the rising edge of the TCK signal. For a description of the TAP controller states, please refer to the IEEE 1149.1 document.

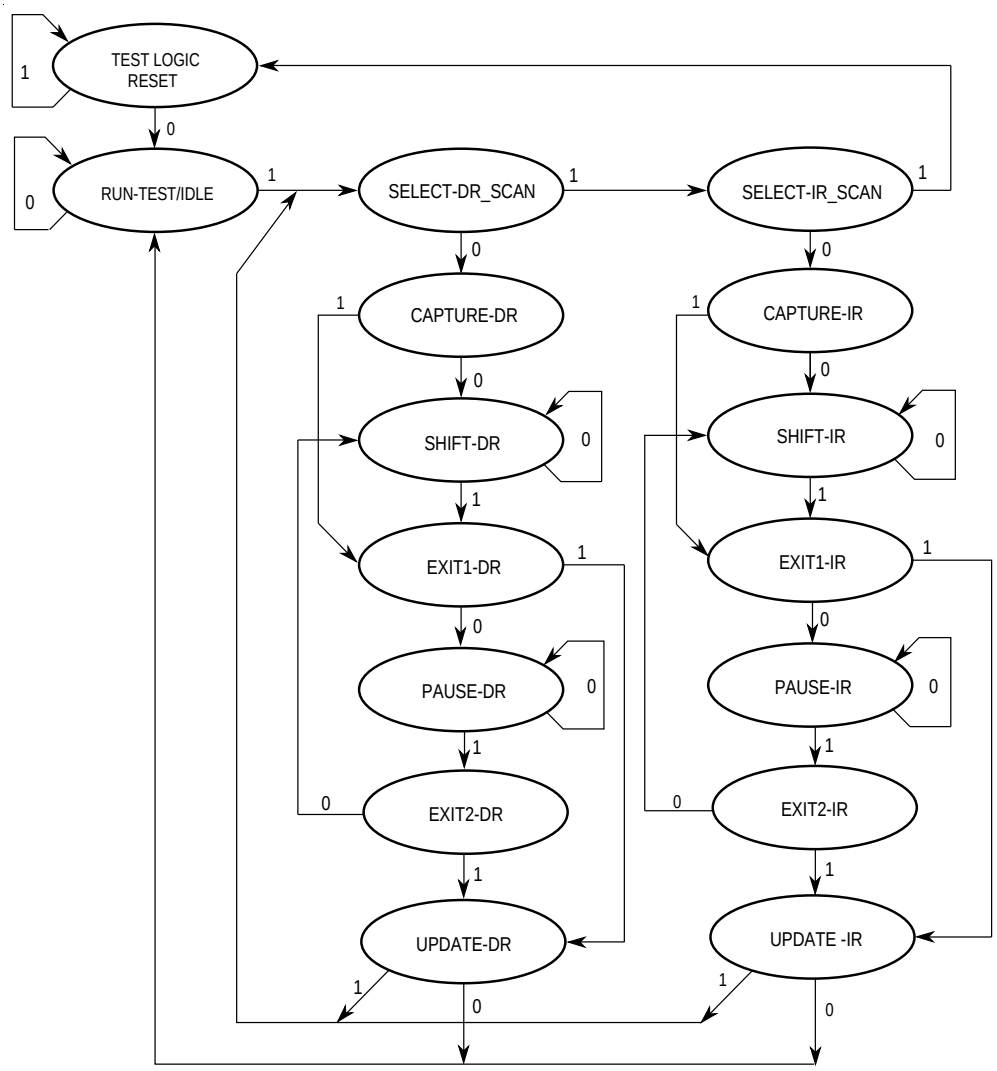


Figure 9-2. TAP Controller State Machine

9.3 BOUNDARY SCAN REGISTER

The MC68307 IEEE 1149.1 implementation has a 124-bit boundary scan register. This register contains bits for all device signal and clock pins and associated control signals. The XTAL, EXTAL and $\overline{\text{RSTIN}}$ pins are associated with analog signals and are not included in the boundary scan register.

All MC68307 bidirectional pins, except the open-drain I/O pins ($\overline{\text{HALT}}$, $\overline{\text{DTACK}}$, $\overline{\text{RESET}}$, SCL and SDA), have a register bit for the output path and another for the input path. In addition, SCL and SDA have a third bit which controls these pins. All open drain I/O pins have a single register bit for pin data and no associated control bit. To ensure proper operation, the open-drain pins require external pullups. Thirty-two control bits in the boundary scan register define the output enable signal for associated groups of bidirectional and three-state pins. The control bits and their bit positions are listed in Table 9-1.

Table 9-1. Boundary Scan Control Bits

Name	Bit Number	Name	Bit Number	Name	Bit Number	Name	Bit Number
bus.ctl	3	pa3.ctl	57	pb11.ctl	73	pb3.ctl	89
rw.ctl	10	pa2.ctl	59	pb10.ctl	75	pb2.ctl	91
adb.ctl	21	pa1.ctl	61	pb9.ctl	77	pb1.pu	93
ab.ctl	46	pa0.ctl	63	pb8.ctl	79	pb1.ctl	94
pa7.ctl	49	pb15.ctl	65	pb7.ctl	81	pb0.pu	96
pa6.ctl	51	pb14.ctl	67	pb6.ctl	83	pb0.ctl	97
pa5.ctl	53	pb13.ctl	69	pb5.ctl	85	dhi.ctl	99
pa4.ctl	55	pb12.ctl	71	pb4.ctl	87	dlo.ctl	108

Boundary scan bit definitions are shown in Table 9-2. The first column in Table 9-2 defines the bit's ordinal position in the boundary scan register. The shift register bit nearest TDO (i.e., first to be shifted out) is defined as bit 0; the last bit to be shifted out is bit 116.

The second column references one of the five MC68307 cell types depicted in Figure 9-3 to Figure 9-7, which describe the cell structure for each type.

The third column lists the pin name for all pin-related bits or defines the name of bidirectional control register bits.

The last column indicates the associated boundary scan register control bit.

Bidirectional pins include a single scan bit for data (IO.Cell) as depicted in Figure 9-7. These bits are controlled by one of the two bits shown in Figure 9-5 and Figure 9-6. The value of the control bit determines whether the bidirectional pin is an input or an output. One or more bidirectional data bits can be serially connected to a control bit as shown in Figure 9-8. Note that, when sampling the bidirectional data bits, the bit data can be interpreted only after examining the IO control bit to determine pin direction.

Table 9-2. Boundary Scan Bit Definitions

Bit Num	Cell Type	Signal	Control
0	O.Cell	ALE	pb0.pu
1	O.Cell	$\overline{RD}$	pb0.ctl
2	O.Cell	$\overline{WR}$	SCL
3	En.Cell	bus.ctl	dhi.ctl
4	IO.Cell	$\overline{AS}$	D15
5	IO.Cell	$\overline{UDS}$	D14
6	IO.Cell	$\overline{LDS}$	D13
7	IO.Cell	$\overline{RW}$	D12
8	O.Cell	$\overline{DTACK}$	D11
9	I.Cell	$\overline{DTACK}$	D10
10	En.Cell	rw.ctl	D9
11	O.Cell	$\overline{HALT}$	D8
12	I.Cell	$\overline{HALT}$	dlo.ctl
13	O.Cell	$\overline{RESET}$	D7
14	I.Cell	$\overline{RESET}$	D6
15	O.Cell	$\overline{CS0}$	D5
16	O.Cell	$\overline{CS1}$	D4
17	O.Cell	$\overline{CS2}$	D3
18	O.Cell	$\overline{CS3}$	D2
19	O.Cell	CLKOUT	D1
20	I.Cell	BUSW	D0
21	En.Cell	adb.ctl	pb0.pu
22	IO.Cell	AD0	pb0.ctl
23	IO.Cell	AD1	SCL
24	IO.Cell	AD2	dhi.ctl
25	IO.Cell	AD3	D15
26	IO.Cell	AD4	D14
27	IO.Cell	AD5	D13
28	IO.Cell	AD6	D12
29	IO.Cell	AD7	D11
30	IO.Cell	A8	D10
31	IO.Cell	A9	D9
32	IO.Cell	A10	D8
33	IO.Cell	A11	dlo.ctl
34	IO.Cell	A12	ab.ctl
35	IO.Cell	A12	ab.ctl
36	IO.Cell	A13	ab.ctl
37	IO.Cell	A14	ab.ctl
38	IO.Cell	A15	ab.ctl
39	IO.Cell	A16	ab.ctl
40	IO.Cell	A17	ab.ctl
41	IO.Cell	A18	ab.ctl
42	IO.Cell	A19	ab.ctl
43	IO.Cell	A20	ab.ctl
44	IO.Cell	A21	ab.ctl
90	IO.Cell	RXD	pb3.ctl

Bit Num	Cell Type	Signal	Control
45	IO.Cell	A22	ab.ctl
46	En.Cell	ab.ctl	—
47	IO.Cell	A23	ab.ctl
48	I.Cell	$\overline{IRQ7}$	—
49	En.Cell	pa7.ctl	—
50	IO.Cell	$\overline{BGACK}$	pa7.ctl
51	En.Cell	pa6.ctl	—
52	IO.Cell	$\overline{BG}$	pa6.ctl
53	En.Cell	pa5.ctl	—
54	IO.Cell	$\overline{BR}$	pa5.ctl
55	En.Cell	pa4.ctl	—
56	IO.Cell	TOUT2	pa4.ctl
57	En.Cell	pa3.ctl	—
58	IO.Cell	TOUT1	pa3.ctl
59	En.Cell	pa2.ctl	—
60	IO.Cell	$\overline{CS2D}$	pa2.ctl
61	En.Cell	pa1.ctl	—
62	IO.Cell	$\overline{CS2C}$	pa1.ctl
63	En.Cell	pa0.ctl	—
64	IO.Cell	$\overline{CS2B}$	pa0.ctl
65	En.Cell	pb15.ctl	—
66	IO.Cell	$\overline{INT8}$	pb15.ctl
67	En.Cell	pb14.ctl	—
68	IO.Cell	$\overline{INT7}$	pb14.ctl
69	En.Cell	pb13.ctl	—
70	IO.Cell	$\overline{INT6}$	pb13.ctl
71	En.Cell	pb12.ctl	—
72	IO.Cell	$\overline{INT5}$	pb12.ctl
73	En.Cell	pb11.ctl	—
74	IO.Cell	$\overline{INT4}$	pb11.ctl
75	En.Cell	pb10.ctl	—
76	IO.Cell	$\overline{INT3}$	pb10.ctl
77	En.Cell	pb9.ctl	—
78	IO.Cell	$\overline{INT2}$	pb9.ctl
79	En.Cell	pb8.ctl	—
80	IO.Cell	$\overline{INT1}$	pb8.ctl
81	En.Cell	pb7.ctl	—
82	IO.Cell	TIN2	pb7.ctl
83	En.Cell	pb6.ctl	—
84	IO.Cell	TIN1	pb6.ctl
85	En.Cell	pb5.ctl	—
86	IO.Cell	$\overline{CTS}$	pb5.ctl
87	En.Cell	pb4.ctl	—
88	IO.Cell	$\overline{RTS}$	pb4.ctl
89	En.Cell	pb3.ctl	—
104	IO.Cell	D11	dhi.ctl

Table 9-2. Boundary Scan Bit Definitions (Continued)

91	En.Cell	pb2.ctl	—
92	IO.Cell	TXD	pb2.ctl
93	En.Cell	pb1.pu	—
94	En.Cell	pb1.ctl	—
95	IO.Cell	SDA	pb1.ctl/ pb1.pu
96	En.Cell	pb0.pu	—
97	En.Cell	pb0.ctl	—
98	IO.Cell	SCL	pb0.ctl/ pb0.pu
99	En.Cell	dhi.ctl	—
100	IO.Cell	D15	dhi.ctl
101	IO.Cell	D14	dhi.ctl
102	IO.Cell	D13	dhi.ctl
103	IO.Cell	D12	dhi.ctl

105	IO.Cell	D10	dhi.ctl
106	IO.Cell	D9	dhi.ctl
107	IO.Cell	D8	dhi.ctl
108	En.Cell	dlo.ctl	—
109	IO.Cell	D7	dlo.ctl
110	IO.Cell	D6	dlo.ctl
111	IO.Cell	D5	dlo.ctl
112	IO.Cell	D4	dlo.ctl
113	IO.Cell	D3	dlo.ctl
114	IO.Cell	D2	dlo.ctl
115	IO.Cell	D1	dlo.ctl
116	IO.Cell	D0	dlo.ctl

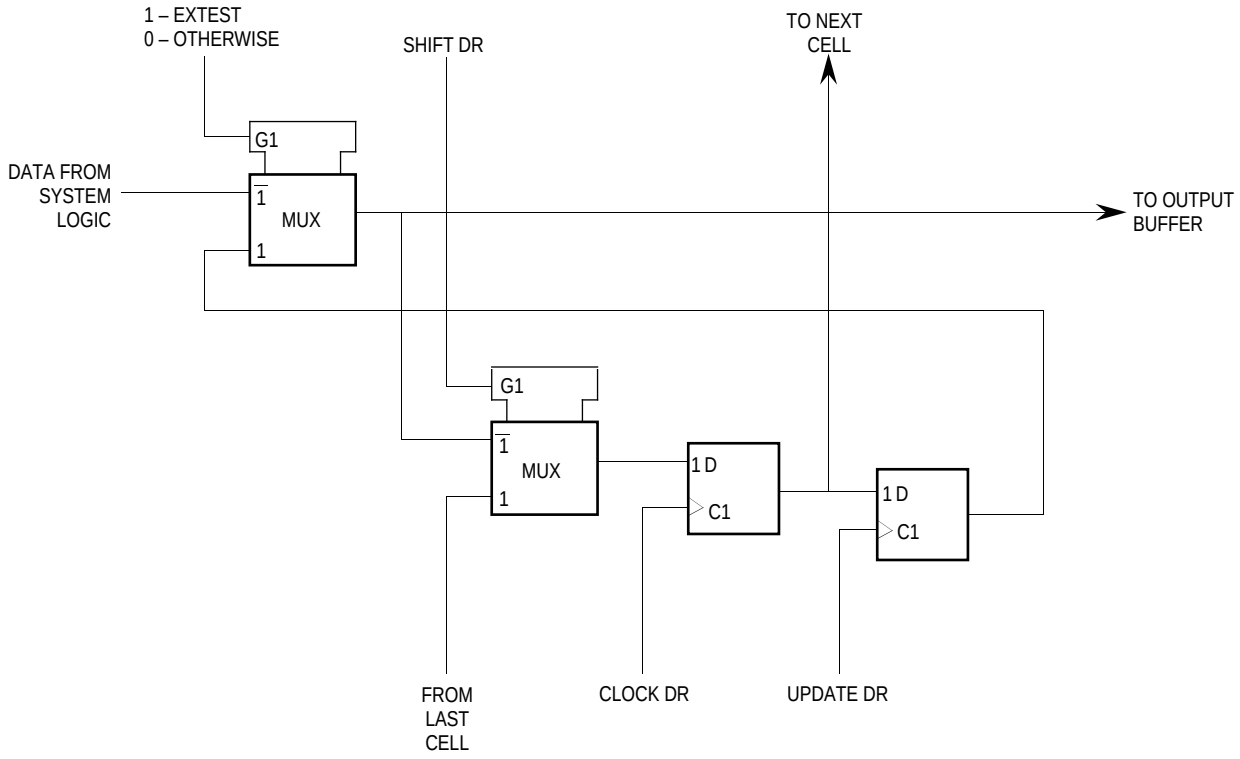


Figure 9-3. Output Cell (O.Cell)

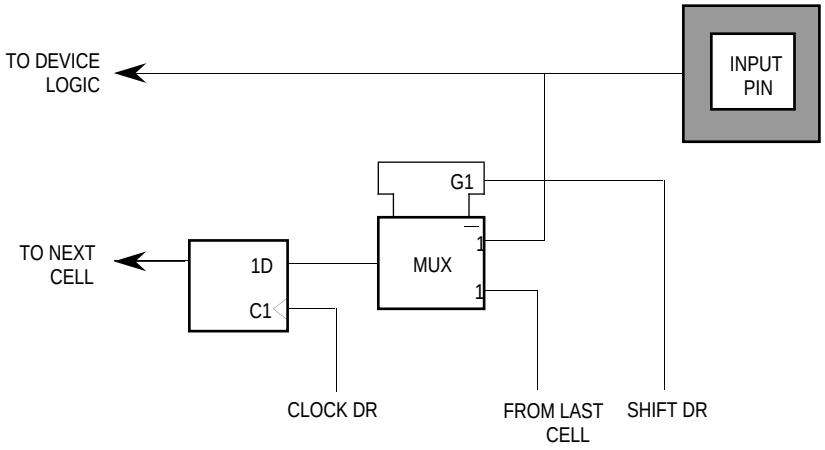


Figure 9-4. Input Cell (I.Cell)

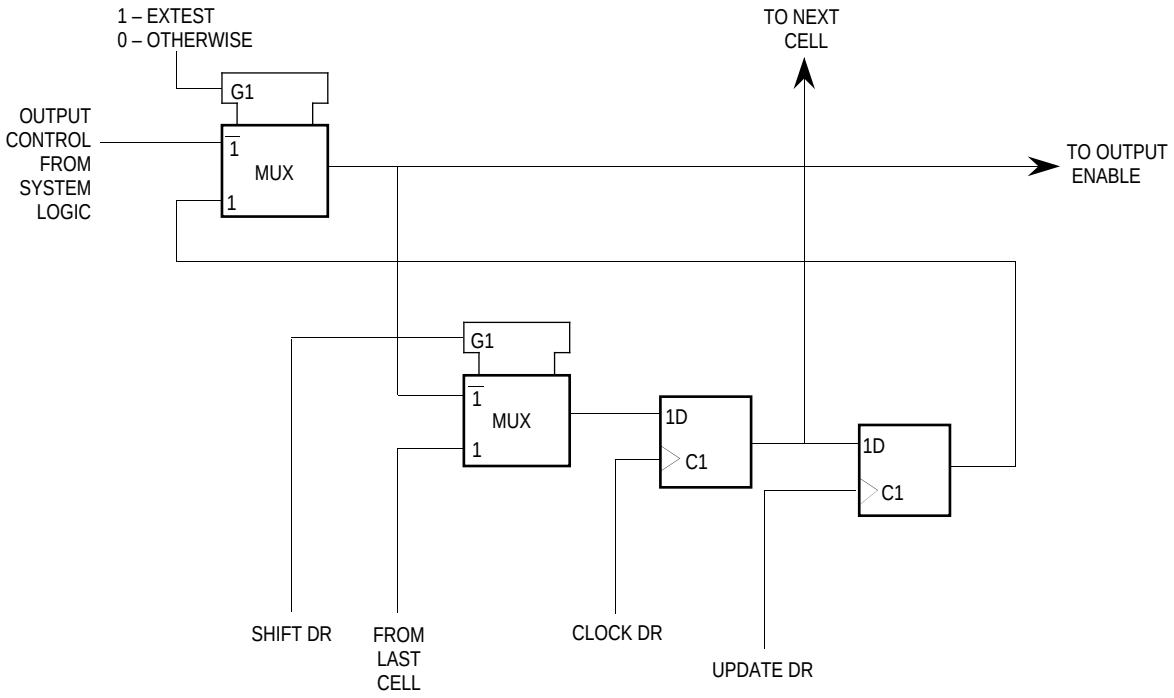


Figure 9-5. Output Control Cell (En.Cell)

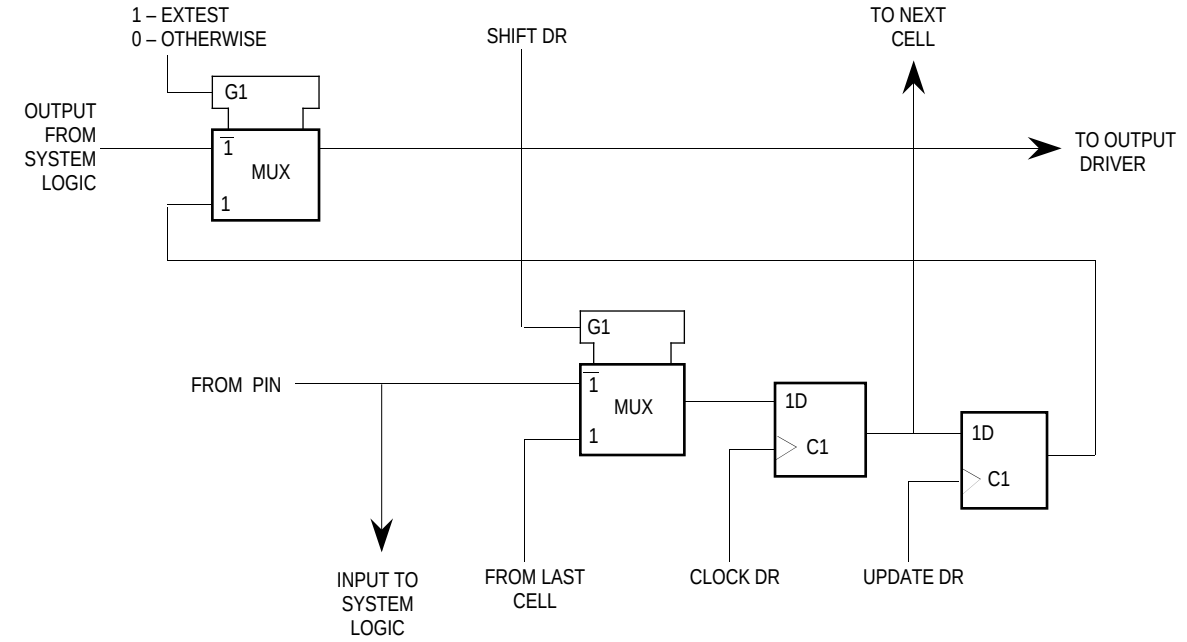


Figure 9-6. Bidirectional Cell (IO.Cell)

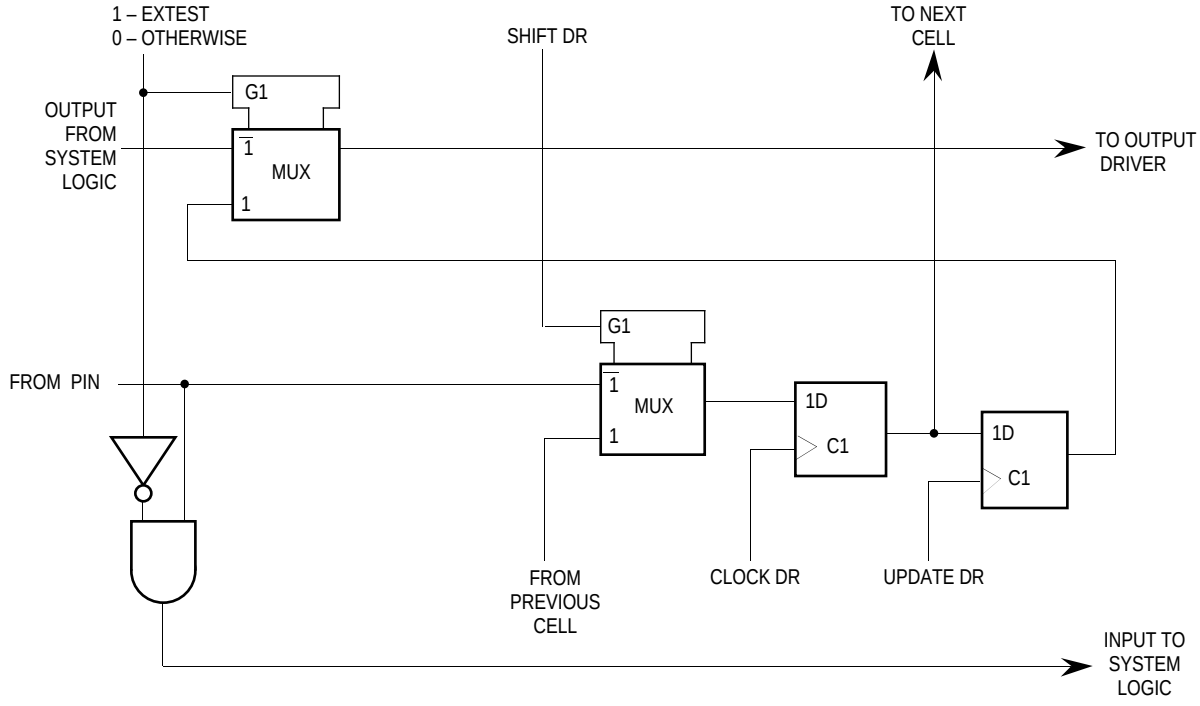
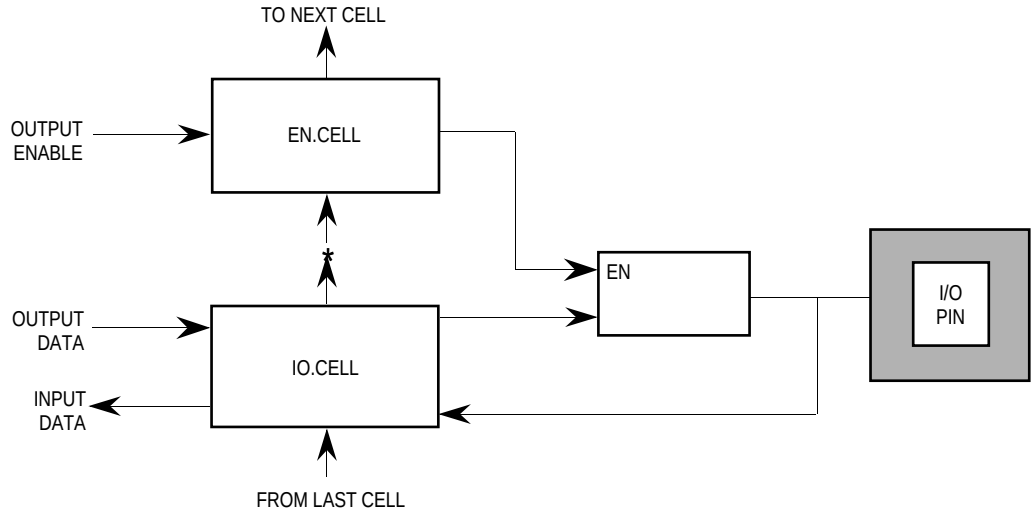


Figure 9-7. Bidirectional Cell (IOx0.Cell)



NOTE: More than one IO.Cell could be serially connected and controlled by a single En.Cell.

Figure 9-8. General Arrangement for Bidirectional Pins

9.4 INSTRUCTION REGISTER

The MC68307 IEEE 1149.1 implementation includes the three mandatory public instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS), the optional public ID instruction, plus two additional public instructions (CLAMP and HI-Z) defined by IEEE 1149.1. The MC68307 includes a 4-bit instruction register without parity, consisting of a shift register with four parallel outputs. Data is transferred from the shift register to the parallel outputs during the update-IR controller state. The four bits are used to decode the seven unique instructions listed in Table 9-3.

The parallel output of the instruction register is reset to 0001 in the test-logic-reset controller state. Note that this preset state is equivalent to the ID instruction.

Table 9-3. Instructions

Code				Instruction
B3	B2	B1	B0	
0	0	0	0	EXTEST
0	0	0	1	ID
0	0	1	0	SAMPLE
1	0	0	1	HI-Z
1	1	0	0	CLAMP
1	1	0	1	MODULE MODE
1	1	1	1	BYPASS

During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the 4-bit binary value (0001). The parallel outputs, however, remain unchanged by this action since an update-IR signal is required to modify them.

9.4.1 EXTEST (0000)

The external test (EXTEST) instruction selects the 117-bit boundary scan register. EXTEST asserts internal reset for the MC68307 system logic to force a predictable benign internal state while performing external boundary scan operations.

By using the TAP, the register is capable of a) scanning user-defined values into the output buffers, b) capturing values presented to input pins, c) controlling the direction of bidirectional pins, and d) controlling the output drive of three-state output pins. For more details on the function and uses of EXTEST, please refer to the IEEE 1149.1 document.

9.4.2 SAMPLE/PRELOAD (0010)

The SAMPLE/PRELOAD instruction selects the 117-bit boundary scan register and provides two separate functions. First, it provides a means to obtain a snapshot of system data and control signals. The snapshot occurs on the rising edge of TCK in the capture-DR controller state. The data can be observed by shifting it transparently through the boundary scan register.

NOTE

Since there is no internal synchronization between the IEEE 1149.1 clock (TCK) and the system clock (CLKOUT), the user must provide some form of external synchronization to achieve meaningful results.

The second function of SAMPLE/PRELOAD is to initialize the boundary scan register output bits prior to selection of EXTEST. This initialization ensures that known data appears on the outputs when entering the EXTEST instruction.

9.4.3 BYPASS (1111)

The BYPASS instruction selects the single-bit bypass register as shown in Figure 9-9. This creates a shift-register path from TDI to the bypass register and, finally, to TDO, circumventing the 117-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MC68307 becomes the device under test.

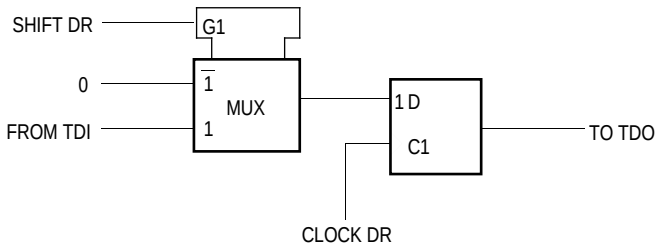


Figure 9-9. Bypass Register

When the bypass register is selected by the current instruction, the shift-register stage is set to a logic zero on the rising edge of TCK in the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero.

9.4.4 CLAMP (1100)

When the CLAMP instruction is invoked, the boundary scan multiplexer control signal EXTEST is asserted, and the BYPASS register is selected. CLAMP should be invoked after valid data has been shifted into the boundary scan register, e.g., by SAMPLE/PRELOAD. CLAMP allows static levels to be presented at the MC68307 output and bidirectional pins, like EXTEST, but without the shift latency of the boundary scan register from TDI to TDO.

9.5 MC68307 RESTRICTIONS

The control afforded by the output enable signals using the boundary scan register and the EXTEST instruction requires a compatible circuit-board test environment to avoid device-destructive configurations. The user must avoid situations in which the MC68307 output drivers are enabled into actively driven networks. Overdriving the TDO driver when it is active is not recommended.

9.6 NON-IEEE 1149.1 OPERATION

In non-IEEE 1149.1 operation, the IEEE 1149.1 test logic must be kept transparent to the system logic by forcing the TAP controller into the test-logic-reset state. This requires the TMS, TCK and TDI inouts to be high.

SECTION 10
APPLICATIONS INFORMATION

10.1 MC68307 MINIMUM STAND-ALONE SYSTEM HARDWARE

This section details a simple stand-alone system using the MC68307. It demonstrates the simplicity of system hardware design and inherent low system chip count when using the highly integrated MC68307. The system hardware is shown in Figure 10-1.

The 5-V design consists of a 16.67MHz MC68307, 8-bit boot EPROM, 16-bit SRAM, an RS232 driver and a few logic gates. It is considered stand-alone in that a dumb terminal (or terminal emulator) can be serially connected to the UART (via RS232) and used to control the system.

10.1.1 MC68307 Signal Configuration

BUSW0 is tied low, meaning that the MC68307 boots up in 8-bit mode after reset. Alternatively, BUSW0 could use a pull-up resistor to select 16-bit mode at reset, requiring a further 8-bit or a more expensive 16-bit EPROM.

Clock crystal connection is simple. Figure 10-1 shows a typical configuration. The usual oscillator start-up capacitors and bias resistor are included on-chip. The crystal parameters do not need to be particular values; C_0 (Shunt capacitance) < 10pF and Rx = 50 ohms were used in this case. An oscillator module could be used if preferred, connected to the EXTAL pin with XTAL unconnected.

CLKOUT is typically used as the reference clock for external system peripherals. In this case it remains unconnected.

The $\overline{\text{RSTIN}}$ input can be used to extend the power-on reset time if desired. If slower system peripherals require an extended $\overline{\text{RESET}}$ at power-up, an RC network attached to $\overline{\text{RSTIN}}$ can be used. In this case, the standard 32K clock stretch time at power-on suffices. At 16.67MHz, the 32K clocks equate to a 2ms $\overline{\text{RESET}}$ hold-on interval allowing the clock and power supply to normalize. $\overline{\text{RSTIN}}$ is attached to a debounced button to cold reset the MC68307 (and the system if other devices are connected to the $\overline{\text{RESET}}$ output). On each reset button press, the device is held in reset for the same 32K clocks as power-on.

$\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are bidirectional open drain lines, requiring low value pull-up resistors to maintain the inactive state when not asserted. $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are not used here, but could be asserted together to reset the system hardware if required. In this case the reset is not lengthened internally by 32k clocks.

$\overline{IRQ7}$ is used as a non-maskable software abort interrupt source, driven by a debounced abort button. Whenever the button is pressed, a level 7 interrupt is asserted to the MC68307. At the next instruction boundary the interrupt request is acknowledged, with the MC68307 generating the corresponding interrupt vector. In fact, the MC68307 always supplies all its own interrupt vectors, whether the interrupt request is generated internally (by on-chip modules), or externally ($\overline{IRQ7}$, $\overline{INT8}$–$\overline{INT1}$). Therefore, the interrupt vector numbers should be set up in the MC68307's PIVR to point to the appropriate vector location in memory.

$\overline{DTACK}$ is a bidirectional open drain line used to terminate bus accesses. In this design, $\overline{DTACK}$ is generated internally and asserted as an output using the MC68307's internal chip select and wait state control. External decode may also assert $\overline{DTACK}$ as an input.

$\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, and R/$\overline{W}$ control signals are all outputs. As the main asynchronous bus controls, each line has a pull-up resistor such that they are in the inactive state when they are not actively driven. They are not driven during reset, arbitration or CPU stop periods.

Chip select outputs ($\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$ and $\overline{CS3}$) are always driven and do not need pull-up resistors.

The 8051-compatible control lines (ALE, $\overline{RD}$, $\overline{WR}$) are all unused outputs.

The JTAG input lines (TDI, TMS, TCK) are not used, so are pulled inactive through resistors. The TDO output is not connected.

Port A and port B signals default to general purpose inputs after reset. If they are to remain unconnected, they should be reprogrammed as outputs. Otherwise pull-up resistors are necessary to ensure known input logic levels. All the port lines except PB2–PB5 remain unconnected and are reprogrammed as outputs. Port lines PB2–PB5 are used in their dedicated UART function. The TXD and $\overline{RTS}$ output lines use pull-up resistors to ensure logic levels between booting-up as a general purpose input and reprogramming as a UART output.

A23 uses a pull-up resistor to select normal MC68307 processor mode at power-up. If instead it is pulled low through a resistor, a slave mode is selected. Do not connect A23 to a supply rail directly. Always use a pull-up/pull-down resistor.

Address/data/control buses are three-stated when the MC68307 is arbitrated off the bus, held in reset, or the CPU is stopped (via LPEN bit in SCR). If the buses are to be three-stated for an extended period of time, pull-up resistors are recommended for these lines. The pull-up resistor value is typically in the range 20k to 50k ohms, subject to leakage current and loading requirements.

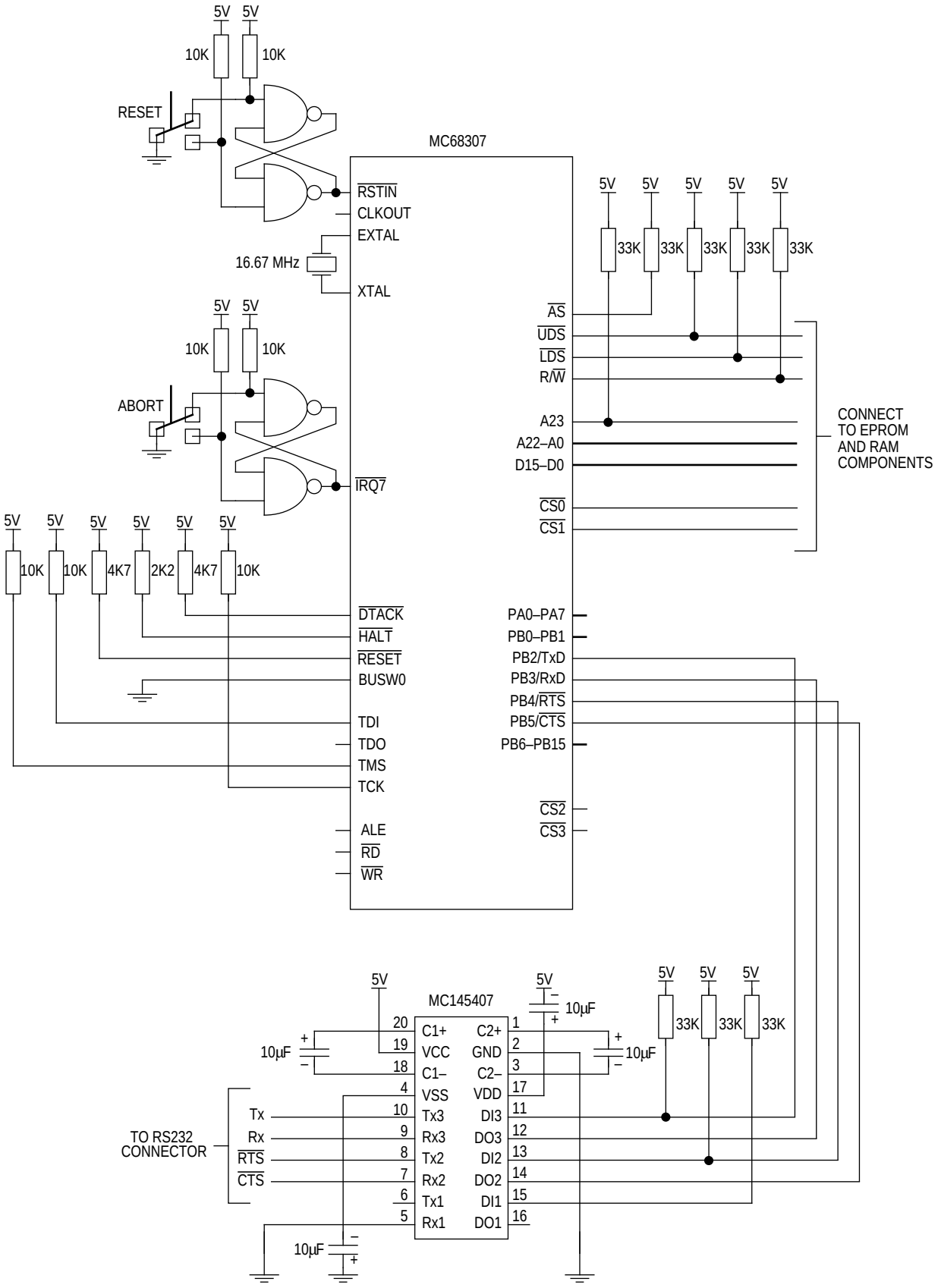


Figure 10-1. MC68307 Minimum System Configuration (1 of 2)

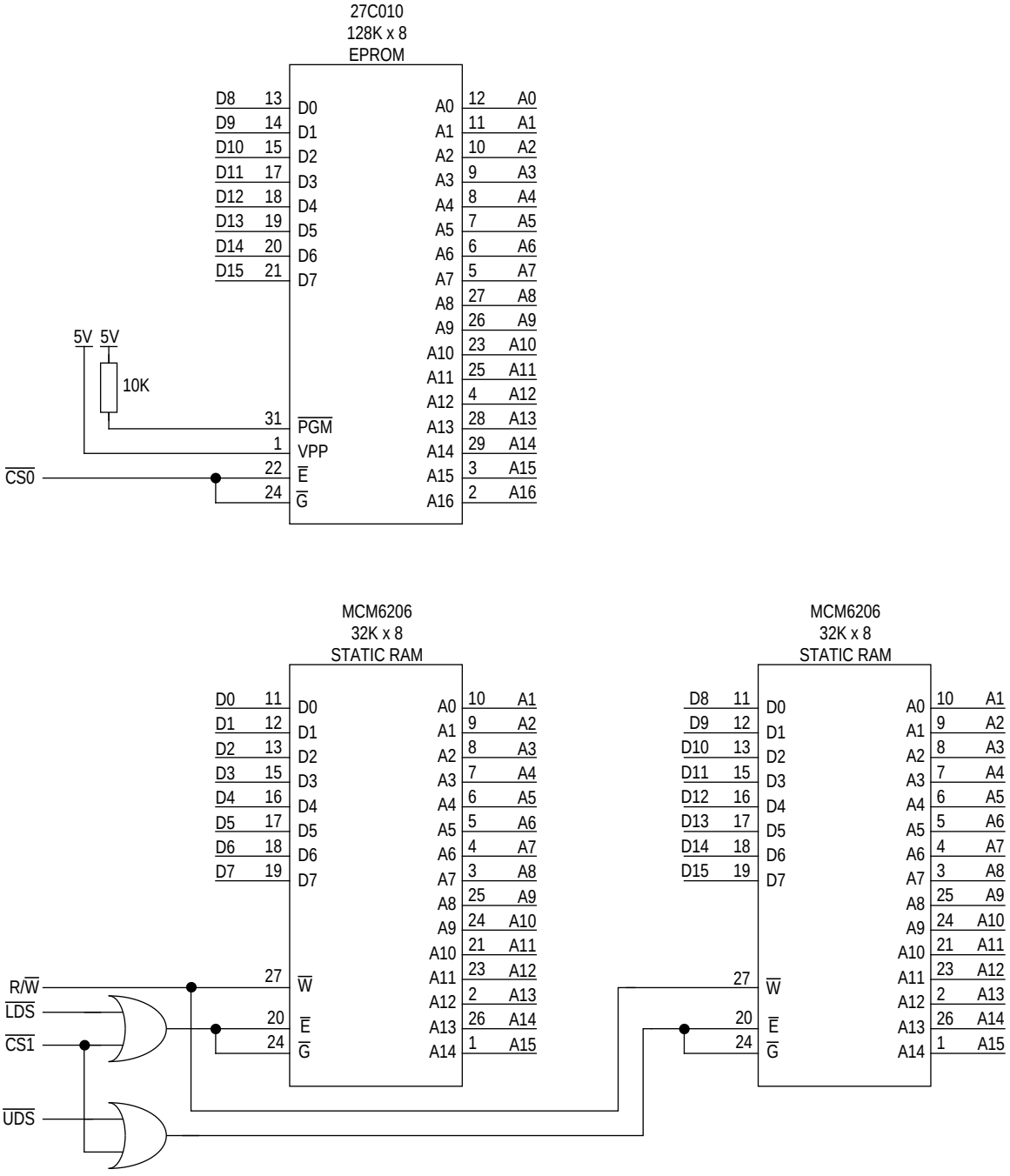


Figure 10-1. MC68307 Minimum System Configuration (2 of 2)

10.1.2 EPROM Memory Interface

The BUSW0 signal is used to select either an 8- or 16-bit EPROM mode for boot-up after reset. In this case BUSW0 is tied low, and a standard, low cost, 8-bit EPROM is used. The EPROM interface is a straight connection of $\overline{CS0}$, address and data. The 8 data lines are connected on the upper half of the data bus (D15–D8).

$\overline{CS0}$ is used to select the EPROM and enable its output for a specified address range. After reset, the default $\overline{CS0}$ programming is for the first 8 Kbytes address range, accessible with 6 wait states. The CS0 registers are usually reprogrammed as one of the first software activities, to set up the desired address space and wait state options.

Some processor systems need a read/write control signal to the EPROM output enable to avoid data contention in the event of an accidental write to ROM. The MC68307 can avoid this by programming $\overline{CS0}$ to assert for read accesses only.

Note that with BUSW0 pulled high, the 16-bit EPROM interface using two 8-bit EPROMs is equally simple. Both EPROMs are chip selected by $\overline{CS0}$. The upper (even) byte EPROM is output enabled by $\overline{UDS}$, while the lower (odd) byte EPROM is output enabled by $\overline{LDS}$. The address lines used by two 27C010's would be A17–A1.

10.1.3 RAM Memory Interface

The 16-bit RAM interface is controlled using $\overline{CS1}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$, address and data. $\overline{CS1}$ is used to select RAM for read/write accesses within a programmed address range. It must be software configured before any RAM accesses are possible. For example, stacking operations such as interrupt or exception handling, and subroutine branches are not possible until CS1 is configured. In this instance, CS1 should be programmed as a 16-bit port, with suitable address range, address space and wait states.

The $\overline{UDS}$/$\overline{LDS}$ signals are used to qualify accesses to the upper/lower byte of the selected RAM data word. The R/$\overline{W}$ dictates whether a RAM read or write takes place.

It simplifies the interface to output enable the RAMs each time they are chip selected. Although, when using alternative RAMs, care should be taken that the RAMs do not drive data out when output enabled during a write. Almost all RAMs including the MCM6206 can do this.

Note that for an 8-bit RAM interface using an MC6206, $\overline{CS1}$ could be connected to the chip enable, $\overline{UDS}$ to the output enable, R/$\overline{W}$ to the read/write, A14–A0 to the address lines and D15-D8 to the data bus. After reset, $\overline{CS1}$ must be software configured for 8-bit mode.

10.1.4 RS232 UART Port

The MC68307's UART signals (TxD, RxD, $\overline{RTS}$, $\overline{CTS}$) are connected to an MC145407 RS232 level driver. On the RS232 side, the UART signals are made available to a standard RS232 connector. This creates a communication channel between the user and system. For instance, a terminal (or terminal emulator) could control the system via a user preferred debug monitor coded into the EPROM; this is an invaluable debug feature during the development stages of a system design.

10.1.5 EPROM Timing

The EPROM interface offers zero wait state accesses with a 100ns EPROM. A timing worth special note is the EPROM enable to EPROM data out time. $\overline{CS0}$ is asserted on the rising edge of S2, and the selected EPROM has to return valid data a setup time before the falling edge of S6 (an access time of approximately 2.5 clocks). At 5V and 16.67MHz, the equation for the EPROM access time is as follows:

2.5 clocks – data setup time – clock to CS asserted time (maximum)= 150 – 5 – 30 = 115ns

Programming $\overline{CS0}$ for 1 wait state would increase this access time to 175ns, while 6 wait states permit an access time up to 475ns at 16.67MHz, and 5V.

For slow memories, the EPROM chip enable high to output float time should be confirmed to ensure the EPROM-driven data is removed before the start of the next bus cycle (next rising edge of S2). Although the exact timing is dependent upon the system, a maximum time of 1 clock from $\overline{CS0}$ negated to EPROM data floating is a good guideline for use with the MC68307.

10.1.6 RAM Timing

The R/$\overline{W}$ signal is always setup prior to the RAM chip enable, so the timing is always RAM enable (E) controlled rather than write (W) controlled. Besides the additional delay introduced by the OR gate for decoding the RAM upper/lower byte enables, the RAM read timing is the same as that of the EPROM. It is the RAM write timing that dictates the RAM access time requirements.

The RAM chip enable to end of write time determines the necessary speed of RAM. The chip enable time begins on the rising edge of S4 when the data strobes ($\overline{UDS}$, $\overline{LDS}$) are asserted, and ends when the data strobes/$\overline{CS1}$ negate, approximately 1 clock later. This typically includes RAMs with access times of up to 60–70ns, for zero wait states at 16.67MHz and 5V (check specific RAM manufacturer’s data sheets). The MCM6206s used have a 35ns access time.

For proper timing of write cycles, it is also important to know how much time elapses between CS negated and the data output hold time. With a 5V, 16.67MHz MC68307, the data out hold time is a 15ns minimum. Thus, on the enable controlled RAM write cycle, the RAM data-in hold time requirement must be ≤15ns OR gate delay. When using a 74F32 OR gate, the RAM data-in hold time must be ≤ 7ns and while using a 74AC32 it must be ≤ 5ns. For the MCM6206, it is 0.

10.2 POWER MANAGEMENT

The fully static MC68307 has a number of flexible power management features that enable the power consumption to be minimized for different applications. This section describes how to use these features to best effect in reducing overall system power consumption.

10.2.1 Fully Static Operation

The MC68307 is fully static; this means that the clock to the device can be stopped completely without causing damage. In addition the clock can be restarted with no loss of status occurring.

These two fundamental benefits of fully static operation are utilized in the MC68307 power management features. The MC68307 allows the clocks to different modules within the device to be stopped under software control in order to minimize power consumption for a particular user application. The external clock input can also be varied in frequency between DC (i.e., stopped externally) and the maximum operating frequency (16.67 MHz).

By setting the appropriate bits in the configuration register (MBCD, TMCD, UACD, CKD and LPEN), the internal clocks to the M-bus, timer, UART, CLKOUT and CPU can be individually stopped. With the exception of the CPU, these clocks can all be restarted by clearing the corresponding bit in the configuration register. When a module clock is turned off, the module current consumption is reduced to very low leakage levels.

Table 10-1 gives typical percentage power savings that can be made by stopping the clocks to modules that are not used in a particular operating mode. These figures are from measurements taken on a development board. They represent percentages of full operating power consumption and are averages over four different test cases (16.67MHz and 8MHz at 5V and 3.3V).

Table 10-1. Power Contribution from Modules

Module	Percentage I_{DD}
All Operational	100%
CLKOUT	5%
M-bus	6%
Timer	7%
UART	15%
CPU (EC000)	61%
Low-Power Sleep	6%

10.2.2 Prescalable CPU Clock

It is apparent from Table 10-1 that the EC000 CPU core contributes the most significant percentage of the total power consumption. As well as being able to stop the clock to the CPU completely (when the CPU clock is stopped it cannot process code) this clock can be selectively programmed to a prescaled value. This is achieved by writing to the CDEN bit in the configuration register, with the CD2–CD0 bits set to select the required prescale value.

Only the CPU clock is divided by this prescale value, the enabled clocks to the peripheral modules remain at the system clock frequency. Hence the module interfaces will operate as normal.

This mode of operation is useful when the CPU is required to do some background housekeeping tasks, which do not require the full processing power of the CPU, but one or more of the peripheral modules is required to be operating at full speed. The CPU can subsequently select full speed clock by writing to the configuration register when an event occurs that requires the full processing power.

The power consumption of the CPU is approximately proportional to the frequency, hence by carefully selecting the correct prescale value for the current task, the power contribution of the CPU to the overall device can be minimized.

10.2.3 Wake-Up

When the CPU clock has been stopped in low-power sleep mode it can only be restarted by a wake-up condition or a system reset. Any event that causes an interrupt wakes-up the CPU. For an interrupt to occur, the corresponding module must be clocked and the interrupt priority level must be set to nonzero in the corresponding field of the interrupt control register.

It is also possible to set a bit in the configuration register (UACW) that automatically restarts the clock to the UART when a falling edge is detected on the RxD line. Any subsequent interrupt from the UART module wakes-up the CPU.

This very flexible feature enables the CPU to wake-up in response to many different events, for example:

—If UART receives a character
—If M-bus activity is detected
—After a fixed time by programming a timer
—If an external interrupt occurs

Because there is no loss of data in the fully static CPU core, the wake-up latency is reduced to the minimum of only 10 system clocks. When the CPU wakes-up it immediately processes the interrupt and then returns to the mainline code from which it stopped its own clock.

A STOP instruction is not absolutely necessary when the MC68307 is put into low-power sleep mode, although, if such an instruction is used it will be handled correctly by the CPU. Care must be taken with the interrupt mask since any interrupt wakes-up the CPU, but only interrupts of priority greater than the mask cause the CPU to exit from the sleep instruction.

The wake-up latency is increased slightly to 24 system clocks when a STOP instruction is used. Note that the MC68307 does not execute the LPSTOP instruction used by some other Motorola microprocessors (for example, MC68060, MC68340, and MC68360).

All enabled interrupts wake-up the CPU. If the interrupt mask in the CPU is set to a level greater than the active interrupt, this interrupt will not be processed immediately by the CPU. This technique gives the absolute minimum wake-up latency, since time-critical code can be placed immediately after the instruction that stopped the CPU clock, but before the interrupt mask is lowered to process the interrupt which caused the wake-up.

The MC68307 does not require or generate a reset on wake-up. If the MC68307 is reset the CPU clock and all the other internal clocks are restarted, however the wake-up latency will be large since reset exception processing is required.

10.2.4 Low-Power Sleep Mode

When all the internal clocks are stopped, including the CPU, the MC68307 is said to be in low-power sleep mode.

In low-power sleep mode the external clock, either crystal oscillator or square wave input, is still toggling. The only logic internal to the MC68307 that is getting clocked is the reset circuit and interrupt controller. The device can wake-up from low-power sleep mode by an external interrupt on $\overline{\text{IRQ7}}$ or any of the $\overline{\text{INTx}}$ inputs. In addition the UART clock can automatically restart when a falling edge is detected on RxD, and subsequently wake-up the CPU when an interrupt occurs.

The CPU is arbitrated off the system bus when it enters low-power sleep mode. The address and data buses are three-stated. For a system that has the MC68307 in this mode for prolonged periods of time, the system address and data buses should be pulled up using high value resistors (e.g., 22–100Kohms depending upon signal loading) to prevent these buses floating to mid-range voltage levels and hence generating large currents in input buffers.

When a crystal is connected across EXTAL and XTAL pins on the MC68307 the contribution of the oscillator cell to the sleep-mode current is large. Depending on the type of crystal used, this bias current can be up to several milliamps. Careful choice of crystal and board layout can help minimize the effect of the bias current, but it will always make a significant contribution to sleep current. If an external oscillator is used to generate a square wave input to EXTAL, the bias current is much less. When an external oscillator is used, the XTAL pin should be left completely unconnected.

Low-power sleep mode consumes about 4% to 6% of the fully operational power depending on the configuration of the oscillator pins.

10.2.5 Low-Power Stop Mode

The power consumption of the MC68307 can be minimized by stopping the clock to the EXTAL pin externally. This is referred to as low-power stop mode. External hardware is required to control the system clock circuit. The current consumption of the MC68307 is at an absolute minimum in this mode.

Note that since the external clock input to the MC68307 is stopped the device cannot wake-up from low-power stop mode directly.

Because the MC68307 is fully static, the clock can be stopped at any time and in any phase. Lowest power will be obtained if the clock is stopped after the device has been put into low-power sleep mode. If the clock is stopped when the CPU is executing a bus cycle or processing an instruction the current will not be minimal. Low-power sleep mode uses arbitration to ensure that the CPU is idle before stopping the internal clock.

When the clock is restarted, the MC68307 will continue processing from where it was previously with no loss of status. If the device is in low-power sleep mode an interrupt will be required to wake-up the CPU.

Instead of stopping the clock externally the frequency could be reduced to a very low value in order to minimize power. The whole MC68307 could operate at this reduced frequency, allowing timers for example to wake-up the CPU after a long delay and subsequently set a general purpose port line to reselect the maximal frequency external clock.

Whenever the external clock is being stopped, started or multiplexed, care must be taken to ensure that no pulses narrower than the specified minimum clock high or low periods are generated (minimum pulse width = 27 ns). If an external crystal is stopped then started again there may be considerable wake-up latency due to the crystal start-up time. This depends on the external component selection and is not a function of the MC68307.

10.3 USING M-BUS SOFTWARE TO COMMUNICATE BETWEEN PROCESSOR SYSTEMS

M-bus is an I^2C-compatible bus interface used in the MC68300 family. It is a serial interface comprising two open drain bidirectional signals, namely serial clock (SCL) and serial data (SDA). Multiple devices can be connected directly to these open drain lines, and indeed this is good reason for the widespread adoption of the bus as an efficient IC communication method in end-systems.

A typical scenario would consist of a processor with an M-bus master controlling the data flow between several slaves, such as LCDs, real-time clocks, keypads, A/D converters and memories. Moreover, a built-in bus collision mechanism supports multiple M-bus masters as well as multiple slaves. The M-bus module of the MC68300 is flexible enough to operate as either an M-bus master or a slave.

This section demonstrates control software for M-bus communication between two identical MC68307 systems, one configured with an M-bus master and the other an M-bus slave. Only a short piece of initialization code must be changed to make the MC68307 code applicable to other MC68300 devices with M-bus.

10.3.1 Overview of M-Bus Software Transfer Mechanism

For clarity, a brief overview of the M-bus software control mechanism is provided here.

The M-bus communication is on a byte-wide basis. The components of the hardware transfer protocol are a START condition, 8 data bits, an acknowledge bit and a STOP condition. Before starting a communication, an M-bus master should carry out a software check to ensure the bus is free, and therefore all other M-bus transfers are complete. Thereafter, the bus master initiates a transfer by software, writing a START condition onto the bus. This is an indicator to all connected M-bus devices that the master is taking charge of the bus and that the address of the targeted slave is to follow. For the MC68300 M-bus master, writing the targeted slave address to the data register initiates the 8-bit transfer (MSB first).

If a system has two or more M-bus masters which poll the bus free and start a transfer at the same time, then the collision detection arbitration throughout the transfer of the slave address transfer and subsequent data bytes decides which device gets charge of the bus. If the MC68300 M-bus loses arbitration in this way, it stops driving data onto the bus to prevent data corruption. Furthermore, it switches automatically into slave mode, pre-empting the alternate master addressing it as a slave. If interrupts are enabled, an interrupt is generated on the completion of that byte, and a status bit indicates arbitration lost as the interrupt source.

The first data byte transmitted by the M-bus master is always the targeted slave address, with the least significant bit determining whether the slave remains ready to receive or transmit subsequent bytes. The addressed slave can then acknowledge the received byte, or not, depending upon the software protocol and acknowledge capability of the slave devices used. Each acknowledge is like a 9th data bit, asserted by the receiver as a handshake to successfully transmitted data.

A block transfer comprising a series of data bytes (and acknowledge bits) follows as commanded by the software protocol. The bus remains busy throughout the block, precluding all other masters from starting transfers. At the end of the block, the bus master relinquishes the bus by software placing a STOP condition onto the bus.

Ultimately, the M-bus master is responsible for starting and stopping transfers, but the number of bytes transferred can be dictated by either the master or slave depending upon the desired software protocol. For example, a slave may acknowledge all bytes received until it saturates, at which point the master STOPs the block transfer. Alternatively, the slave receiver may acknowledge received bytes until the master transmitter indiates that there are no more bytes to send. Indeed, both master and slave can be charged with controlling the transfer block. For instance, the software protocol may transfer a byte count as part of the communication or use a fixed number of transfer bytes every time.

For the best choice in software control, transfers can adopt either a status polling method or interrupts at the end of each byte. The interrupt option is most commonly used to minimize the processor overhead or the time during which the processor is tied up with the transfers. If enabled, the interrupts are generated on the completion of each 9 bits (8 data bits plus an acknowledge).

10.3.2 M-Bus Master Mode Operation

Using interrupts to transmit data to the addressed slave is straight forward. During the M-bus initialization, the MC68300 M-bus sets up the master transmitter mode, sets the M-bus frequency, enables interrupts, provides an interrupt handler and STARTs the block transfer. The targeted slave address (with LSB = 1 for slave receiver mode) is transmitted by writing to the M-bus data register. On each subsequent end-of-byte interrupt, further data bytes are transmitted by writing data to the M-bus data register until the block is complete. On the interrupt at the end of the last byte the software STOPs the transfer.

For receiving from the addressed slave, the initialization is exactly the same. Remember that even if receiving, the first operation is to transmit the targeted slave address (except that this time LSB = '0'). In the interrupt handler at the end of the slave address transmit byte, the transmit mode is changed to receive. Then, to initiate the first byte receive operation, the MC68300 M-bus master software carries out a dummy read of the data register. No sensible data is read at this point, but it is the action of this read which starts the data receive. At the end of each subsequent received byte, the interrupt generated is used to read the data register again for valid data, and to start the next byte receive. This continues until the master receiver STOPs the block transfer.

The receiver is always responsible for the generation of acknowledges. The MC68300 M-bus receiver can be programmed to generate acknowledges automatically for each byte received. Most slave transmitters take an acknowledge from the master receiver to mean that further bytes are desired. In fact, for some slave transmitters it is necessary for the master receiver to acknowledge all received bytes except the last one, to indicate that more data byte transmits are required. This is not a requirement of the MC68300 M-bus slave.

10.3.3 M-Bus Slave Mode Operation

Many of the principles discussed for the master operation also hold true for the slave MC68300 M-bus. The main differences are that the MC68300 slave M-bus is not controlling the transfer (STARTing and STOPping) or providing the M-bus clock, but is instead following what the master dictates.

For slave operation, again initialize the M-bus frequency, M-bus slave address, interrupt handler and interrupt enable. As the first transfer is always the receipt of the slave address, slave receive mode should always be programmed initially. All target slave addresses which are transmitted by the master (first byte after START) are then checked against the programmed MC68300 M-bus slave address for a match. When they match, the MC68300 M-bus slave generates an interrupt (if enabled), and a status bit indicates the cause as M-bus addressed-as-slave (MAAS).

On entering the corresponding interrupt handler, the software read/write status indicator is read to determine whether the slave is to receive or transmit subsequent bytes and that the transmit/receive mode is set accordingly. If in transmit mode, the first data byte transmit is initiated by writing to the data register. If in receive mode, the first receive byte is initiated by a dummy read of the data register. There is no sensible data read at this point, but having started the receive process, data register reads in subsequent end-of-byte interrupts obtain

valid data and initiate the next byte receive. Again, the software protocol between M-bus master and slave determines the use of acknowledges.

10.3.4 Description of Setup

The hardware consists of two identical MC68307 systems connected together via the M-bus as shown in Figure 10-2. Both systems have the MC68307 processor core executing instructions prefetched from its own ROM.

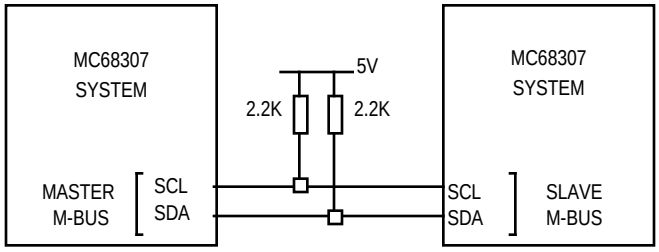


Figure 10-2. Hardware Setup

Each MC68307 system has 128 Kbytes EPROM and 128 Kbytes SRAM and runs a debug monitor. A complete description of the system hardware is provided in AN490, titled “Multiple Bus Interfaces using the MC68307.

Using the monitor's download facility, an M-bus control program is downloaded into the SRAM of each board. The code allows one system to control its M-bus module as a master, while the other implements an M-bus slave. Together, the two software programs allow the MC68307 M-bus master to write data to the MC68307 M-bus slave and later read it back for verification.

10.3.5 Software Flow

The MC68307 master M-bus controls the number of blocks transferred via START and STOP conditions. In this example, there are only two communication blocks, one transmitting data to the slave (master transmit block), and one receiving data back from the slave for verification (master receive block). The master/slave responsibilities during the master transmit block are outlined in Figure 10-3 and for the master receive block in Figure 10-5.

On these diagrams, note that for a given transfer byte, the end-of-byte interrupts on the master and slave occur at around the same time. The built-in M-bus transfer mechanism means it does not matter in which order they are serviced. The master and slave interrupt service order used in the flowcharts of Figure 10-3 and Figure 10-5 is purely for demonstration purposes. The interrupt handlers are shown such that the data flow is always from transmitter to receiver, although, it should be understood that the master and slave interrupt handlers are happening at the same time, as are the transmit and receive of a particular byte between the two systems.

10.3.6 Transfer Blocks

The master M-bus controls of the number of data bytes within each transmit/receive block. Observe Figure 10-4 and Figure 10-6, which give a summary of the activity on the M-bus during the master transmit and master receive blocks respectively.

When the master is transmitting data (master transmit block) the slave acknowledges all bytes received, and the master decides when the transfer is completed by setting a STOP condition (see Figure 10-4). When the master is receiving data (master receive block) it decides when the transfer is complete by stopping acknowledges on the last received byte, thereby stopping the slave transmitting, and setting a STOP condition (see Figure 10-6).

10.3.7 Software Implementation

The software used is shown in **Section 10.3.7.1 Software Listing 1—M-bus Master Software** and **Section 10.3.7.2 Software Listing 2—M-bus Slave Software**. Only the method of enabling the M-bus and interrupts at the start of the software listings is specific to the MC68307. Thereafter, the code is generic for any MC68300 device with an M-bus module.

The MC68300 M-bus slave software should always be set running before the master software, such that the prospective slave is initialized as a receiver before the master transmits the slave address.

The software uses interrupts to control the byte transfers within each block. The M-bus master starts the transfer by transmitting the slave address. Thereafter interrupts are generated on both the master and slave M-bus to control the test. The M-bus hardware protocol does not care which order the interrupts are serviced by the master (transmitter or receiver) or slave (transmitter or receiver) at the end of each byte. Consider that the master is in charge of generating the SCL clocks to shift data out the transmitter and into the receiver, when a transmit/receive is initiated by writing/reading the M-bus data register respectively. However, the clocks do not start until the slave has released the clock line on the bus by making its corresponding read/write of its M-bus data register. Therefore, both MC68300 M-bus master and slave interrupts have to initiate the next data transfer.

The slave frequency can be programmed as greater or less than that of the master. M-bus implements a clock synchronization mechanism such that the clock with the shortest high time and longest low time dictates the open drain clock. For example, if the programmed slave M-bus clock frequency is less than the master, the slave can stretch the clock as necessary.

The number of transfer and receive blocks and the number of data bytes within each block can be altered in the master software. The slave software remains the same throughout. If the user desires detailed crosschecks on the software flow, interrupt counts (for number of bytes transferred) or a flag passing mechanisms could be implemented. For simplicity this is not used in the example software.

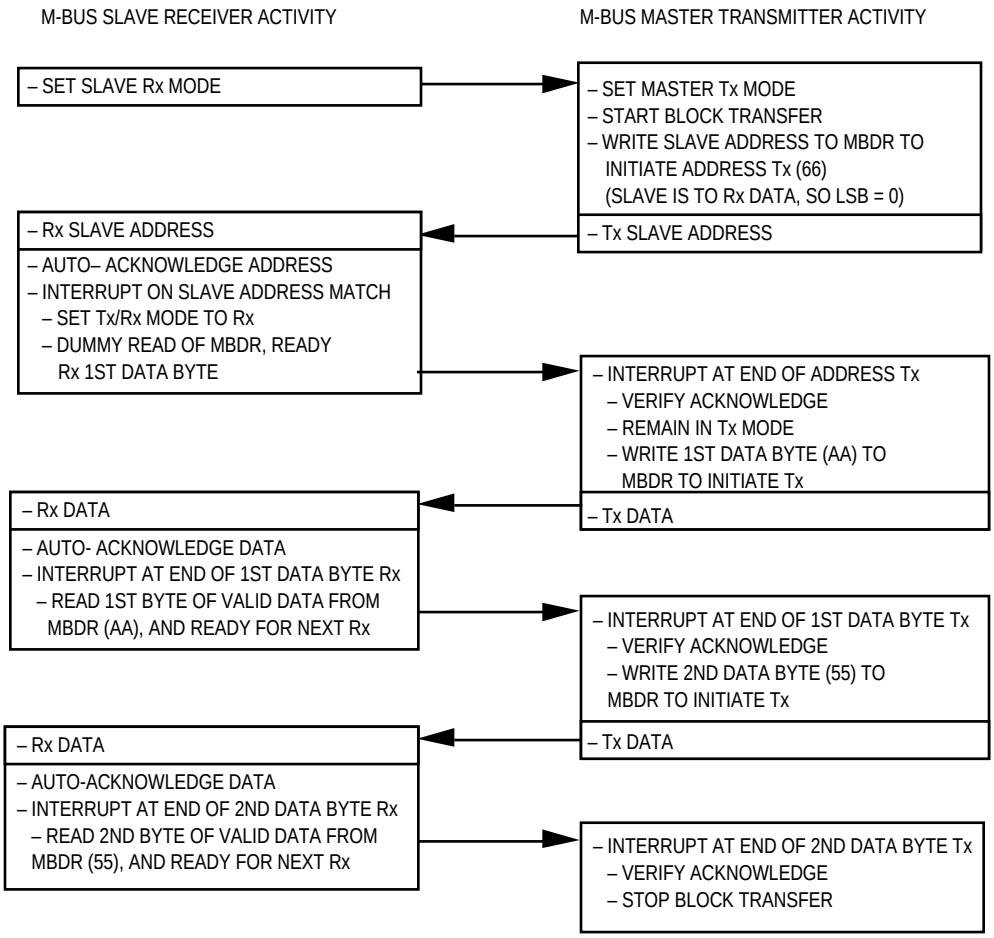


Figure 10-3. Master/Slave Responsibilities for the Master Transmit Block

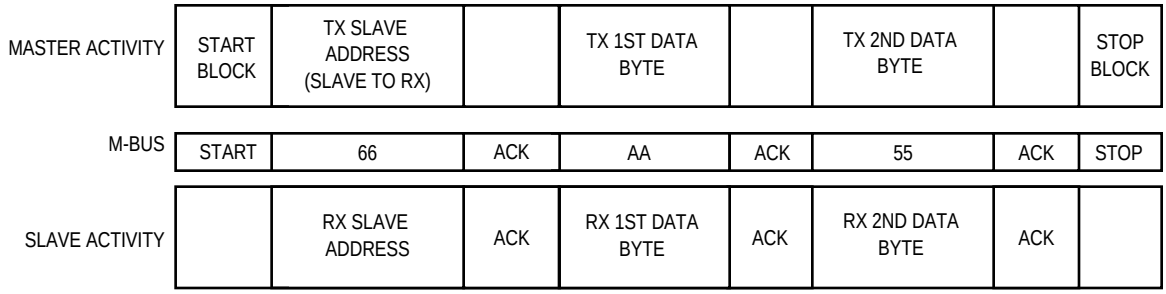


Figure 10-4. Summary of M-Bus Activity for the Master Transmit Block

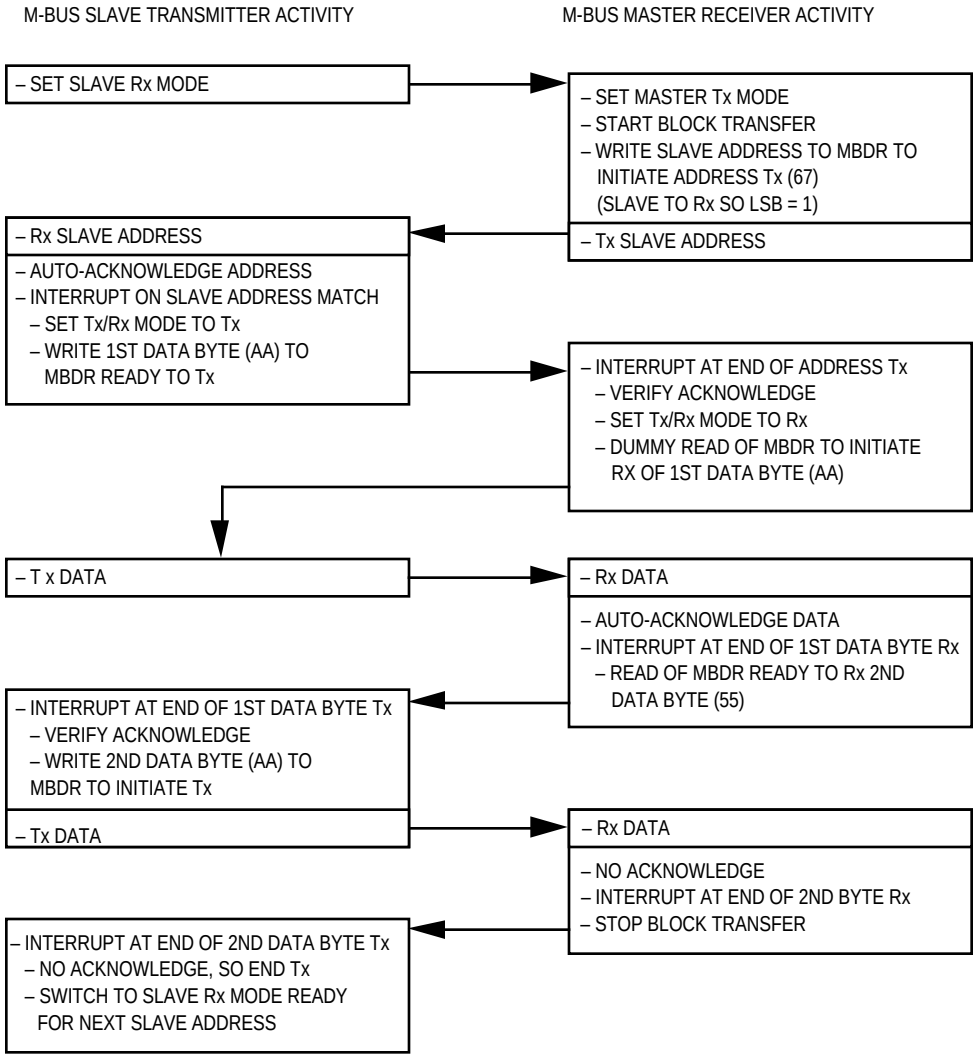


Figure 10-5. Master/Slave Responsibilities for the Master Receive Block

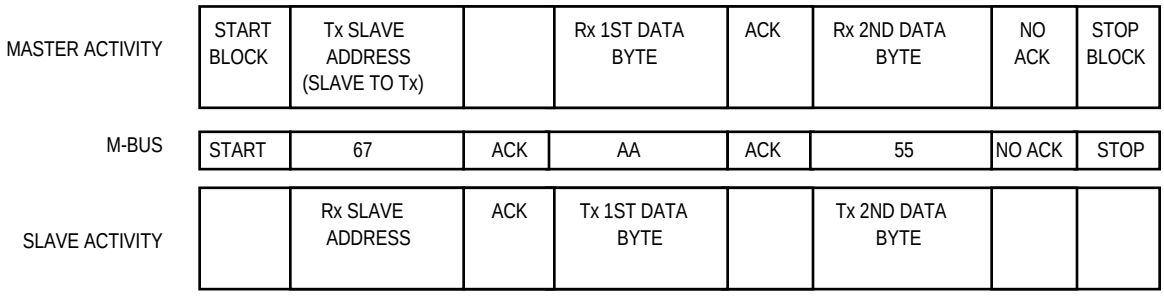


Figure 10-6. Summary of M-Bus Activity for the Master Receive Block

10.3.7.1 SOFTWARE LISTING 1—M-BUS MASTER SOFTWARE

```
*------------------------------------------------------------------------
*             MOTOROLA 68307 IMBP TEST BOARD - M-bus
*------------------------------------------------------------------------
*
*        MODULE:   MBM_INT.SRC      DATE:         8/4/94
*
*        Developed by :   Motorola
*                         HI-END Applications
*                         East Kilbride.
*
*        NOTES:
*        Master M-bus Routine using interrupts for a Master/Slave Test
*
*        The number of bytes transmitted and received is completely
*        controlled by the master. (i.e. When the slave is receiving data,
*        it acknowledges all the time, and the master dictates the number of
*        bytes to transfer. When the slave is transmitting, the master
*        receiver acknowledges dictate whether the slave is to send further
*        bytes or not.
*
*        The Master:
*        1) Writes out the slave chip address, and 2 slave data bytes.
*        2) Writes out the slave chip address, and reads 2 slave data bytes
*        3) Verifies the data read back against that originally sent.
*
*------------------------------------------------------------------------
*        External Reference Declarations
*------------------------------------------------------------------------
          XREF    SCR             ; System Control Reg
          XREF    PBCNT           ; Port B Control Reg
          XREF    PIVR            ; Peripheral Interrupt Vector Reg
          XREF    PICR            ; Peripheral Interrupt Control Reg
          XREF    MADR            ; M-bus Address Reg
          XREF    MFDR            ; M-bus Freq Divider Reg
          XREF    MBCR            ; M-bus Control Reg
          XREF    MBSR            ; M-bus Status Reg
          XREF    MBDR            ; M-bus Data Reg

*------------------------------------------------------------------------
*        Constants
*------------------------------------------------------------------------
UVECBASE   EQU     $100            ; User Vector Base
MBUSVEC    EQU     UVECBASE+($D*4) ; M-bus vector location
MBUSHAN    EQU     $15000          ; M-bus Interrupt Handler location

S307_AD    EQU     $66             ; Slave 68307 M-bus Address
DRXCNT     EQU     $3              ; Data RECEIVE COUNT (2 + 1 Dummy)
ATXCNT     EQU     $1              ; Address TRANSMIT COUNT
DTXCNT     EQU     $2              ; Data TRANSMIT COUNT

*------------------------------------------------------------------------
*        Main Program
*------------------------------------------------------------------------
```

```
            ORG     $10000           ; Random location for assembly
            AND.L   #$FFFFFEFF,SCR   ; Clear SCR bit 8, M-bus CLock Active
            MOVE.B  #$40,PIVR        ; Vector = #$40, Vector @ address $100
            OR.W    #$000D,PICR      ; M-bus Interrupt level = 5
            MOVE.L  #MBUSHAN,MBUSVEC ; Set up M-bus Interrupt Handler
            OR.W    #$0003,PBCNT     ; Enable M-bus Lines
            BSR     INIT_MBM         ; Initialize M-bus as master

* WRITE TO SLAVE 68307 M-bus
* Write Chip Address, and Two bytes of data
START       BSR     MBBUSY           ; Poll the M-bus, wait till bus free

            MOVE.B  #0,V_DRXCNT      ; Data RECEIVE COUNT
            MOVE.B  #ATXCNT,V_ATXCNT ; Address TRANSMIT COUNT
            MOVE.B  #DTXCNT,V_DTXCNT ; Data TRANSMIT COUNT
            MOVE.B  #1,V_WRITE       ; Set Write to slave var = TRUE
            MOVE.B  #S307_AD,V_CHIPAD; Slave 68307 M-bus receiver Addr.
            MOVE.L  #S307_DATA,A0    ; Pointer to stored data for transfer

            BSRWRITE1                ; Send out the Chip Address

* READ FROM SLAVE 68307 M-bus
* Write Chip Address, and READ Two bytes of data

            BSR     MBBUSY           ; Poll the M-bus, wait till bus free

            MOVE.B  #DRXCNT,V_DRXCNT ; Data RECEIVE COUNT
            MOVE.B  #ATXCNT,V_ATXCNT ; Address TRANSMIT COUNT
            MOVE.B  #0,V_DTXCNT      ; Data TRANSMIT COUNT
            MOVE.B  #0,V_WRITE       ; Set Write to slave var = FALSE
            MOVE.B  #S307_AD,D6      ; Alter chip address lsb for
            OR.B    #$01,D6          ; slave transmit and
            MOVE.B  D6,V_CHIPAD      ; write to chip address variable
            MOVE.L  #S307_DATA,A0    ; Pointer to data for memory 1

            BSR     WRITE1           ; Send out the Chip Address

*Test Complete
            BSR     MBBUSY           ; Poll the M-bus, wait till bus free
FOREVER     BRA     FOREVER          ; Test complete & passed, loop forever

*-----------------------------------------------------------------------
*        M-Bus Setup/Initialization
*-----------------------------------------------------------------------
INIT_MB     MMOVE.W #$2700,SR        ; Disable interrupts – set to level 7
            MOVE.B  #0,MBSR          ; Clear interrupt pend, arbitr. lost
            MOVE.B  #$0C,MFDR        ; Set frequency
            MOVE.B  #$00,MBCR        ; Disable and reset M-bus
            MOVE.B  #$80,MBCR        ; Enable M-bus
            RTS
* NOTE - By not writing MADR, the 68307 M-bus slave address = 0

*-----------------------------------------------------------------------
*          Poll the M-bus BUSY
*-----------------------------------------------------------------------
```

```
MBBUSY      BTST    #5,MBSR          ; Test MBB bit,
            BNE     MBBUSY           ; and wait until it is clear
            RTS

*-----------------------------------------------------------------------
*         Generation first byte of data transfer
*-----------------------------------------------------------------------
WRITE1      BTST    #5,MBSR          ; Test MBB bit,
            BNE     WRITE1           ; and wait until it is clear
TXSTART
            BSET    #4,MBCR          ; Set TRANSMIT Mode
            BSET    #5,MBCR          ; Set Master Mode (Generate START)
            BSET    #6,MBCR          ; Enable M-bus Interrupts
            MOVE.W  #$2000,SR        ; Enable interrupts – set to level 0

MBFREE      BTST    #5,MBSR          ; Test MBB bit,
            BEQ     MBFREE           ; If bus is still free, wait until busy
            RTS

*-----------------------------------------------------------------------
*         Post Byte Transmission/Reception Software Response
*-----------------------------------------------------------------------
            ORG     MBUSHAN          ; Start of Interrupt Handler
ISR         BCLR    #1,MBSR          ; Clear the MIF Flag
            MOVE.L  D0,-(A7)         ; Push D0 Register to Stack
            MOVE.L  D1,-(A7)         ; Push D1 Register to Stack
            MOVE.L  #0,D0            ; Clear general data reg
            MOVE.L  #0,D1            ; Clear general data reg

            BTST    #5,MBCR          ; Check the MSTA Flag
            BEQ     SLAVE            ; Branch if Slave Mode

            BTST    #4,MBCR          ; Check the MODE Flag
            BEQ     MASTRX           ; Branch if Receive Mode

*-----------------------------------------------------------------------
*         Master TRANSMIT caused Interrupt
*-----------------------------------------------------------------------
MASTX       BTST    #0,MBSR          ; Check ACK From Receiver,
            BNE     ENDMASTX         ; If no ACK, End Transmission

TXADDR      MOVE.B  V_ATXCNT,D1      ; Check Address TX COUNT
            BEQ     TXDATA           ; If address complete go to data
            SUBQ.B  #1,V_ATXCNT      ; Decrement Address Tx Count

TXDATA      MOVE.B  V_WRITE,D1       ; Check if writing or reading slave
            BEQ     SETMASTRX        ; If reading set to Master receive

            MOVE.B  V_DTXCNT,D1      ; CHECK Data TX COUNT
            BEQ     ENDMASTX         ; If no more data then STOP bit
            SUBQ.B  #1,V_DTXCNT      ; Reduce Tx Count
            MOVE.B  (A0)+,MBDR       ; Transmit next byte
            BRA     END              ; Exit

ENDMASTX    BCLR    #5,MBCR          ; Generate STOP Condition
```

```
            BRA     END             ; Exit

SETMASTRX   BCLR    #3,MBCR         ; Enable TXAK
            BCLR    #4,MBCR         ; Set Master Receive Mode
            BSET    #5,MBCR         ; Set Master Mode (Generate START)

*-----------------------------------------------------------------------
*        Master RECEIVE
*-----------------------------------------------------------------------
MASTRX      SUBQ.B  #1,V_DRXCNT     ; Decrement receive count
            MOVE.B  V_DRXCNT,D1
            CMP.B   #DRXCNT-1,D1    ; First byte read Check
            BNE     NOTFIRST        ; If not first, read etc. as usual
            MOVE.B  MBDR,D0         ; If first, DUMMY read only to start Rx
            BRA     END

NOTFIRST    CMP.B   #0,D1
            BEQ     ENMASR          ; Last byte to be read check
            SUBQ.B  #1,D1           ; Last second byte to be read check
            BNE     NXMAR           ; Not last one or last second, so branch
LAMAR       BSET    #3,MBCR         ; Last second, disable ACK transmitting
            BRA     NXMAR

ENMASR      BCLR    #5,MBCR         ; Last one, generate STOP signal

NXMAR       MOVE.B  MBDR,D0         ; Read data
            CMP.B   (A0)+,D0        ; Compare with written data
            BEQ     END             ; If data as expected o.k.

READERR     BRA     READERR         ; Else ERROR loop forever.

END         MOVE.L  (A7)+,D1        ; Pop D1 Register From Stack
            MOVE.L  (A7)+,D0        ; Pop D0 Register From Stack
            RTE

SLAVE       NOP
            BRA     SLAVE           ; Slave operation not implemented

*-----------------------------------------------------------------------
*        Buffers and Variables
*-----------------------------------------------------------------------
V_WRITE     DC.B    $1              ; Slave write = True
V_CHIPAD    DC.B    S307_AD         ; Chip Address variable = Slave 307 Add
V_DRXCNT    DC.B    DRXCNT          ; Set up variables - Data Receive Count
V_ATXCNT    DC.B    ATXCNT          ; - Addr Transmit Count
V_DTXCNT    DC.B    DTXCNT          ; - Data Transmit Count
S307_DATA   DC.B    $AA,$55         ; Chip 1 Data

                    END
```

10.3.7.2 SOFTWARE LISTING 2—M-BUS SLAVE SOFTWARE

```
*------------------------------------------------------------------------
*             MOTOROLA 68307 IMBP TEST BOARD -  M-bus
*------------------------------------------------------------------------
*
*        MODULE:   MBM_INT.SRC      DATE:         8/4/94
*
*        Developed by :   Motorola
*                         HI-END Applications
*                         East Kilbride.
*
*        NOTES:
*        Slave M-bus Routine using interrupts for a Master/Slave Test
*
*        The number of bytes transmitted and received is completely
*        controlled by the master. (i.e. When the slave is receiving data,
*        it acknowledges all the time, and the master dictates the number of
*        bytes to transfer. When the slave is transmitting, the master
*        receiver acknowledges dictate whether the slave is to send further
*        bytes or not.
*
*        The Slave:
*        1) Recognizes its slave chip address, and receives 2 data bytes.
*        2) Recognizes its slave chip address, and transmits the 2 bytes
*------------------------------------------------------------------------
*        External Reference Declarations
*------------------------------------------------------------------------
           XREF    SCR              ; System Control Reg
           XREF    PBCNT            ; Port B Control Reg
           XREF    PIVR             ; Peripheral Interrupt Vector Reg
           XREF    PICR             ; Peripheral Interrupt Control Reg
           XREF    MADR             ; M-bus Address Reg
           XREF    MFDR             ; M-bus Freq Divider Reg
           XREF    MBCR             ; M-bus Control Reg
           XREF    MBSR             ; M-bus Status Reg
           XREF    MBDR             ; M-bus Data Reg

*------------------------------------------------------------------------
*        Constants
*------------------------------------------------------------------------
UVECBASE   EQU     $100             ; User Vector Base
MBUSVEC    EQU     UVECBASE+($D*4)  ; M-bus vector location
MBUSHAN    EQU     $15000           ; M-bus Interrupt Handler location

S307_AD    EQU     $66              ; Slave 68307 M-bus Address

*------------------------------------------------------------------------
*        Main Program
*------------------------------------------------------------------------
           ORG     $10000           ; Random location for assembly
           AND.L   #$FFFFFEFF,SCR   ; Clear SCR bit 8, M-bus Clock Active
           MOVE.B  #$40,PIVR        ; Vector = #$40, Vector @ address $100
           OR.W    #$000D,PICR      ; M-bus Interrupt level = 5
           MOVE.L  #MBUSHAN,MBUSVEC ; Set up M-bus Interrupt Handler
```

```
            OR.W    #$0003,PBCNT     ; Enable M-bus Lines
            BSR     INIT_MBS         ; Initialize M-bus as slave

FINISH      BRA     FINISH           ; Loop forever

*-----------------------------------------------------------------------
*        M-Bus Setup/Initialization
*-----------------------------------------------------------------------
INIT_MBS    MOVE.W  #$2700,SR        ; Disable interrupts – set to level 7
            MOVE.B  #0,MBSR          ; Clear interrupt pend, arbitr. lost
            MOVE.B  #$10,MFDR        ; Set frequency
            MOVE.B  #S307_AD,MADR    ; Set M-bus slave address
            MOVE.B  #$00,MBCR        ; Disable and reset M-bus
            OR.B    #$C0,MBCR        ; Enable M-bus, Ints, TXAK
            MOVE.W  #$2300,SR        ; Enable INTS by setting to level 3
            RTS

*-----------------------------------------------------------------------
*        Poll the M-bus BUSY
*-----------------------------------------------------------------------
MBBUSY      BTST    #5,MBSR          ; Test MBB bit,
            BNE     MBBUSY           ; and wait until it is clear
            RTS

*-----------------------------------------------------------------------
*        Post Byte Transmission/Reception Software Response
*-----------------------------------------------------------------------
            ORG     MBUSHAN          ; Start of Interrupt Handler
ISR         BCLR    #1,MBSR          ; Clear the MIF Flag
            MOVE.L  D0,-(A7)         ; Push D0 Register to Stack
            MOVE.L  D1,-(A7)         ; Push D1 Register to Stack
            MOVE.L  #0,D0            ; Clear general data reg
            MOVE.L  #0,D1            ; Clear general data reg
* Interrupt Counter
            ADDQ.L  #1,D3            ; (Not used, simply monitor)

            BTST    #5,MBCR          ; Check the MSTA Flag
            BEQ     SLAVE            ; Branch if Slave mode

MASTER      BRA     MASTER           ; Master not implemented, so error

*-----------------------------------------------------------------------
SLAVE       MOVE.B  MBSR,D6          ; Read MBSR
            BTST.B  #6,D6            ; Is it slave address byte?
            BEQ     SLAVE_DATA       ; If not, then data

*-----------------------------------------------------------------------
*        Addressed as Slave
*-----------------------------------------------------------------------
SLAVE_ADD   BTST    #2,D6            ; Read SRW to verify slave Tx or Rx
            BEQ     INIT_SRX         ; If Rx, initialize Slave receive count

INIT_STX    OR.B    #$10,MBCR        ; Set transmit mode
            MOVE.L  #DATABUF,A0      ; Pointer to data storage buffer
            MOVE.B  (A0)+,MBDR       ; First data byte transmit
```

```
            BRA     END_SLAVE

INIT_SRX    AND.B   #$E7,MBCR       ; Set receive mode and TXAK
            MOVE.L  #DATABUF,A0     ; Pointer to data storage buffer
            MOVE.B  MBDR,D0         ; Start receive via Dummy byte read
            BRA     END_SLAVE

*-----------------------------------------------------------------------
*        Slave Data
*-----------------------------------------------------------------------
SLAVE_DATA  BTST    #4,MBCR         ; Read Tx or Rx mode
            BEQ     SRX_DATA

*-----------------------------------------------------------------------
*        Post Slave data Transmit Control
*-----------------------------------------------------------------------
STX_DATA    BTST    #0,MBSR         ; Check ACK From Receiver,
            BEQ     NXT_TX          ; If ACK, then TX Next Data Byte

            AND.B   #$EF,MBCR       ; TX complete so swap to Rx
            MOVE.B  MBDR,D0         ; Dummy read to free bus (SCL)
            BRA     END_SLAVE       ; Finish and await Master

NXT_TX      MOVE.B  (A0)+,MBDR      ; Tx next data byte
            BRA     END_SLAVE       ; Exit

*-----------------------------------------------------------------------
*        Post Slave data Receive Control
*-----------------------------------------------------------------------
SRX_DATA    MOVE.B  MBDR,D0         ; Read Data
            MOVE.B  D0,(A0)+        ; Store data in next buffer location

END_SLAVE   MOVE.L  (A7)+,D1        ; Pop D1 Register from Stack
            MOVE.L  (A7)+,D0        ; Pop D0 Register from Stack
            RTE

*-----------------------------------------------------------------------
*        Buffers and Variables
*-----------------------------------------------------------------------
DATABUF     DS.B    0               ; Slave data buffer between Rx and Tx

            END
```

10.4 MC68307 UART DRIVER EXAMPLES

The MC68307 UART driver code listed in this section was developed from MC68681 DUART driver software. The two principal changes described below are (a) the implementation of one UART channel, and (b) the use of the UART timer as the UART baud rate source. The UART driver code is given in **Section 10.4.1 Software Listing 3**. It provides well commented initialization, status, input and output routines.

(a) Where the DUART implemented two serial channels, the MC68307 UART implements one. Accordingly, the memory map has changed to include only those registers appropriate to the single channel (channel A).

(b) For peak integration of the MC68307, all internal module clocks are based on the single processor clock. So, while the MC68681 based the standard baud rate settings of its clock select register on a fixed 3.6864MHz external clock source, the MC68307's baud rate is developed from its 16.67MHz to D.C. range processor clock source. Therefore, the MC68307 uses its programmable UART timer to prescale this processor clock source, and generate the baud rate reference as desired.

The equation to calculate the UART timer prescaler for a desired baud rate is:

Baud rate generator prescaler = 68307 clock frequency/[16 x 2 x (UART baud rate)]

where, Baud rate generator prescaler = {UBG1:UBG2}

e.g., For 9600 baud from a 16.67MHz clock, the UART timer prescaler is 54 decimal (36 Hex). So, UBG1 = 0x00 and UBG2 = 0x36.

NOTE

The standard baud rate settings of the MC68307's UART clock select register (UCSR) are only relevant if the processor clock frequency is a suitable multiple of 3.6864 MHz.

10.4.1 Software Listing 3

```
SIO         EQU     $FFF101          ; Serial IO Base Address

* PORTW     EQU     1                ; Byte wide port
PORTW       EQU     2                ; Word wide port
* PORTW     EQU     4                ; Long word wide port

CON_INDEX   EQU     0

UMR1        EQU     0*PORTW          ; Mode register 1
UMR2        EQU     0*PORTW          ; Mode register 2
USR         EQU     1*PORTW          ; Status register
UCSR        EQU     1*PORTW          ; Clock select register
UCR         EQU     2*PORTW          ; Command register
URB         EQU     3*PORTW          ; Receiver buffer register
UTB         EQU     3*PORTW          ; Transmitter buffer regiseter
UACR        EQU     4*PORTW          ; Auxiliary control register
```

```
UISR       EQU     5*PORTW          ; Interrupt status register
UBG1       EQU     6*PORTW          ; Counter/timer upper register
UBG2       EQU     7*PORTW          ; Counter/timer lower register

BRGSET1    EQU     0                ; Bit 7 selects set 1 if clear
*-------------------------------------------------------------------------
CRGSET2    EQU     $E0              ; Bit 7 selects set 2 if set
                                    ; Timer mode, divided by 1
SCALEU     EQU     0                ; Upper Byte Prescaler for Timer
SCALEL     EQU     54               ; Lower Byte Prescale for Timer
*-------------------------------------------------------------------------
CON_SR     EQU     SIO+CON_INDEX+USR
CON_RH     EQU     SIO+CON_INDEX+URB
CON_CR     EQU     SIO+CON_INDEX+UCR
CON_TH     EQU     SIO+CON_INDEX+UTB

BRKRECD_BIT   EQU         7         ; Break received if set
ERRFRAME_BIT  EQU         6         ; Framing error if set
ERRPAR_BIT    EQU         5         ; Parity error if set
ERROVER_BIT   EQU         4         ; Overrun error if set
TXEMT_BIT     EQU         3         ; Transmitter empty if set
TXRDY_BIT     EQU         2         ; Transmitter ready
FFULL_BIT     EQU         1         ; Fifo full if set
RXRDY_BIT     EQU         0         ; Receiver ready if set

RSTMRPTR   EQU     $10              ; Reset mode register pointer
RSTRCVR    EQU     $20              ; Reset receiver
RSTXMIT    EQU     $30              ; Reset transmitter
RSTERROR   EQU     $40              ; Reset errors
RSTBRK     EQU     $50              ; Reset break change interrupt
SETBRK     EQU     $60              ; Start break
ENDBRK     EQU     $70              ; Stop break
DISTX      EQU     $08              ; Disable transmitter
ENATX      EQU     $04              ; Enable transmitter
DISRX      EQU     $02              ; Disable receiver
ENARX      EQU     $01              ; Enable receiver

           PAGE
*-------------------------------------------------------------------------
*        INIT PROCEDURE (BOTH PORTS)
*        INPUT:  NONE
*        OUTPUT: NONE
*-------------------------------------------------------------------------
CON_INIT:
           MOVEM.L D1/A0-A1,-(SP)   ; Save registers
           MOVE.L  #2000,D1
CON_WAIT:
           BTST.B  #TXEMT_BIT,CON_SR; wait for xmtr to finish
           NOP
           DBNE D1,CON_WAIT
           LEA.L   SIO+CON_INDEX,A0 ; Console port
           MOVE.L  #$001307DD,D1    ; No parity
                                    ; 8 data bits
                                    ; 1 stop bit
                                    ; Timer for baud rate
```

```
            BSR     INIT              ; INIT console
            MOVEM.L (SP)+,D1/A0-A1    ; Restore registers
            RTS

INIT:
            MOVE.B  #RSTMRPTR,UCR(A0); Reset MODE register
            MOVE.B  #CRGSET2,SIO+UACR; Set the baud rate set
            MOVE.B  #SCALEU,SIO+UBG1 ; Set timer prescalar for 9600 baud
            MOVE.B  #SCALEL,SIO+UBG2 ;
            MOVE.B  #0,UISR(A0)       ; Clear INT status register
            ROL.L   #8,D1
            ROL.L   #8,D1
            MOVE.B  D1,UMR1(A0)       ; Setup MODE register 1
            ROL.L   #8,D1
            MOVE.B  D1,UMR2(A0)       ; Setup MODE register 2
            ROL.L   #8,D1
            MOVE.B  D1,UCSR(A0)       ; set baud rate clock as timer
            MOVE.B  #RSTRCVR,UCR(A0) ; Reset the transmitter
            MOVE.B  #RSTXMIT,UCR(A0) ; Reset the receiver
            MOVE.B  #ENARX,UCR(A0)    ; Enable receiver
            RTS

            PAGE
*-------------------------------------------------------------------------
*        STATUS PROCEDURE:
*        INPUT:  NONE
*        OUTPUT: D0 = 0  IF RECEIVE BUFFER EMPTY
*                   = 1  IF RECEIVE BUFFER FULL
*                   = 2  IF BREAK DETECTED
*-------------------------------------------------------------------------
CON_STS:
            MOVE.B  CON_SR,D0         ;Read status register
            BTST.B  #RXRDY_BIT,D0
            BEQ     CON_STS_1         ; nothing yet
            BTST.L  #BRKRECD_BIT,D0   ; Break?
            BEQ     CON_STS_1         ; No
            MOVE.B  CON_RH,D0         ; Read in break char
            MOVE.B  #RSTBRK,CON_CR    ; Req'd by 68681
            MOVE.B  #RSTERROR,CON_CR ;
            MOVE.B  #2,D0             ; Break status
            RTS
CON_STS_1:
            ANDI.B  #1,D0             ; Return 1 or 0
            RTS

*-------------------------------------------------------------------------
*        INPUT PROCEDURE
*        INPUT:  NONE
*        OUTPUT: D0 = CHARACTER
* THIS IS CALLED ONLY AFTER A STATUS OF 'BUFFER FULL' HAS BEEN READ
*-------------------------------------------------------------------------
CON_IN:
            MOVE.B  CON_RH,D0
            MOVE.B  #RSTERROR,CON_CR ;
            RTS
```

```
            PAGE
*-------------------------------------------------------------------------
*         OUTPUT PROCEDURE
*         INPUT:  D0.B = CHARACTER
*         OUTPUT: NONE
* CALLER ASSUMES CHARACTER IS 'LOGICALLY' WRITTEN AFTER THIS CALL
*-------------------------------------------------------------------------
CON_OUT:
            MOVE.B  #ENATX,CON_CR    ;Enable transmitter
CON_OUT_1:
            BTST.B  #TXRDY_BIT,CON_SR;Wait until
            BEQ.S   CON_OUT_1        ;transmitter ready
            MOVE.B  D0,CON_TH        ;Send char
            RTS

            END
```

10.5 SWAPPING ROM AND RAM MAPPING ON THE MC68307

It is often essential for embedded systems to locate the exception and interrupt vectors in read/write memory, to allow vector table changes after system boot-up. Indeed, before being able to port some debug monitors to an application this is a requirement.

For CPU32 based MC68300 integrated processors, the vector base register (VBR) resolves this issue, permitting vectors to be located anywhere in the address space. For MC68300 processors with an M68000 processor core (MC68302, MC68306, MC68307, etc.) this feature is not available, and a ROM/RAM swap technique is necessary to exchange the usual ROM at address $0 with RAM.

This section demonstrates a RAM/ROM swap mechanism specifically for the MC68307. However, the method can be applied equally to the other M68000 core based members of the MC68300 family.

10.5.1 Software Implementation

The swap mechanism relies on a section of position-independent code and careful reprogramming of the chip selects registers during boot up. **Section 10.5.1.1 Software Listing 4** provides full details of the code used.

Initially, the ROM and RAM chip selects reserve areas in the memory map at base address $0 and address $200000 respectively. Thereafter, a swap code routine is copied from ROM into RAM, before jumping to RAM to execute it. On completion, the execution returns to continue in ROM, which is now relocated at base address $080000. Finally, the RAM is then relocated at base address $0 as required. The resulting memory map is shown in Figure 10-7.

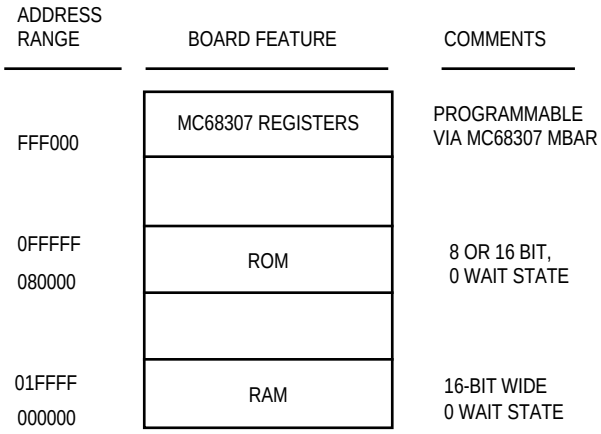


Figure 10-7. Memory Map after Swap Complete

The method described uses an application with 512 Kbytes of EPROM and 128 Kbytes of SRAM. If further details of the hardware are required, a full description is available in AN490, titled “Multiple Bus Interfaces Using the MC68307”.

The swap relies on the code being loaded at address $080000 during the link process, although, any other multiple of the ROM size could be used (e.g., $080000, $100000 and so on....), so long as the ROM chip select is programmed accordingly.

10.5.1.1 SOFTWARE LISTING 4

```
*-----------------------------------------------------------------------
*        68307 PROCESSOR REGISTERS.
REG_BASE   EQU     $FFF000          ; start of internal registers

MBAR       EQU     $0000F2          ; Base Address Register
SCR        EQU     $0000F4          ; System Control Register(32 bits)

BR0        EQU     REG_BASE+$040    ; CS0 Base Register
OR0        EQU     REG_BASE+$042    ; CS0 Option register
BR1        EQU     REG_BASE+$044    ; CS1 Base register
OR1        EQU     REG_BASE+$046    ; CS1 Option Register
WRR        EQU     REG_BASE+$12A    ; Software Watchdog Reference Reg

*-----------------------------------------------------------------------
TMP_BASE   EQU     $00200000        ; TEMP RAM BASE FOR ROM/RAM SWAP

*-----------------------------------------------------------------------
*        Start-up code section
SECT 9,c
           DC.L    STACK            ; Initial SP
           DC.L    $08              ; Initial PC

START      AND.W   #$FFFF,MBAR      ; 68307 register base at $FFF000
           MOVE.W  #$0000,WRR       ; Disable Software Watchdog

           MOVE.W  #$1F02,OR0       ; 0 waits, 512kB block, r/w not masked,
```

```
                                    ; FC's masked
            MOVE.W  #$C001,BR0      ; CS0 Supervisor program space, base $0,
                                    ; read, EN

            MOVE.W  #$1FC0,OR1      ; 0 waits, 128kB block, r/w bit masked,
                                    ; FC's masked
            MOVE.W  #$A401,BR1      ; CS1 Supervisor data space, base
                                    ; $200000, read, EN

*-----------------------------------------------------------------------
*        Copy swap code to RAM
*-----------------------------------------------------------------------
            LEA     SWAPC(PC),A0    ; Get start address of SWAP code
            LEA     SWAPCE(PC),A1   ; Get end address of SWAP code
            MOVEA.L #TMP_BASE,A2    ; Get address of temp RAM base
SWAPLP      CMPA.L  A0,A1           ; Has all code been copied?
            BEQ     JUMPSC          ; If so, jump to code RAM
            MOVE.W  (A0)+,(A2)+     ; Otherwise, copy ROM code to RAM
            BRA     SWAPLP          ; Continue in swapping loop
JUMPSC      JMP     (TMP_BASE).L    ; Execute SWAP code in RAM

*-----------------------------------------------------------------------
*        Code for RAM to $0 and ROM to $80000 swap
*        A section of code is set up to relocate ROM to $80000
*        This is copied to RAM, jumped to in RAM and executed,
*        before jumping back to ROM and relocating RAM at $0.
*-----------------------------------------------------------------------
SWAPC       MOVE.W  #$C101,BR0      ; Place ROM at $80000 by writing to BR0
            JMP     (SWAPCE).L      ; Return back to new ROM location
SWAPCE
            MOVE.W  #$A001,BR1      ; Place RAM at $0 by writing to BR1
```

SECTION 11
ELECTRICAL CHARACTERISTICS

This section contains detailed information on power considerations, DC/AC electrical characteristics, and AC timing specifications of the MC68307. Refer to **Section 12 Ordering Information and Mechanical Data** for specific part numbers corresponding to voltage, frequency, and temperature ratings.

11.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage[1, 2]	V_{CC}	–0.3 to + 7.0	V
Input Voltage[1, 2]	V_{in}	–0.3 to + 7.0	V
Operating Temperature Range	T_A	–40 to 85	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C

NOTES:
1. Permanent damage can occur if maximum ratings are exceeded. Exposure to voltages or currents in excess of recommended values affects device reliability. Device modules may not operate normally while being exposed to electrical extremes.
2. Although sections of the device contain circuitry to protectagainst damage from high static voltages or electrical fields, take normal precautions to avoid exposure to voltages higher than maximum-rated voltages.

This device contains protective circuitry against damage due to high static voltages or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

The following ratings define a range of conditions in which the device will operate without being damaged. However, sections of the device may not operate normally while being exposed to the electrical extremes.

11.2 THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance—Junction to Case Plastic 100-Pin QFP Plastic 100-Pin Thin QFP	θ_{JC}	20[1] 20[1]	°C/W
Thermal Resistance—Junction to Ambient Plastic 100-Pin QFP Plastic 100-Pin Thin QFP	θ_{JA}	42[1] 42[1]	°C/W

NOTE:
1. Estimated

11.3 POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \bullet \theta_{JA}) \quad (1)$$

where:

T_A = Ambient Temperature, °C
θ_{JA} = Package Thermal Resistance, Junction-to-Ambient, °C/W
P_D = $P_{INT} + P_{I/O}$
P_{INT} = $I_{CC} \times V_{CC}$, Watts—Chip Internal Power
$P_{I/O}$ = Power Dissipation on Input and Output Pins—User Determined

For most applications, $P_{I/O} < P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273°C) \quad (2)$$

Solving Equations (1) and (2) for K gives:

$$K = P_D \bullet (T_A + 273°C) + \theta_{JA} \bullet P_D^2 \quad (3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at thermal equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

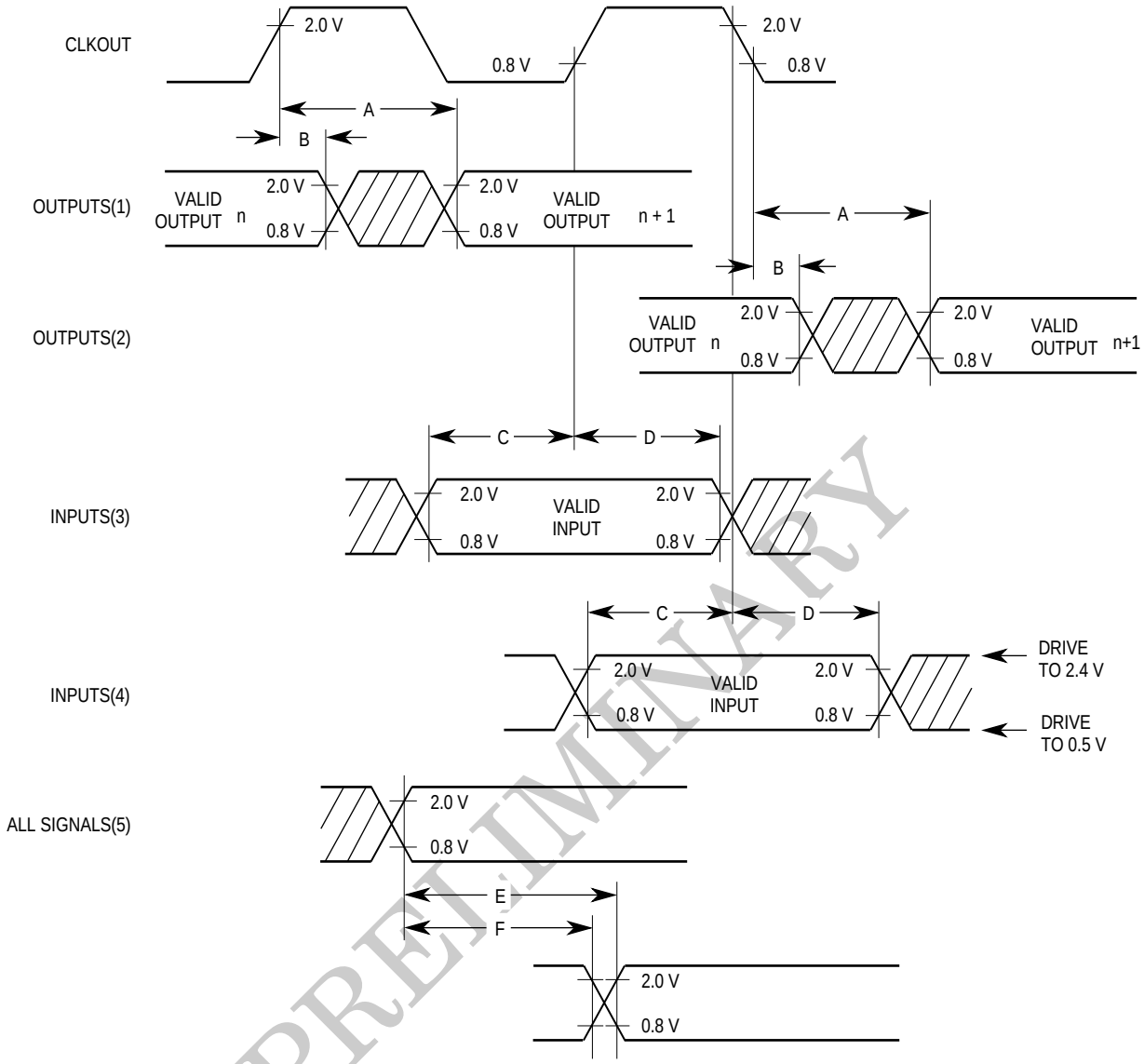
11.4 AC ELECTRICAL SPECIFICATION DEFINITIONS

The AC specifications presented consist of output delays, input setup and hold times, and signal skew times. All signals are specified relative to an appropriate edge of the clock and possibly to one or more other signals.

The measurement of the AC specifications is defined by the waveforms shown in Figure 11-1. To test the parameters guaranteed by Motorola, inputs must be driven to the voltage levels specified in that figure. Outputs are specified with minimum and/or maximum limits, as appropriate, and are measured as shown in Figure 11-1. Inputs are specified with minimum setup and hold times and are measured as shown. Finally, the measurement for signal-to-signal specifications is also shown.

NOTE

The testing levels used to verify conformance to the AC specifications do not affect the guaranteed DC operation of the device as specified in the DC electrical specifications.



NOTES:
1. This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2. This output timing is applicable to all parameters specified relative to the falling edge of the clock.
3. This input timing is applicable to all parameters specified relative to the rising edge of the clock.
4. This input timing is applicable to all parameters specified relative to the falling edge of the clock.
5. This timing is applicable to all parameters specified relative to the assertion/negation of another signal.

LEGEND:
A. Maximum output delay specification.
B. Minimum output hold time.
C. Minimum input setup time specification.
D. Minimum input hold time specification.
E. Signal valid to signal valid specification (maximum or minimum).
F. Signal valid to signal invalid specification (maximum or minimum).

Figure 11-1. Drive Levels and Test Points for AC Specifications

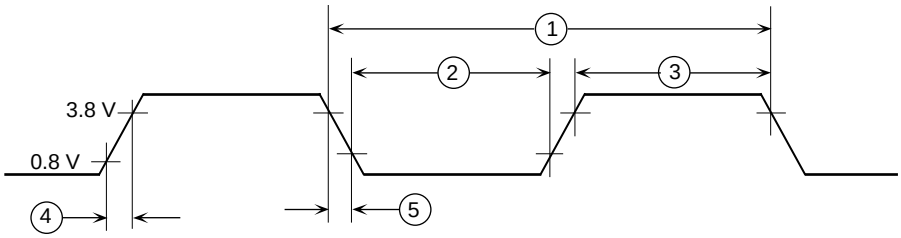
11.5 DC ELECTRICAL SPECIFICATIONS

Characteristic	Symbol	Min	Max	Unit
Input high voltage (except EXTAL)	V_{IH}	2.0	V_{CC}	V
Input low voltage	V_{IL}	GND – 0.3	0.8	V
Clock input high voltage (EXTAL)	VIHC	0.7 V_{CC}	V_{CC} + 0.3	V
Input leakage current @5.25V (all input-only pins)[1]	I_{IN}	– 2.5	2.5	µA
Three-state (off state) input current @2.4V/0.4V	I_{TSI}	—	20	µA
Output high voltage (I_{OH} = 4mA)	V_{OH}	V_{CC} – 0.75	—	V
Output low voltage (I_{OL} = 4mA)	V_{OL}	—	0.5	V
Current dissipation V_{CC} = 5.0V±0.5V[2], f_{EXT} = 16.67MHz V_{CC} = 3.3V±0.3V [2], f_{EXT} = 8MHz Low power SLEEP mode V_{CC} = 5.0V±0.5V, f_{EXT} = 16.67MHz V_{CC} = 3.3V±0.3V, f_{EXT} = 8MHz	I_D	 — — — —	 TBD TBD TBD TBD	mA
Power dissipation V_{CC} = 5.0V±0.5V, f_{EXT} = 16.67MHz V_{CC} = 3.3V±0.3V, f_{EXT} = 8MHz	P_D	 — —	 TBD TBD	W
Input capacitance[3] All input-only pins All I/O pins	C_{IN}	 — —	 10 20	pF
Load capacitance [3] All output pins (except SCL and SDA) SCL, SDA	C_L	 — —	 100 400	pF

NOTES:
1. Not including internal pull-up or pull-down.
2. Currents listed are with no loading.
3. Capacitance is periodically sampled rather than 100% tested.

11.6 AC ELECTRICAL SPECIFICATIONS—CLOCK TIMING (see Figure 11-2)

Num	Characteristic	3.3 V		3.3 V or 5 V	
		8.33 MHz		16.67 MHz	
		Min	Max	Min	Max
	Frequency of operation	0.0	8.33	0.0	16.67
1	Clock period (EXTAL)	120	—	60	—
2,3	Clock pulse width (EXTAL)	54	—	27	—
4,5	Clock rise and fall times (EXTAL)	—	5	—	5



NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 3.8 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 3.8 V.

Figure 11-2. Clock Timing

11.7 AC ELECTRICAL SPECIFICATIONS—READ AND WRITE CYCLES

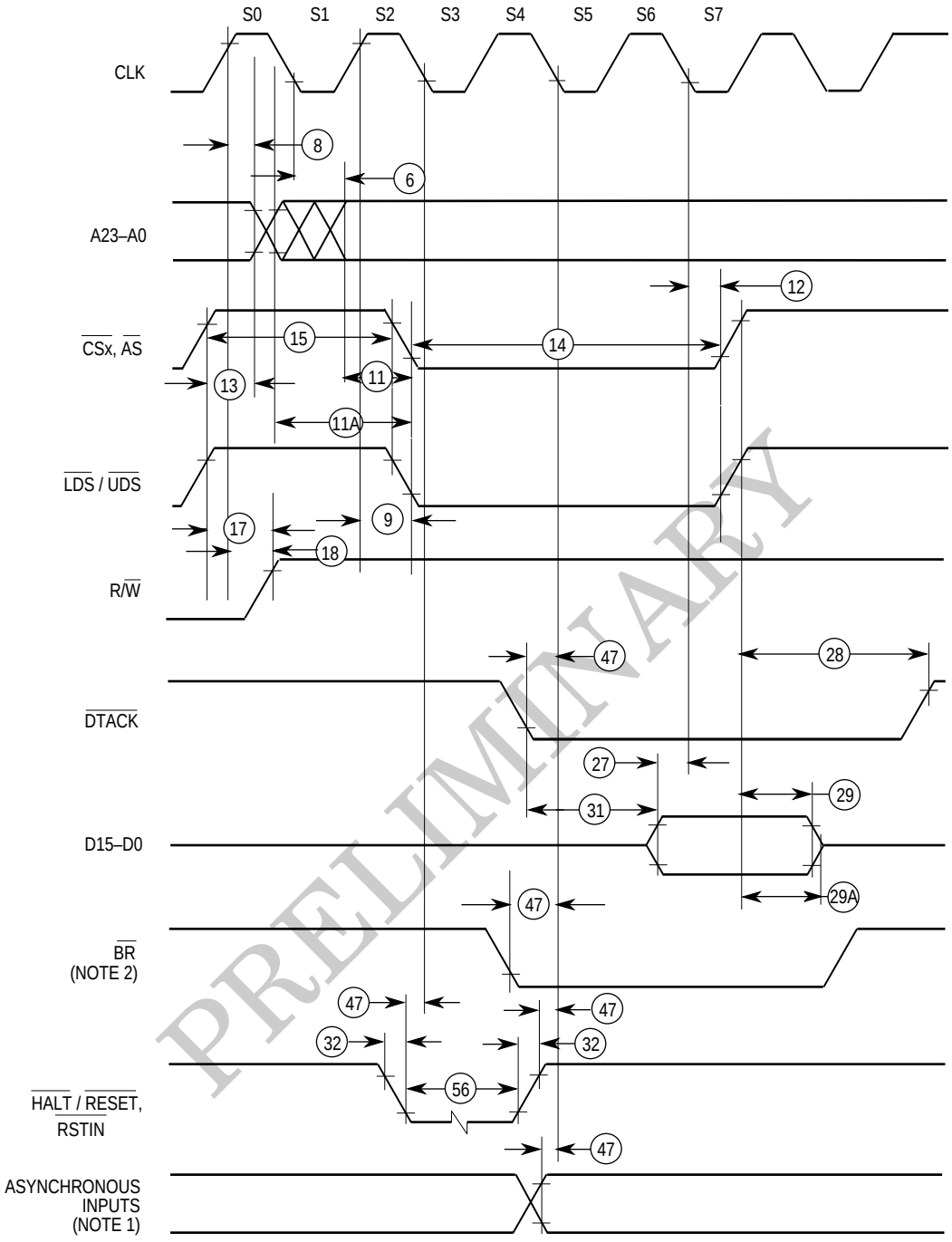
(V_{CC} = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; $T_A = T_L$ to T_H) (see Figure 11-3 and Figure 11-4**)**

Num	Characteristic	3.3V 8.33 MHz		3.3 V or 5 V 16.67 MHz		Unit
		Min	Max	Min	Max	
6	Clock low to address valid	—	60	—	30	ns
7	Clock high to address, data bus high impedance (maximum)	—	100	—	50	ns
8	Clock high to address invalid (minimum)	0	—	0	—	ns
9[1]	Clock high to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ asserted	3	60	3	30	ns
11[2]	Address valid to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ asserted (read)/$\overline{AS}$, $\overline{CSx}$ asserted (write)	30	—	15	—	ns
12[1]	Clock low to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated	3	60	3	30	ns
13[2]	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated to address invalid	30	—	15	—	ns
14[2]	$\overline{AS}$, $\overline{CSx}$, (and $\overline{LDS}$, $\overline{UDS}$ read) width asserted	240	—	120	—	ns
14A[2]	$\overline{LDS}$, $\overline{UDS}$ width asserted (write)	120	—	60	—	ns
15[2]	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ width negated	120	—	60	—	ns
16	Clock high to control bus high impedence	—	100	—	50	ns
17[2]	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated to R/$\overline{W}$ invalid	30	—	15	—	ns
18[1]	Clock high to R/$\overline{W}$ high (read)	0	60	0	30	ns
20[1]	Clock high to R/$\overline{W}$ low (write)	0	60	0	30	ns
20A[2,3]	$\overline{AS}$, $\overline{CSx}$, asserted to R/$\overline{W}$ low (write)	—	20	—	10	ns
21[2]	Address valid to R/$\overline{W}$ low (write)	0	—	0	—	ns
22[2]	R/$\overline{W}$ low to $\overline{LDS}$, $\overline{UDS}$ asserted (write)	60	—	30	—	ns
23	Clock low to data-out valid (write)	—	60	—	30	ns
25[2]	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated to data-out invalid (write)	30	—	15	—	ns
26[2]	Data-out valid to $\overline{LDS}$,$\overline{UDS}$ asserted (write)	30	—	15	—	ns
27[4]	Data-in valid to clock low (setup time on read)	10	—	5	—	ns
28[2]	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated to $\overline{DTACK}$ negated (asynchronous hold)	0	220	0	110	ns
29	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated to data-in invalid (hold time on read)	0	—	0	—	ns
29A	$\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$ negated to data-in high impedance	—	180	—	90	ns
31[2,4]	$\overline{DTACK}$ asserted to data-in valid (setup time)	—	100	—	50	ns
32	$\overline{HALT}$,$\overline{RESET}$ and $\overline{RSTIN}$ input transition time	0	300	0	150	ns
33	Clock high to $\overline{BG}$ asserted	0	40	0	20	ns

Num	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
34	Clock high to $\overline{BG}$ negated	0	40	0	20	ns
35[9]	$\overline{BR}$ asserted to $\overline{BG}$ asserted (for last cycle of operand transfer)	1.5	3.5	1.5	3.5	Clks
36	$\overline{BR}$ negated to $\overline{BG}$ negated	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ asserted to $\overline{BG}$ asserted	1.5	3.5	1.5	3.5	Clks
37A[2]	$\overline{BGACK}$ asserted to $\overline{BR}$ negated	40 ns	1.5	20 ns	1.5	Clks
38	$\overline{BG}$ asserted to control, address, data bus high impedance ($\overline{AS}$, $\overline{CSx}$ negated)	—	100	—	50	ns
39	$\overline{BG}$ width negated	1.5	—	1.5	—	Clks
46	$\overline{BGACK}$ width low	1.5	—	1.5	—	Clks
47[4]	Asynchronous input setup time	10	—	5	—	ns
53	Data-out hold from clock high	0	—	0	—	ns
55	R/$\overline{W}$ asserted to data bus impedance change	0	—	0	—	ns
56[5]	$\overline{HALT}$/$\overline{RESET}$ pulse width, $\overline{RSTIN}$ pulse width	10	—	10	—	Clks
57	$\overline{BGACK}$ negated to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$, R/$\overline{W}$ driven	1.5	—	1.5	—	Clks
58	$\overline{BR}$ negated to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$, R/$\overline{W}$ driven	1.5	—	1.5	—	Clks

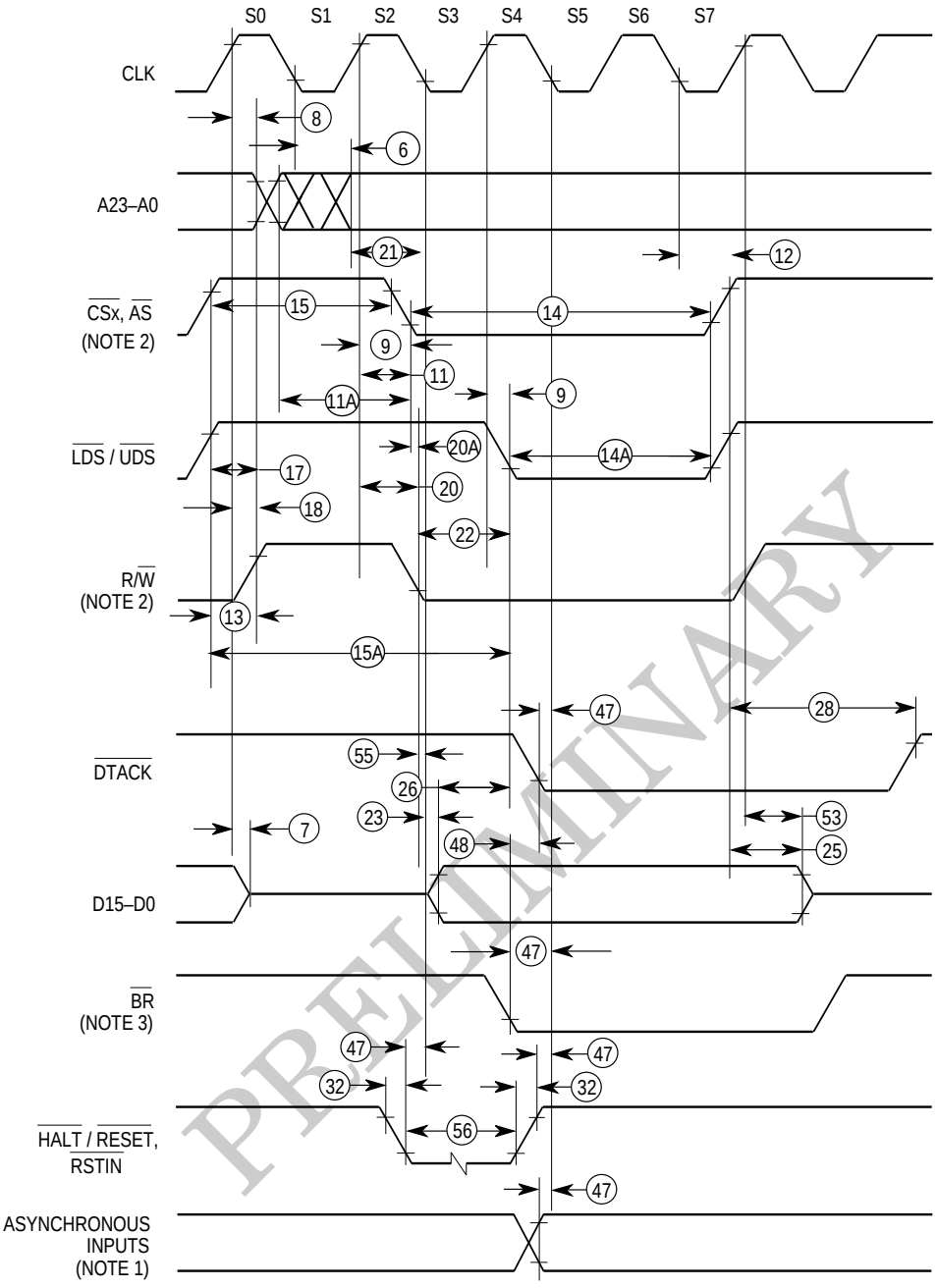
NOTES:

1. For a loading capacitance of less than or equal to 50 pF, subtract 5 ns from the value given in the maximum columns.
2. Actual value depends on clock period.
3. When $\overline{AS}$, $\overline{CSx}$ and R/$\overline{W}$ are equally loaded (±20%), subtract 5 ns from the values given in these columns.
4. If the asynchronous input setup time (#47) requirement is satisfied for $\overline{DTACK}$, the $\overline{DTACK}$ asserted to data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle.
5. For power-up, the MC68307 is held in the reset state for 32768 clock cycles after V_{CC} becomes stable to allow stabilization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width of $\overline{RESET}$/$\overline{HALT}$ required to reset the controller. This pulse is stretched internally to 132 clocks. If $\overline{RSTIN}$ is used, the pulse is stretched internally to 32768 clocks, and $\overline{RESET}$ and $\overline{HALT}$ are asserted as outputs.
6. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
7. The minimum value must be met to ensure proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reeasserted.
8. $\overline{AS}$ is always asserted, regardless of whether it is mapped to internal or external peripherals/memory. If the designer wishes to decode more chip selects than are provided, one of $\overline{CS0}$ – $\overline{CS3}$ should be used as the enable for the external decode.
9. During a read modify write cycle or a dynamically sized cycle, $\overline{BG}$ is delayed if $\overline{BR}$ is asserted before the last bus cycle of the operand transfer, in order to ensure operand coherency. $\overline{BG}$ will be asserted once $\overline{AS}$ has asserted for the last bus cycle of the transfer.



NOTES:
1. Setup time (#47) for asynchronous inputs ($\overline{\text{HALT}}$, $\overline{\text{RESET}}$, $\overline{\text{BR}}$, $\overline{\text{BGACK}}$, $\overline{\text{DTACK}}$, $\overline{\text{IRQ7}}$, $\overline{\text{INTx}}$) guarantees their recognition at the next falling edge of the clock.
2. $\overline{\text{BR}}$ need fall at this time only to ensure being recognized at the end of the bus cycle.

Figure 11-3. Read Cycle Timing Diagram



NOTES:
1. Setup time (#47) for asynchronous inputs ($\overline{\text{HALT}}$, $\overline{\text{RESET}}$, $\overline{\text{BR}}$, $\overline{\text{BGACK}}$, $\overline{\text{DTACK}}$, $\overline{\text{IRQ7}}$, $\overline{\text{INTx}}$) guarantees their recognition at the next falling edge of the clock.
2. Because of loading variations, R/$\overline{\text{W}}$ may be valid after $\overline{\text{AS}}$ even though both are initiated by the rising edge of S2 (specification #20A).
3. $\overline{\text{BR}}$ need fall at this time only to ensure being recognized at the end of the bus cycle.

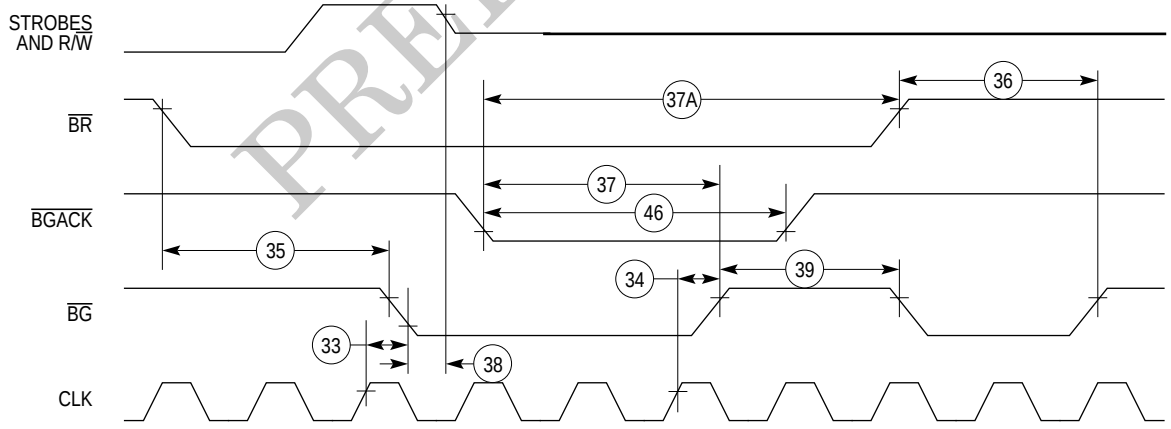
Figure 11-4. Write Cycle Timing Diagram

11.8 AC ELECTRICAL SPECIFICATIONS—BUS ARBITRATION

(See Figure 11-5 and Figure 11-6.)

Num	Characteristic	3.3V 8.33 MHz Min	3.3V 8.33 MHz Max	3.3 V or 5 V 16.67 MHz Min	3.3 V or 5 V 16.67 MHz Max	Unit
7	Clock high to address, data bus high impedance (maximum)	—	100	—	50	ns
16	Clock high to control bus high impedence	—	100	—	50	ns
33	Clock high to $\overline{BG}$ asserted	0	40	0	20	ns
34	Clock high to $\overline{BG}$ negated	0	40	0	20	ns
35	$\overline{BR}$ asserted to $\overline{BG}$ asserted (for last cycle of operand transfer)	1.5	3.5	1.5	3.5	Clks
36	$\overline{BR}$ negated to $\overline{BG}$ negated	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ asserted to $\overline{BG}$ asserted	1.5	3.5	1.5	3.5	Clks
37A[1]	$\overline{BGACK}$ asserted to $\overline{BR}$ negated	40 ns	1.5	20 ns	1.5	Clks
38	$\overline{BG}$ asserted to control, address, data bus high impedance ($\overline{AS}$, $\overline{CSx}$ negated)	—	100	—	50	ns
39	$\overline{BG}$ width negated	1.5	—	1.5	—	Clks
47[2]	Asynchronous input setup time	10	—	5	—	Clks
57	$\overline{BGACK}$ negated to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$, R/$\overline{W}$ driven	1.5	—	1.5	—	Clks
58	$\overline{BR}$ negated to $\overline{AS}$, $\overline{CSx}$, $\overline{LDS}$, $\overline{UDS}$, R/$\overline{W}$ driven	1.5	—	1.5	—	Clks

NOTES:
1. Actual value depends on clock period.
2. If the asynchronous input setup time (#47) requirement is satisfied for $\overline{DTACK}$, the $\overline{DTACK}$ asserted to data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle.



NOTE: Setup time to the clock (#47) for the asynchronous inputs $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{HALT}$, $\overline{RESET}$, $\overline{IRQ7}$ and $\overline{INTx}$ guarantees their recognition at the next falling edge of the clock.

Figure 11-5. Three-Wire Bus Arbitration Diagram

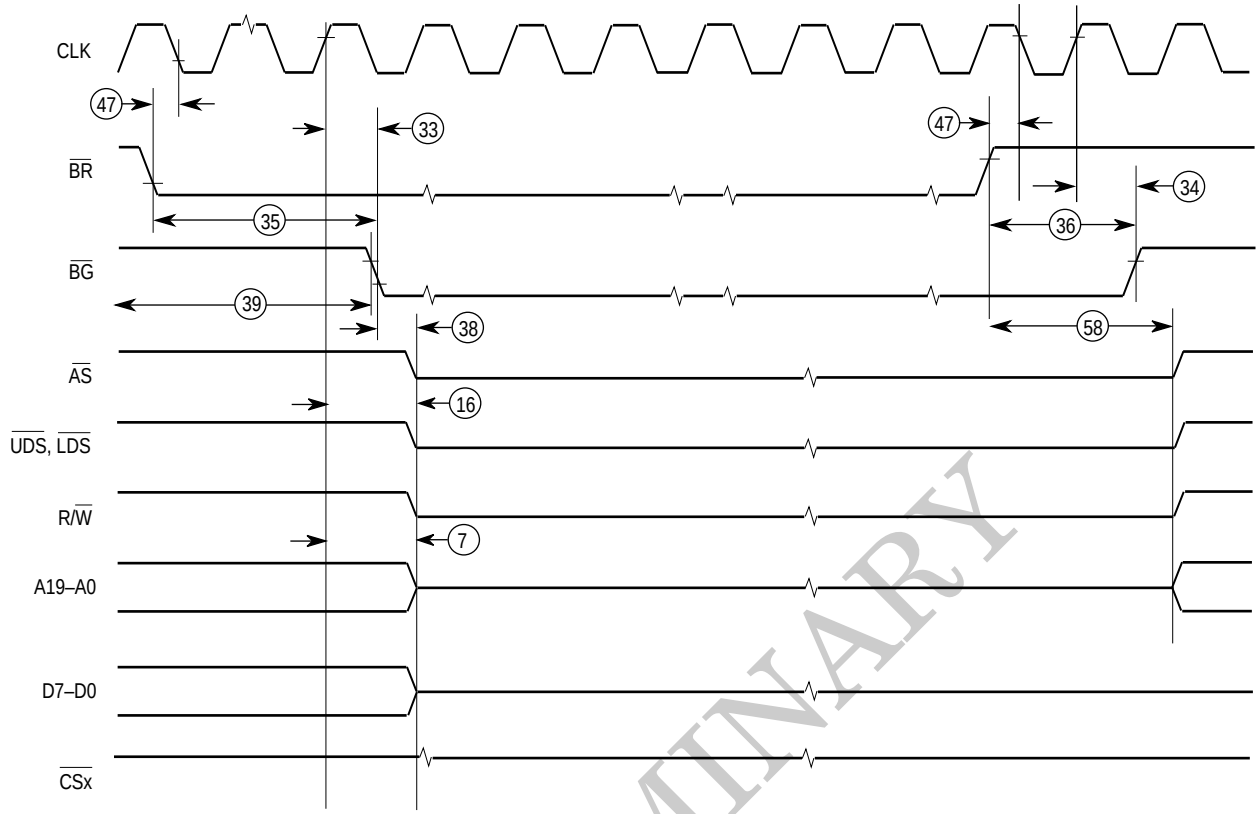


Figure 11-6. Two-Wire Bus Arbitration Timing Diagram

11.9 AC ELECTRICAL SPECIFICATIONS—8051 BUS INTERFACE MODULE (V_{CC} = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; $T_A = T_L$ to T_H) (See Figure 11-7 and Figure 11-8.)

Symbol	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
t_{cyc}	Cycle time	120	—	60	—	ns
TLHLL	ALE pulse width high	t_{cyc} – 30	—	t_{cyc} – 15	—	ns
TAVLL	Address valid to ALE low	1.5 x t_{cyc} – 30	—	1.5 x t_{cyc} – 15	—	ns
TLLAX	Address hold after ALE low	t_{cyc} – 60	—	t_{cyc} – 30	—	ns
TRLRH	$\overline{RD}$ pulse width asserted[1]	4.5 x t_{cyc} – 30	—	4.5 x t_{cyc} - 15	—	ns
TWLWH	$\overline{WR}$ pulse width asserted[1]	4.5 x t_{cyc} – 30	—	4.5 x t_{cyc} - 15	—	ns
TRLDV	$\overline{RD}$ asserted to valid data in[1]	—	4.5 x t_{cyc} – 70	—	4.5 x t_{cyc} – 35	ns
TRHDX	Data hold after $\overline{RD}$ negated	0	—	0	—	ns
TRHDZ	Data float after $\overline{RD}$ negated	—	0.5 x t_{cyc}	—	0.5 x t_{cyc}	ns
TLLDV	ALE low to valid data in[1]	—	7.5 x t_{cyc} – 70	—	7.5 x t_{cyc} – 35	ns
TAVDV	Address to valid data in[1]	—	9 x t_{cyc} – 70	—	9 x t_{cyc} – 35	ns
TLLWL	ALE low to $\overline{RD}$ or $\overline{WR}$ asserted	3 x t_{cyc} – 57	3 x t_{cyc} + 57	3 x t_{cyc} – 27	3 x t_{cyc} + 27	ns
TAVWL	Address valid to $\overline{RD}$ or $\overline{WR}$ asserted	4.5 x t_{cyc} – 30	—	4.5 x t_{cyc} – 15	—	ns
TQVWX	Data valid to $\overline{WR}$ asserted	t_{cyc} – 60	—	t_{cyc} – 30	—	ns
TQVWH	Data valid to $\overline{WR}$ negated[1]	5.5 x t_{cyc} – 60	—	5.5 x t_{cyc} – 30	—	ns
TWHQX	Data hold after $\overline{WR}$ negated	t_{cyc} – 90	—	t_{cyc} – 45	—	ns
TRLAZ	$\overline{RD}$ asserted to address float	—	0	—	0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ negated to ALE high	1.5 x t_{cyc} – 30	1.5 x t_{cyc}+ 30	1.5 x t_{cyc} – 15	1.5 x t_{cyc} + 15	ns

NOTES:
1. Wait states can be added.
2. The values in the table are for the minimum 6 wait states of 8051 mode.

Refer to **Section 3 Bus Operation** for a description of the 8051 diagram relative to the underlying 68000 bus cycle.

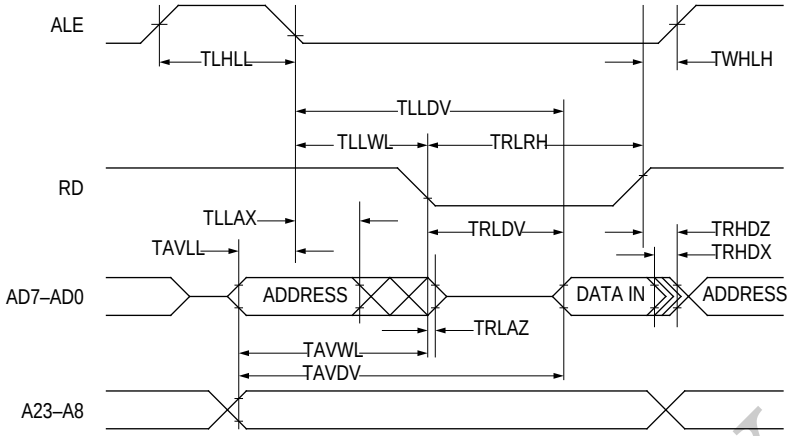


Figure 11-7. External 8051 Bus Read Cycle

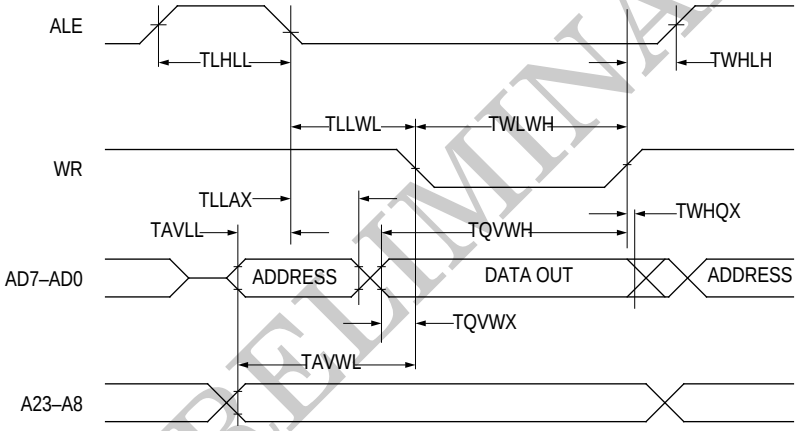


Figure 11-8. External 8051 Bus Write Cycle

11.10 TIMER MODULE ELECTRICAL CHARACTERISTICS

(V_{CC} = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; $T_A = T_L$ to T_H) (See Figure 11-9.)

Num	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
1	TIN clock low pulse width	1	—	1	—	Clk
2	TIN clock high pulse width and input capture high pulse width	2	—	2	—	Clk
3	TIN clock cycle time	3	—	3	—	Clk
4	Clock high to $\overline{\text{TOUT}}$ valid	—	70	—	35	ns

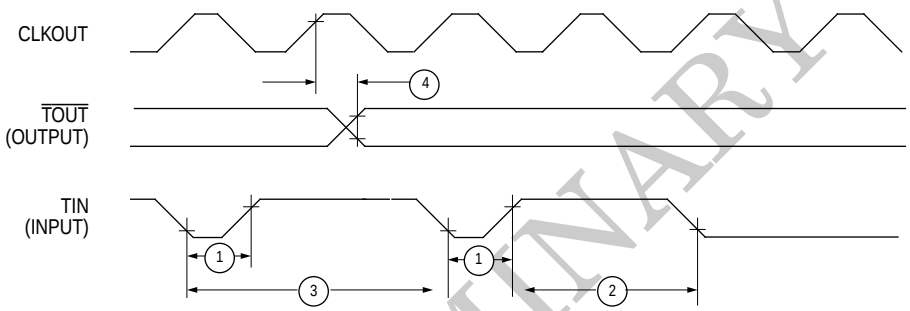


Figure 11-9. Timer Module Timing Diagram

11.11 UART ELECTRICAL CHARACTERISTICS

(VCC = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; TA = TL to TH)

(See Figure 11-10 and Figure 11-11.)

Num	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
1	TxD output valid from TxC low	—	200	—	100	ns
2	RxD data setup time to RxC high	480	—	240	—	ns
3	RxD data hold time from RxC high	400	—	200	—	ns

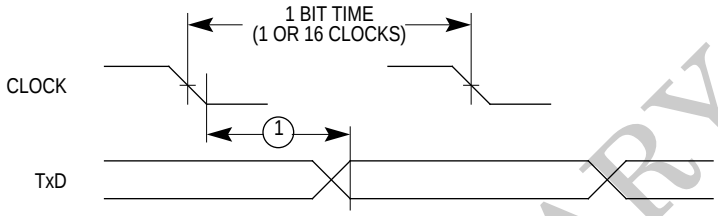


Figure 11-10. Transmitter Timing

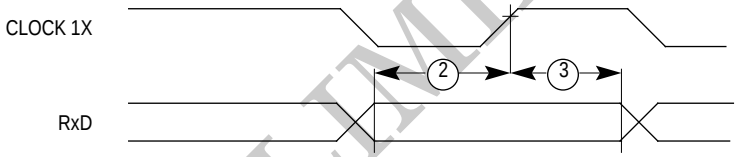


Figure 11-11. Receiver Timing

11.12 AC ELECTRICAL CHARACTERISTICS—M-BUS INPUT SIGNAL TIMING (V_{CC} = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; $T_A = T_L$ to T_H) (See Figure 11-12.)

Num	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
1	Start condition hold time	2	—	2	—	Clk
2	Clock low period	4.7	—	4.7	—	Clk
3	SDA/SCL rise time	—	2	—	1	µs
4	Data hold time	0	—	0	—	Clk
5	SDA/SCL fall time	—	600	—	300	ns
6	Clock high period	4	—	4	—	Clk
7	Data setup time	500	—	250	—	µs
8	Start condition setup time (for repeated start condition only)	2	—	2	—	Clk
9	Stop condition setup time	2	—	2	—	Clk

11.13 AC ELECTRICAL CHARACTERISTICS—M-BUS OUTPUT SIGNAL TIMING (V_{CC} = 5.0V ± 0.5V or 3.3Vdc ± 0.3V; GND = 0Vdc; $T_A = T_L$ to T_H) (See Figure 11-12.)

Num	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
1	Start condition hold time	8	—	8	—	Clk
2	Clock low period	11	—	11	—	Clk
3	SDA/SCL rise time	—	2	—	1	µs
4	Data hold time	0	2	0	2	Clk
5	SDA/SCL fall time	—	600	—	300	ns
6	Clock high period	11	—	11	—	Clk
7	Data setup time	Spec 2 x Clk	—	Spec 2 x Clk	—	ns
8	Start condition setup time (for repeated start condition only)	20	—	10	—	Clk
9	Stop condition setup time	20	—	10	—	Clk

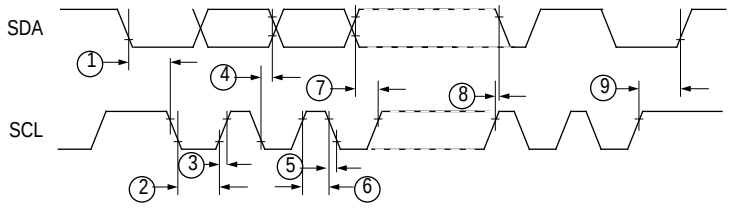


Figure 11-12. M-Bus Interface Input/Output Signal Timing

11.14 AC ELECTRICAL CHARACTERISTICS—PORT TIMING

(See Figure 11-13.)

Characteristic	Symbol	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
Port Input Setup Time to $\overline{UDS}$, $\overline{LDS}$ Asserted	t_{PS}	0	—	0	—	ns
Port Input Hold Time from $\overline{UDS}$, $\overline{LDS}$ Negated	t_{PH}	0	—	0	—	ns
Port Output Valid from $\overline{UDS}$, $\overline{LDS}$ Negated	t_{PD}	—	30	—	60	ns

NOTE: Test conditions for port outputs: C_L = 50 pF, R_L = 27 kΩ to V_{CC}.

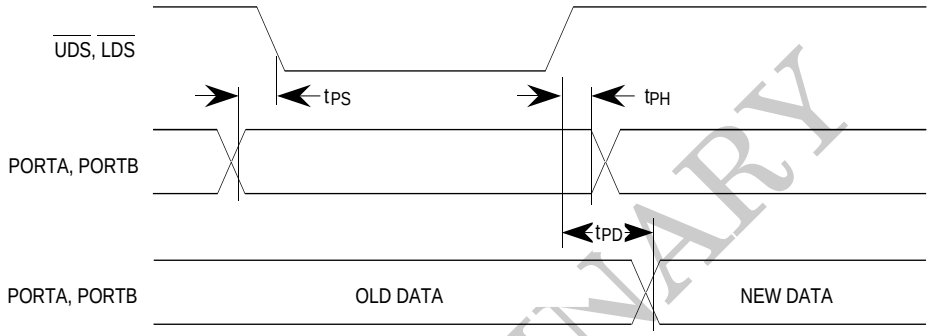


Figure 11-13. Port Timing

11.15 IEEE 1149.1 ELECTRICAL CHARACTERISTICS

(See Figure 11-14, Figure 11-15, and Figure 11-16.)

Num.	Characteristic	3.3V		3.3 V or 5 V		Unit
		8.33 MHz		16.67 MHz		
		Min	Max	Min	Max	
	TCK Frequency of Operation	0	10.0	0	5.0	MHz
1	TCK Cycle Time	100	—	200	—	ns
2	TCK Clock Pulse Width Measured at 1.5 V	45	—	45	—	ns
3	TCK Rise and Fall Times	0	5	0	10	ns
6	Boundary Scan Input Data Setup Time	15	—	30	—	ns
7	Boundary Scan Input Data Hold Time	15	—	30	—	ns
8	TCK Low to Output Data Valid	0	80	0	160	ns
9	TCK Low to Output High Impedance	0	80	0	160	ns
10	TMS, TDI Data Setup Time	15	—	30	—	ns
11	TMS, TDI Data Hold Time	15	—	30	—	ns
12	TCK Low to TDO Data Valid	0	30	0	60	ns
13	TCK Low to TDO High Impedance	0	30	0	60	ns

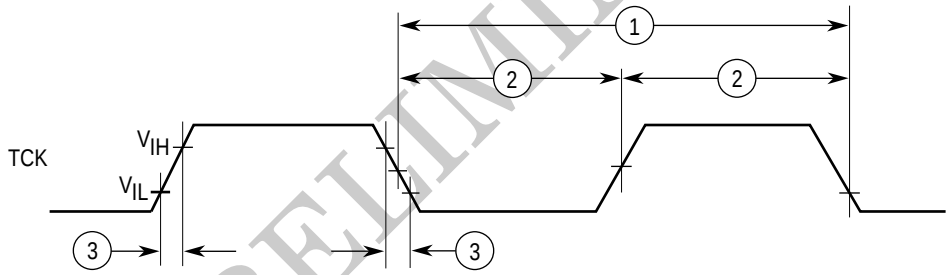


Figure 11-14. Test Clock Input Timing Diagram

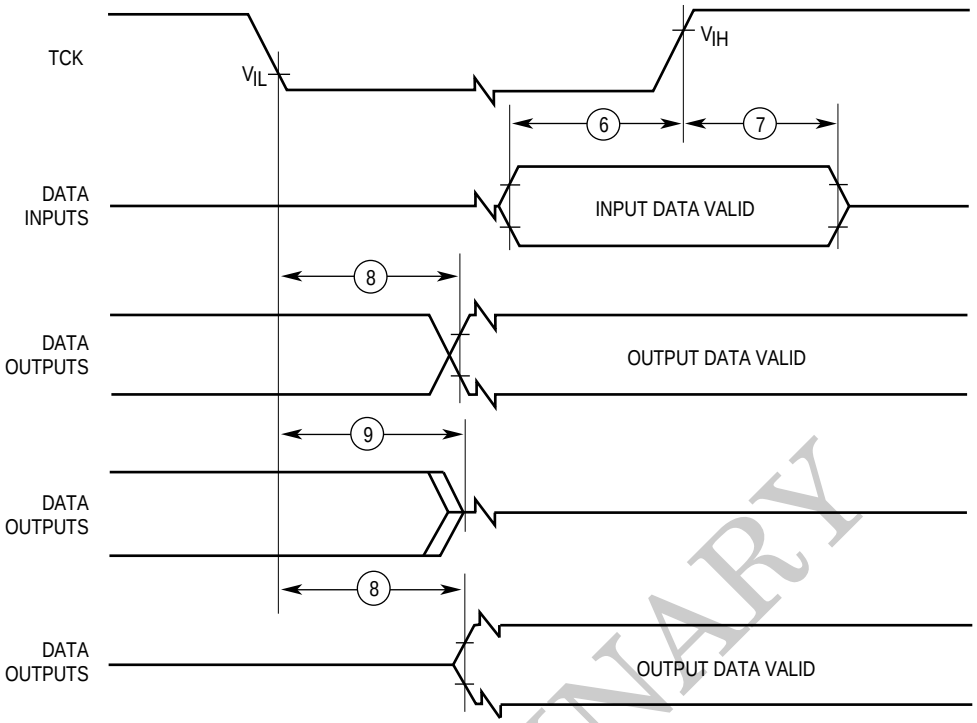


Figure 11-15. Boundary Scan Timing Diagram

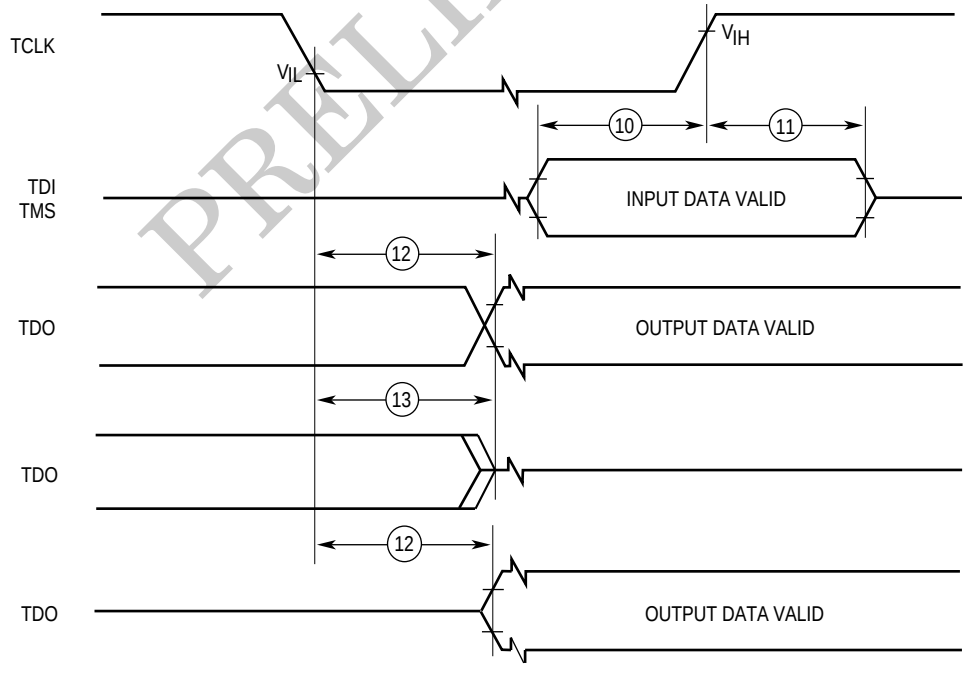


Figure 11-16. Test Access Port Timing Diagram

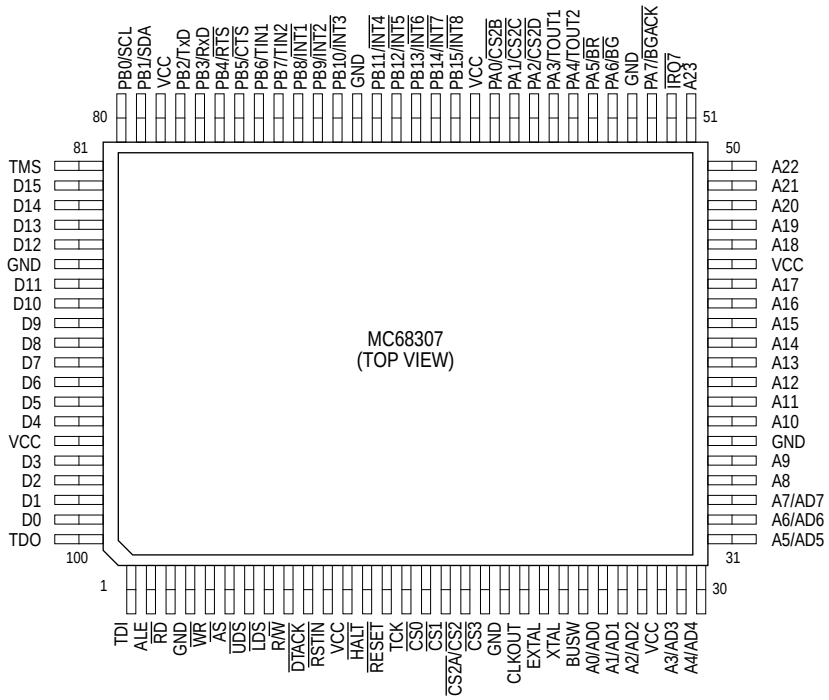
SECTION 12
ORDERING INFORMATION AND MECHANICAL DATA

This section contains the ordering information, pin assignments, and package dimensions for the MC68307.

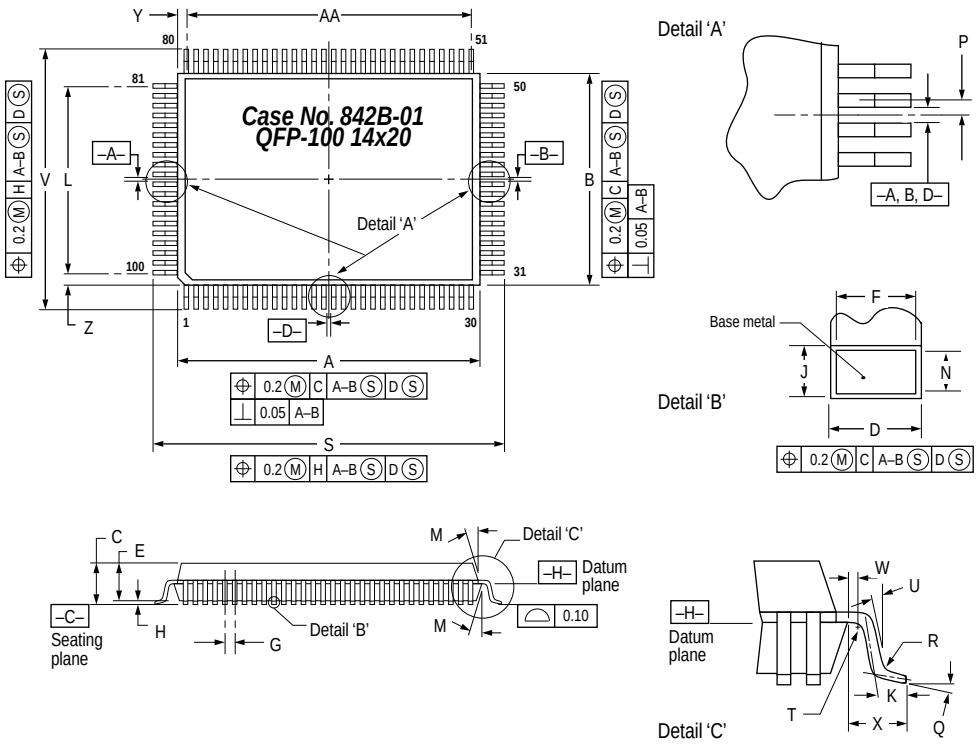
12.1 STANDARD ORDERING INFORMATION

Package Type	Frequency (MHz)	Supply Voltage (V)	Temperature	Order Number
100-Pin Plastic Quad Flat Pack (FG Suffix)	16.67	5	0°C to 70°C	MC68307FG16
100-Pin Thin Quad Flat Pack (PU Suffix)	16.67	5	–40°C to 70°C	MC68307PU16
100-Pin Plastic Quad Flat Pack (FG Suffix)	16.67	5	–40°C to 85°C	MC68307CFG16
100-Pin Plastic Quad Flat Pack (FG Suffix)	16.67	3.3	–40°C to 70°C	MC68307FG16V
100-Pin Thin Quad Flat Pack (PU Suffix)	8.33	3.3	–40°C to 70°C	MC68307PU8V
100-Pin Thin Quad Flat Pack (PU Suffix)	16.67	3.3	–40°C to 70°C	MC68307PU16V

12.2 100-PIN PQFP PIN ASSIGNMENTS (FG SUFFIX)

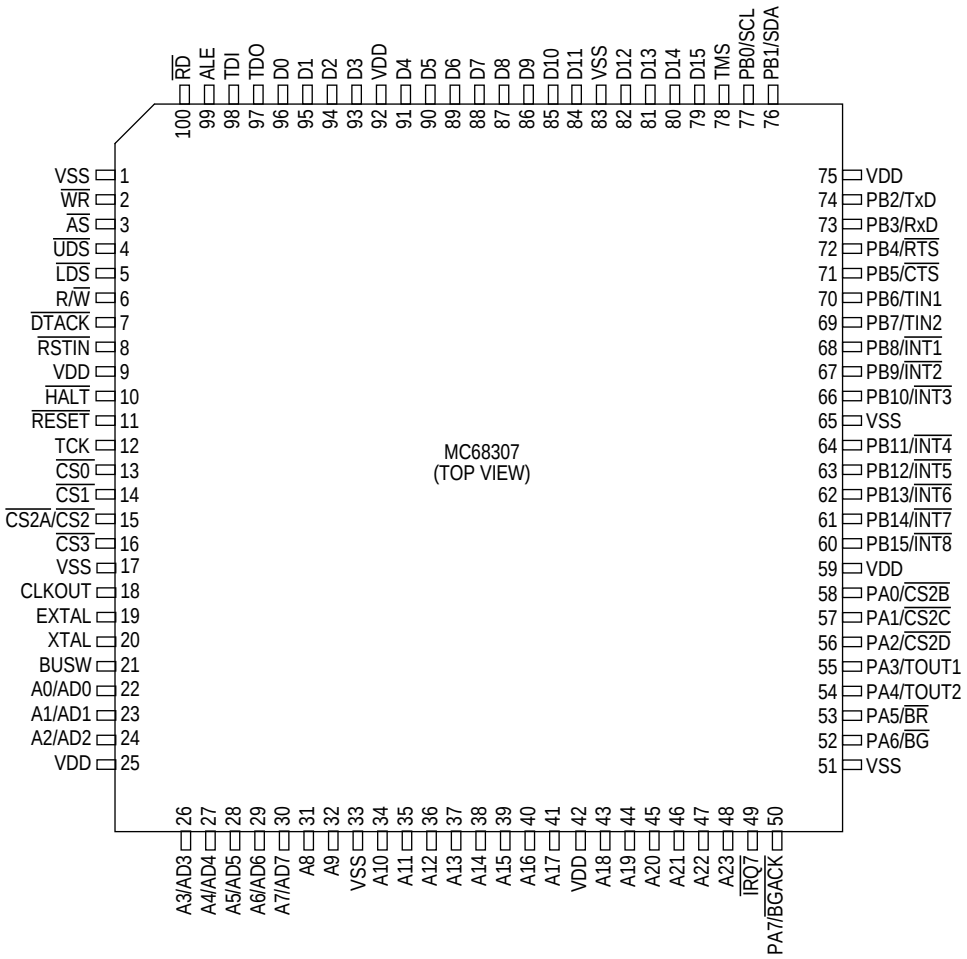


12.3 100-PIN PQFP PACKAGE DIMENSIONS (FG SUFFIX)

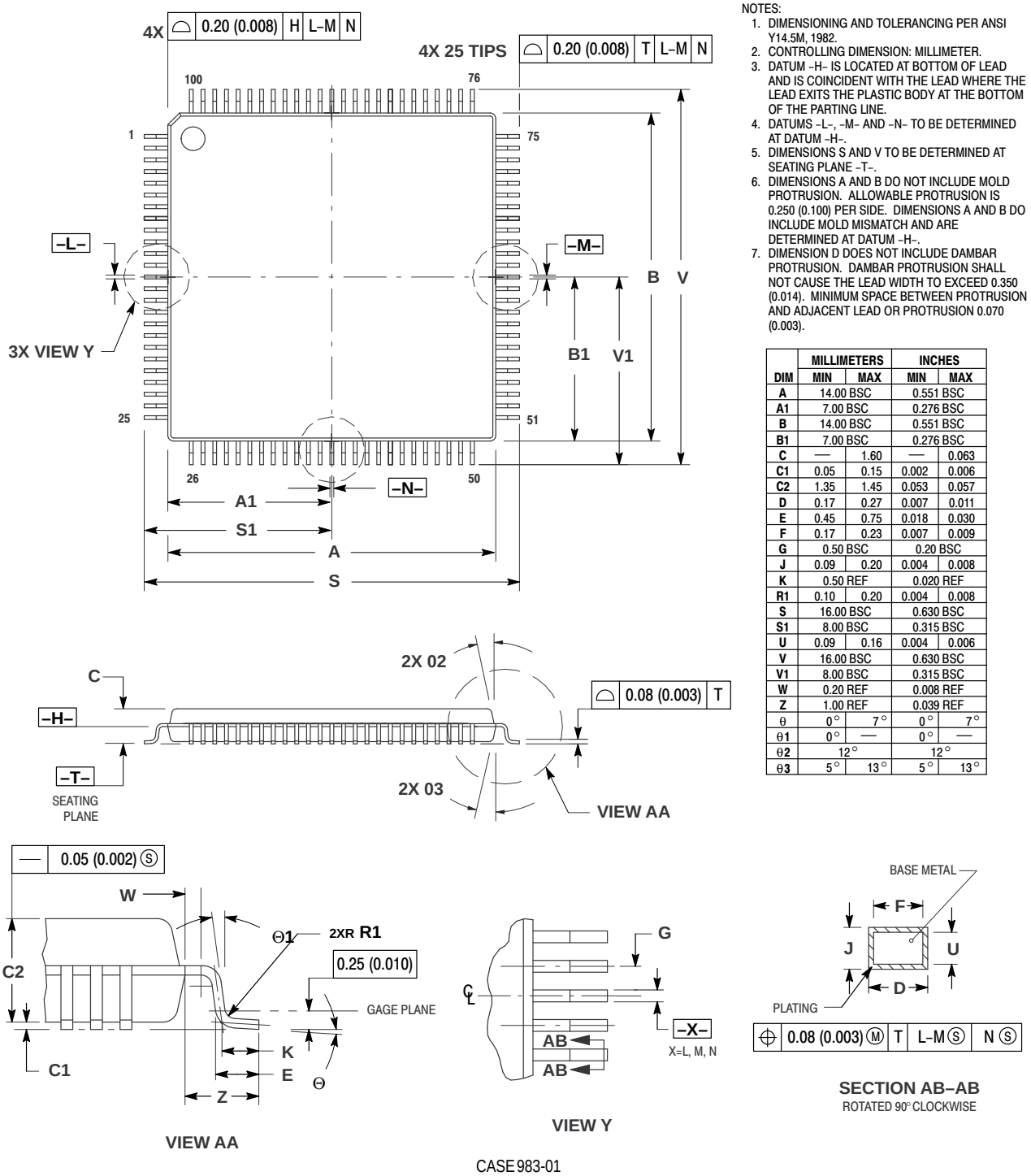


Dim.	Min.	Max.	Notes	Dim.	Min.	Max.
A	19.90	20.10		P	0.325 BSC	
B	13.90	14.10	1. Dimensioning and tolerancing per ANSI Y14.5M, 1982.	Q	0°	7°
C	2.85	3.15	2. All dimensions in mm.	R	0.13	0.30
D	0.22	0.38	3. Datum plane –H– is located at the bottom of the lead and is coincident with the lead where it exits the plastic body at the bottom of the parting line.	S	23.10	23.37
E	2.55	3.05		T	0.13	—
F	0.22	0.33	4. Datums A–B and –D– to be determined at datum plane –H–.	U	0°	—
G	0.65 BSC		5. Dimensions S and V to be determined at seating plane –C–.	V	17.10	17.37
H	0.25	0.35	6. Dimensions A and B do not include mould protrusion; allowable protrusion is 0.25mm per side. Dimensions A and B do include mould mismatch and are determined at datum plane –H–.	W	0.40	—
J	0.13	0.23		X	1.60 REF	
K	0.75	0.92	7. Dimension D does not include dambar protrusion; allowable protrusion is 0.08mm total in excess of the D dimension at maximum material condition. Dambar cannot be located on the lower radius or the foot.	Y	0.58 REF	
L	12.35 REF			Z	0.83 REF	
M	5°	16°		AA	18.85 REF	
N	0.13	0.17		—		

12.4 100-PIN TQFP PIN ASSIGNMENTS (PU SUFFIX)



12.5 100-PIN TQFP PACKAGE DIMENSIONS (PU SUFFIX)



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM –H– IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS –L–, –M– AND –N– TO BE DETERMINED AT DATUM –H–.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE –T–.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.250 (0.100) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM –H–.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL NOT CAUSE THE LEAD WIDTH TO EXCEED 0.350 (0.014). MINIMUM SPACE BETWEEN PROTRUSION AND ADJACENT LEAD OR PROTRUSION 0.070 (0.003).

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	14.00 BSC		0.551 BSC	
A1	7.00 BSC		0.276 BSC	
B	14.00 BSC		0.551 BSC	
B1	7.00 BSC		0.276 BSC	
C	—	1.60	—	0.063
C1	0.05	0.15	0.002	0.006
C2	1.35	1.45	0.053	0.057
D	0.17	0.27	0.007	0.011
E	0.45	0.75	0.018	0.030
F	0.17	0.23	0.007	0.009
G	0.50 BSC		0.20 BSC	
J	0.09	0.20	0.004	0.008
K	0.50 REF		0.020 REF	
R1	0.10	0.20	0.004	0.008
S	16.00 BSC		0.630 BSC	
S1	8.00 BSC		0.315 BSC	
U	0.09	0.16	0.004	0.006
V	16.00 BSC		0.630 BSC	
V1	8.00 BSC		0.315 BSC	
W	0.20 REF		0.008 REF	
Z	1.00 REF		0.039 REF	
θ	0°	7°	0°	7°
θ1	0°	—	0°	—
θ2	12°		12°	
θ3	5°	13°	5°	13°

CASE 983-01

INDEX

Numerics

8051-compatible bus 2-9, 5-8
8051-compatible bus interface 5-10

A

A23–A0 2-5
access errors 4-1
AD7–AD0 2-6
address bus 2-5
address latch enable 2-9
address strobe 2-8
addressing modes 4-4
  index sizing and scaling 4-3
  indexing 4-3
  offset 4-3
  postincrement 4-3
  predecrement 4-3
  program counter indirect 4-3
  register indirect 4-3
ALE 2-9
$\overline{AS}$ 2-8, 3-4, 3-7, 3-8, 3-18
asynchronous bus arbitration signals 3-19
asynchronous mode 3-32

B

base registers 5-30
baud rate generator 8-5
$\overline{BERR}$ 3-4
$\overline{BG}$ 2-10, 3-18
$\overline{BGACK}$ 2-10
boundary scan 9-1
boundary scan bit definitions 9-4
$\overline{BR}$ 2-10
BR3-BR0 5-30
bus arbitration 3-15
  2-wire 3-15
  3-wire 3-15
bus control signals 2-7
bus error exception 3-31
bus grant 2-10
bus grant acknowledge 2-10
bus grant signal 3-16
bus request 2-10
bus width select 2-9
BUSW0 2-9
byte read cycle flowchart 3-2

C

chip select registers 5-30
chip selects 1-5, 2-6, 5-5
clear-to-send 2-12
CLKOUT 2-11
clock output 2-11
clock signals 2-10
crystal oscillator 2-10
$\overline{CS0}$ 2-6
$\overline{CS1}$ 2-6
$\overline{CS2}$ 2-6
$\overline{CS3}$ 2-7
$\overline{CTS}$ 2-12

D

data bus 2-6, 3-29
data formats 4-3
data strobes 2-8
data transfer acknowledge 2-7
data types
  denormalized numbers 4-3
  infinities 4-3
  NANs 4-3
  normalized numbers 4-3
  zeros 4-3
denormalized numbers 4-3
double bus fault 3-31
$\overline{DTACK}$ 2-7, 3-4
dual timer module 6-1

E

EBI 1-5
EPROM memory interface 10-5
EPROM timing 10-6
exceptions 4-9
  access errors 4-1
  address error exception 4-18
  bus error exception 4-18
  control signals 2-9
  exception vector 4-12
  exceptions handler 4-10
  reset exception 4-14
  SR
    M-bit 4-10
  trace exception 4-17
  vector assignments 4-13
  vector table 4-9
EXTAL 2-10
external bus interface 1-5
  control registers 5-34
  logic 5-9

F

FIFO stack 8-8
floating-point unit
  (see data types)
function codes 3-29

G

general purpose timer units 6-4
general-purpose I/O ports 5-10
general-purpose interrupt inputs 5-17
general-purpose timer units 6-1
global chip select operation 5-8

H

$\overline{\text{HALT}}$ 2-10
halt 2-10

I

I/O driver routines 8-31
index sizing and scaling 4-3
indexing 4-3
infinities 4-3
initialization routines 8-31
instructions
  STOP 4-1
  TRAP, TRAPV, CHK, RTE, and DIV 4-9
$\overline{\text{INT1}}$ - $\overline{\text{INT8}}$ 2-13
interrupts 4-1
  acknowledge bus cycle 4-9
  control registers 5-38
  controller logic 5-14
  general-purpose interrupt inputs 5-17
  handling routine 8-31
  latched interrupt control registers 5-38
  level 7 interrupts 4-14
  non-maskable interrupt 5-17
  peripheral interrupt control register 5-39
  peripheral interrupt handling 5-18
  priorities 4-14
  priority mask 4-9
  processing 5-13
  requests 4-15
    inputs 2-13
    signals 8-3
  uninitialized interrupt vector 4-15
  vector generation 5-15
$\overline{\text{IRQ7}}$ 2-13, 5-17

J

JTAG 9-1
  instruction register 9-9

L

latched interrupt control registers 5-38
$\overline{\text{LDS}}$ 2-8, 3-7
LICR1, LICR2 5-38
looping modes 8-10
low-power sleep mode 10-9
low-power stop logic 5-19
low-power stop mode 10-9

M

M68000 bus interface 5-9
MBAR 5-22
M-Bus
  address register 7-6
  arbitration lost 7-13
  arbitration procedure 7-4
  clock stretching 7-5

clock synchronization 7-4
control register 7-7
data I/O register 7-10
data transfer 7-3
frequency divider register 7-6
generation of repeated START 7-13
generation of START 7-11
generation of STOP 7-12
handshaking 7-5
I/O signals 2-11
initialization sequence 7-10
interface module 7-1
interrupt routine 7-14
master mode 10-12
module 1-6
post-transfer software 7-11
programming examples 7-10
programming model 7-5
protocol 7-2
registers
MADR 7-6
MBCR 7-7
MBDR 7-10
MBSR 7-9
MFDR 7-6
repeated START signal 7-4
slave address transmission 7-3
slave mode 7-13, 10-12
software 10-10
START signal 7-3
status register 7-9
STOP signal 7-4
system configuration 7-2
module base address register 5-2, 5-22

N

NANs 4-3
non-IEEE 1149.1 operation 9-11
non-maskable interrupt 5-17
non-maskable interrupt input 2-13
normalized numbers 4-3

O

offset 4-3
operand size 4-3
option registers 5-32
OR3-OR0 5-32
overlap in chip-select ranges 5-8

P

package dimensions
PQFP 12-2
TQFP 12-4
PACNT 5-34
PADAT 5-35
PADDR 5-35
parallel I/O ports 1-5
PB7 2-13
PBCNT 5-36
PBDAT 5-37
PBDDR 5-36
peripheral chip-selects 5-7
peripheral interrupt control register 5-39
peripheral interrupt handling 5-18
PICR 5-39
pin assignments
PQFP 12-1
TQFP 12-3
Port A 5-10, 5-12
control register 5-34
data direction register 5-35
data register 5-35
Port B 2-11–2-13, 5-10, 5-12
control register 5-36
data direction register 5-36
data register 5-37
postincrement 4-3
power-on reset 2-10
predecrement 4-3
privileged instructions 4-17
processing states
normal, exception, halted 4-1
program counter indirect 4-3
programmable data-bus size 5-6

R

R/$\overline{W}$ 2-8, 3-4, 3-7
RAM memory interface 10-5
RAM timing 10-6
$\overline{RD}$ 2-9
read cycle 3-2
read/write 2-8

read-modify-write cycle 3-8, 3-29
receive data 2-12
register indirect 4-3
request-to-send 2-12
RESET 2-9
reset 2-9
RS232 UART port 10-5
RSTIN 2-10
RTS 2-12
RxD 2-12

S

SCL 2-11
SCR 5-23
SDA 2-12
serial clock 2-11
serial data 2-12
serial module registers
  UACR 8-26
  UBG1 8-29
  UBG2 8-29
  UCR 8-23
  UCSR 8-21
  UIMR 8-28
  UIP 8-29
  UIPCR 8-26
  UISR 8-27
  UIVR 8-29
  URB 8-25
  USR 8-19
  UTB 8-25
signal configuration 10-1
signal index 2-14
SIM configuration 5-41
SIM programming model 5-20
SIM07 1-4
single-step 3-30
software listings 10-17–10-29
software watchdog timer 6-3, 6-7
stack frame 4-9
stand alone hardware 10-1
startup - cold reset 5-41
status register 4-9
swapping ROM and RAM mapping 10-27
system configuration and protection 1-5
  registers 5-22
system control register 5-4, 5-23
system integration module 1-4, 5-1
system protection functions 5-5

T

TAP 9-1
TAS 3-8
TCK 2-11
TCN1, TCN2 6-5
TCR1, TCR2 6-5
TDI 2-11
TDO 2-11
TER1, TER2 6-6
test access port 1-7
test clock 2-11
test data in 2-11
test data out 2-11
test mode select 2-11
test signals 2-11
timer 1-6
  capture registers 6-5
  counter 6-5
  event registers 6-6
  I/O signals 2-12
  mode register 6-4
  reference registers 6-5
  registers
    TCN1, TCN2 6-5
    TCR1, TCR2 6-5
    TER1, TER2 6-6
    TMR1, TMR2 6-4
    TRR1, TRR2 6-5
  timer 1 input 2-12
  timer 1 output 2-13
  timer 2 input 2-13
  timer 2 output 2-13
timer/counter 8-3
TIN1 2-12
TIN2 2-13
TMR1, TMR2 6-4
TMS 2-11
TOUT1 2-13
TOUT2 2-13
transmit data 2-12
TRR1, TRR2 6-5
TxD 2-12

U

UACR 8-26
UART driver examples 10-24
UART I/O signals 2-12
UART module 1-6
UBG1 8-29
UBG2 8-29
UCR 8-23
UCSR 8-21
$\overline{\text{UDS}}$ 2-8, 3-7
$\overline{\text{UDS}}$/$\overline{\text{LDS}}$ 3-4, 3-10
UIMR 8-28
UIP 8-29
UIPCR 8-26
UISR 8-27
UIVR 8-29
URB 8-25
USR 8-19
UTB 8-25

V

valid start bit 8-8
vector number 4-9

W

wait state generation 1-5
wait-state logic 5-5
wake-up 10-8
watchdog counter register 6-7
watchdog reference register 6-7
WCR 6-7
word read cycle flowchart 3-2
$\overline{\text{WR}}$ 2-9
write cycle 3-5
WRR 6-7

X

XTAL 2-10

Z

zeros 4-3