1. INTRODUCTION

This product is an ASSP (Application Specific Standard product) with a 68HC000 MPU core and built-in serial I/F, parallel I/F, timer, interrupt controller, address decoder, DMA controller, DRAM controller and stepping motor controller as peripheral devices.

Features

Core : 68HC000
System clock : 16MHz max. (target)

- 17 32-bit register
- 16 M-byte direct addressing
- 56 powerful basic instructions
- 14 addressing modes
- Serial I/F : 2-channel (UART)
- Parallel I/F : 10-bit I/O
- Timer : 5-channel (include built-in watchdog timer)
- Interrupt controller : 12-channel (external : 3 channels , internal : 9channels)
- Address decoder : built-in
- DMA controller : 3-channel
- DRAM controller : $\overline{RAS}$, $\overline{CAS}$, multiplexed address, A1~A9 (A10)
- Stepping motor controller : 2-channel, 4-phase
- Chip Select Signal : $\overline{CS0}$, $\overline{CS1}$
- Automatic wait state insertion
- Bus error detection
- Low power consumption (CMOS)

Figure 1.1 shows TMP68303 configuration.

TMP68303 has two operation modes. One is the normal operation mode and the other is the emulation mode for using 68000 development tools (ICE). In the emulation mode, the 68HC000 core does not operate and the built-in peripheral devices are operated by external address/data control signal.

The 68HC000 core is the same as the TMP68HC000, except that the 8-bit peripheral devices control signals E, $\overline{VPA}$ and $\overline{VMA}$ are not used. Those already familiar with the TMP68HC000 should read the following items related to peripheral devices.

As shown in the programming models (Figures 1.2 and 1.3), the TMP68303 has seventeen 32-bit registers, a 32-bit programming counter and a 16-bit status register (the lower bits are the condition code register).

The first 8 registers (D0-D7) shown in Figure 1.2 are data registers and can be used as byte (8-bit), word (16-bit) and long word(32-bit) operands. The next 7 registers(A0-A6) and user stack pointer (USP) can be used as the software stack pointer and base address register. In addition, these registers can be used as word and long word operands. All 16 registers are used as index registers.

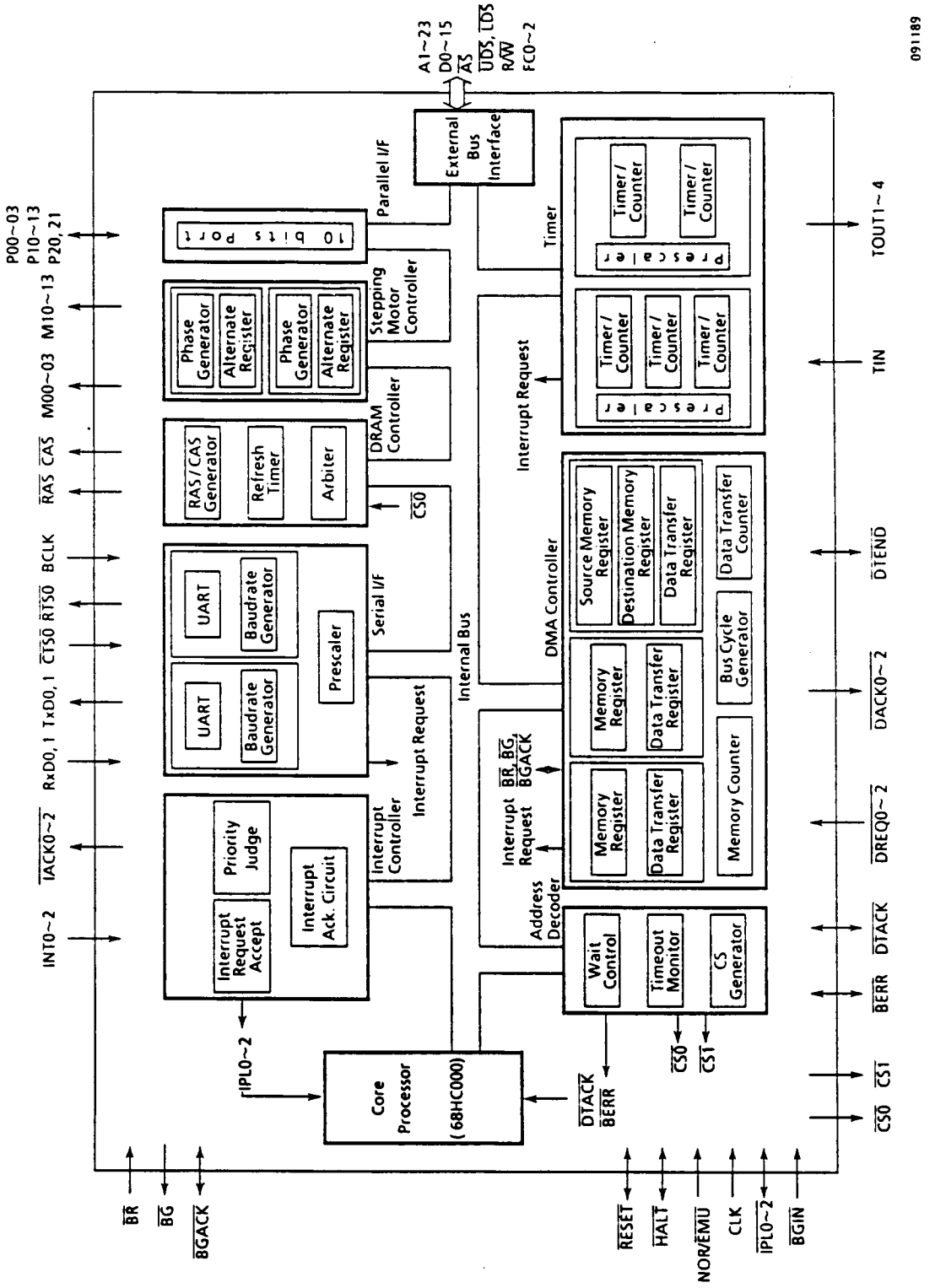


Figure 1.1 TMP68303 Block Diagram

In the supervisor mode, the upper bytes of status register and the supervisor stack pointer (SSP) can be freely used (Figure 1.3).

The status register (Figure 1.4) contains the extend (X) flag, negative sign (N) flag, zero (Z) flag, overflow (V) flag and carry (C) flag as well as the mask bit (8 possible levels). Trace mode (T) and supervisor state (S) bits are also provided as status bits.

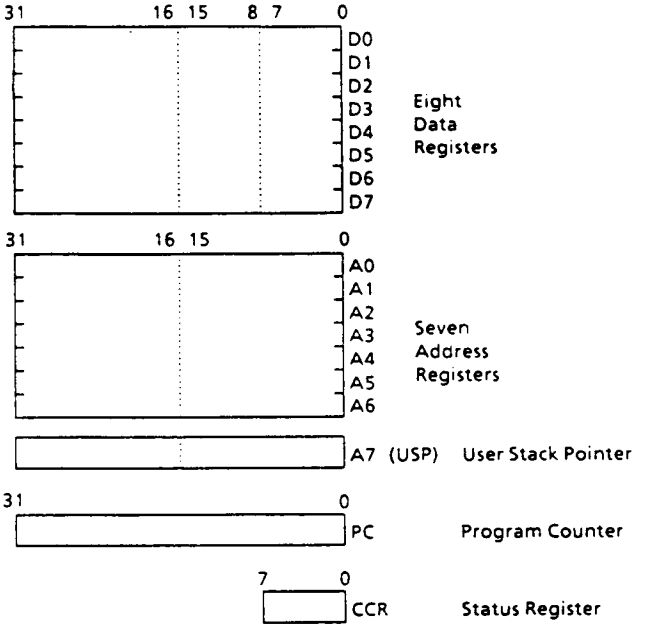


Figure 1.2 User Programming Model

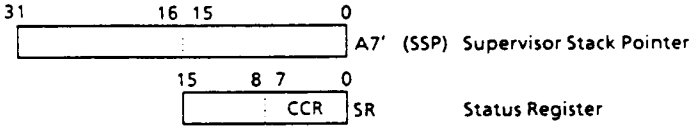


Figure 1.3 Supervisor Programming Model Supplement

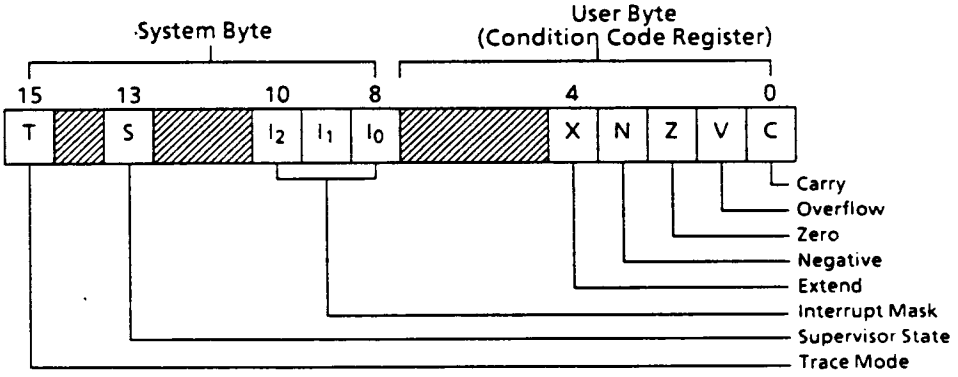


Figure 1.4 Status Register

1.1 DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4 bits)
- Bytes (8 bits)
- Words (16 bits)
- Long Words (32 bits)

In addition, operations on other data types such as memory addresses, status word data, etc., are provided in the instruction set.

The 14 address modes, shown in Table 1.1, include six basic types:

- Register Direct
- Register Indirect
- Absolute
- Program Counter Relative
- Immediate
- Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting, and indexing. The program counter relative mode can also be modified via indexing and offsetting.

Table 1.1 Addressing Modes

Addressing Modes	Syntax
Register Direct Addressing	
Data Register Direct	Dn
Address Register Direct	An
Absolute Data Addressing	
Absolute Short	Abs.W
Absolute Long	Abs.L
Program Counter Relative Addressing	
Relative with Offset	d16 (PC)
Relative with Index Offset	d8 (PC, Xn)
Register Indirect Addressing	
Register Indirect	(An)
Postincrement Register Indirect	(An) +
Predecrement Register Indirect	- (An)
Register Indirect with Offset	d16 (An)
Indexed Register Indirect with Offset	d8 (An, Xn)
Immediate Data Addressing	
Immediate	#xxx
Quick Immediate	#1~#8
Implied Addressing	
Implied Register	SR / USP / SSP / PC

Notes: Dn = Data Register
An = Address Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
SP = Stack Pointer
USP = User Stack Pointer
( ) = Effective Address
d8 = 8-Bit Offset (Displacement)
d16 = 16-Bit Offset (Displacement)
#xxx = Immediate Data

1.2 INSTRUCTION SET OVERVIEW

The TMP68303 instruction set is shown in Table 1.2. Some additional instructions are variations, or subsets, of these and they appear in Table 1.3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned, multiply and divide, "quick"arithmetic operations, BCD arithmetic, and expanded operations (through traps).

Table 1.2 Instruction Set Summary (1/2)

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
Bcc	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Compare
DBcc	Test Condition, Decrement and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right

Table 1.2 Instruction Set Summary (2/2)

Mnemonic	Description
MOVE	Move
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NOP	No Operation
NOT	One's Complement
OR	Logical OR
PEA	Push Effective Address
RESET	Reset External Devices
ROL	Rotate Left without Extend
ROR	Rotate Right without Extend
ROXL	Rotate Left with Extend
ROXR	Rotate Right with Extend
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditional
STOP	Stop
SUB	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink

Table 1.3 Variations of Instruction Types

Instruction Type	Variation	Description
ADD	ADD	Add
	ADDA	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX	Add with Extend
AND	AND	Logical And
	ANDI	AND Immediate
	ANDI to CCR	AND Immediate to Condition Codes
	ANDI to SR	AND Immediate to Status Register
CMP	CMP	Compare
	CMPA	Compare Address
	CMPM	Compare Memory
	CMPI	Compare Immediate
EOR	EOR	Exclusive OR
	EORI	Exclusive OR Immediate
	EORI to CCR	Exclusive OR Immediate to Condition Codes
	EORI to SR	Exclusive OR Immediate to Status Register
MOVE	MOVE	Move
	MOVEA	Move Address
	MOVEQ	Move Quick
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE to CCR	Move to Condition Codes
	MOVE USP	Move User Stack Pointer
NEG	NEG	Negate
	NEGX	Negate with Extend
OR	OR	Logical OR
	ORI	OR Immediate
	ORI to CCR	OR Immediate to Condition Codes
	ORI to SR	OR Immediate to Status Register
SUB	SUB	Subtract
	SUBA	Subtract Address
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX	Subtract with Extend

2. DATA ORGANIZATION AND ADDRESSING CAPABILITIES

This section contains a description of the registers and the data organization of the TMP68303.

2.1 OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implicit instructions support some subset of all three sizes.

2.2 DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the stack pointers support address operadnds of 32 bits.

2.2.1 Data Registers

Each data register is 32 bits wide. Byte operadns occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

When a data register is used as either a source or destination operand, only the appropriate low order portion is changed; the remaining high order portion is neither used nor changed.

2.2.2 Address Registers

Each address register and the stack pointer is 32 bits wide and holds a full 32-bit address. Address registers do not support the sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

2.3 DATA ORGANIZATION IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 2.1. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even) , then the second word of that datum is located at address n + 2.

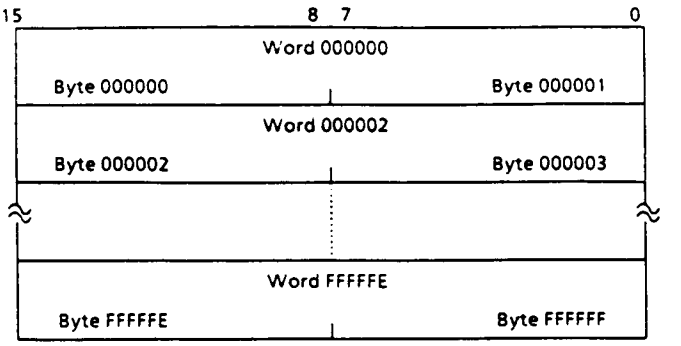


Figure 2.1 Word Organization in Memory

The data types supported by the TMP68303 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 2.2. The numbers indicate the order in which the data accessed from the core processor.

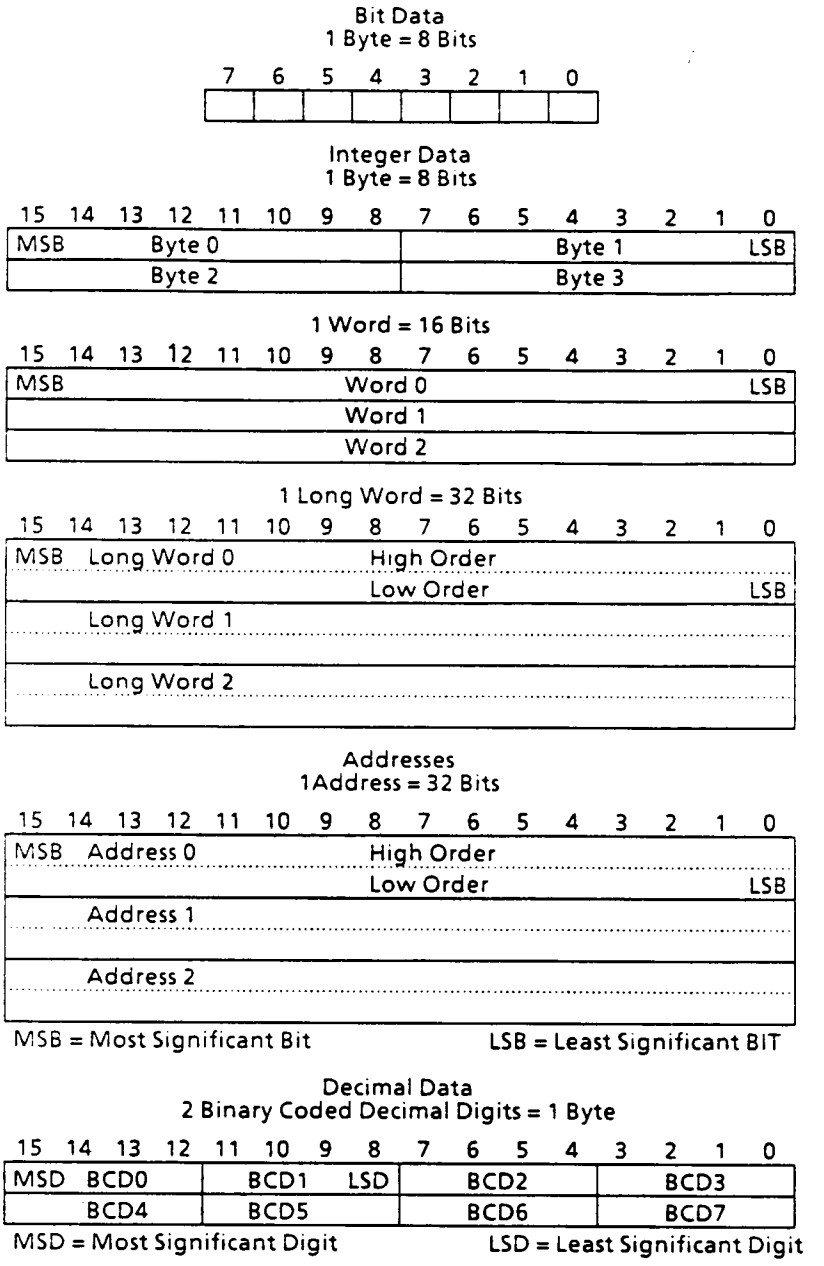


Figure 2.2 Memory Data Organization

2.4 ADDRESSING

Instructions for the TMP68303 contain two kinds of information: the type of function to be performed and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification	—	the number of the register is given in the register field of their instruction.
Effective Address	—	use of the different effective addressing modes.
Implicit Reference	—	the definition of certain instructions implies the use of specific registers.

2.5 INSTRUCTION FORMAT

Instructions are from one to five words in length as shown in Figure 2.3. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

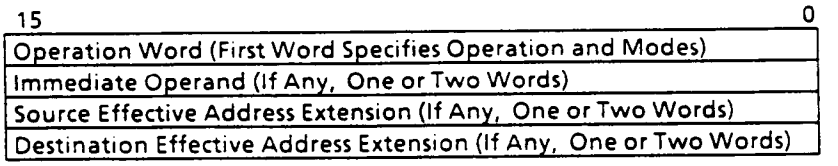


Figure 2.3 Instruction Operation Word General Format

2.6 PROGRAM/DATA REFERENCES

The TMP68303 separates memory references into two classes: program references and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Operand reads are from the data space except in the case of the program counter relative addressing mode. All operand writes are to the data space.

2.7 REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

2.8 EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 2.4 shows the general format of the single-effective-address instruction operation word. The effective address is composed of two 3-bit fields: the mode field and the register field. The Value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 2.3. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

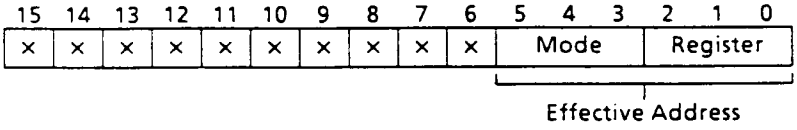


Figure 2.4 Single Effective Address Instruction Operation Word

2.8.1 Register Direct Modes

These effective addressing modes specify that the operand is in one of 16 multifunction registers.

2.8.1.1 Data Register Direct

The operand is in the data register specified by the effective address register field.

2.8.1.2 Address Register Direct

The operand is in the address register specified by the effective address register field.

2.8.2 Memory Address Modes

These effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

2.8.2.1 Address Register Indirect

The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.2.2 Address Register Indirect with Postincrement

The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whehter the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

2.8.2.3 Address Register Indirect with Predecrement

The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

2.8.2.4 Address Register Indirect with Displacement

This addressing mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.2.5 Address Register Indirect with Index

This addressing mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sing-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.3 Special Address Modes

The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

2.8.3.1 Absolute Short Address

This addressing mode requires one word of extension. The address of the operand is the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.3.2 Absolute Long Address

This addressing mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high order part of the address is the first extension word; the low order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instruction.

2.8.3.3 Program Counter with Displacement

This addressing mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.

2.8.3.4 Program Counter with Index

This addressing mode requires one word of extension. The address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference is classified as a program reference.

2.8.3.5 Immediate Data

This addressing mode requires either one or two words of extension depending on the size of the operation.

Byte Operation	:	operand is low order byte of extension word
Word Operation	:	operand is extension word
Long Word Operation	:	operand is in the two extension words, high order 16 bits are in the first extension word, low order 16 bits are in the second extension word

2.8.3.6 Implicit Reference

Some instructions make implicit reference to the program counter (PC) , the system stack pointer (SP) , the supervisor stack pointer (SSP) , the user stack pointer (USP) , or the status register (SR) . A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR
ANDI to SR
EORI to CCR
EORI to SR
ORI to CCR
ORI to SR
MOVE to CCR
MOVE to SR
MOVE from SR

2.9 EFFECTIVE ADDRESS ENCODING SUMMARY

Table 2.1 is a summary of the effective addressing modes discussed in the previous paragraphs.

Table 2.1 Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Register Direct	000	Register Number
Address Register Direct	001	Register Number
Address Register Indirect	010	Register Number
Address Register Indirect with Postincrement	011	Register Number
Address Register Indirect with Predecrement	100	Register Number
Address Register Indirect with Displacement	101	Register Number
Address Register Indirect with Index	110	Register Number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

2.10 SYSTEM STACK

The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP) . The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP) , depending on the state of the S bit in the status register. If the S bit indicates supervisor state, SSP is the active system stack pointer and the USP cannot be referenced as an address register. If the S bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

3. INSTRUCTION SET SUMMARY

This section contains an overview of the form and structure of the TMP68303 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data Movement
Integer Arithmetic
Logical
Shift and Rotate
Bit Manipulation
Binary Coded Decimal
Program Control
System Control

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

3.1 DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data move instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to register. Address move instructions allow word and long word operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), and move quick (MOVEQ). Table 3.1 is a summary of the data movement operations.

Table 3.1 Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Xx↔Xy
LEA	32	EA→An
LINK	-	An→ - (SP) SP→An SP + displacement →SP
MOVE	8, 16, 32	s →d
MOVEM	16, 32	(EA)→An,Dn An,Dn →(EA)
MOVEP	16, 32	(EA)→Dn Dn→(EA)
MOVEQ	8	#xxx→Dn
PEA	32	EA→ - (SP)
SWAP	32	Dn[31:16]↔Dn[15:0]
UNLK	-	An→SP (SP) + →An

Notes : s = source
d = destination
[ ] = bit number
-( ) = indirect with predecrement
( )+ = indirect with postincrement
#xxx = immediate data

3.2 INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotient with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 3.2 is a summary of the integer arithmetic operations.

Table 3.2 Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8,16,32 16,32	Dn + (EA) →Dn (EA) + Dn →(EA) (EA) + #xxx →(EA) An + (EA) →An
ADDX	8,16,32 16,32	Dx + Dy + X →Dx – (Ax) + – (Ay) + X →(Ax)
CLR	8,16,32	0 →(EA)
CMP	8,16,32 16,32	Dn – (EA) (EA) – #xxx (Ax) + – (Ay) + An – (EA)
DIVS	32 ÷ 16	Dn ÷ (EA)→Dn
DIVU	32 ÷ 16	Dn ÷ (EA)→Dn
EXT	8→16 16→32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 × 16→32	Dn × (EA) →Dn
MULU	16 × 16→32	Dn × (EA) →Dn
NEG	8,16,32	0 – (EA)→(EA)
NEGX	8,16,32	0 – (EA) – X→(EA)
SUB	8,16,32 16,32	Dn – (EA) →Dn (EA) – Dn →(EA) (EA) – #xxx →(EA) An – (EA) →An
SUBX	8,16,32	Dx – Dy – X →Dx – (Ax) – – (Ay) – X →(Ax)
TAS	8	(EA) – 0,1→EA[7]
TST	8,16,32	(EA) – 0

Notes : [ ] = bit number
–( ) = indirect with predecrement
( )+ = indirect with postincrement
#xxx = immediate data

3.3 LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 3.3 is a summary of the logical operations.

Table 3.3 Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	Dn∧(EA)→Dn (EA)∧Dn→(EA) (EA)∧#xxx→(EA)
OR	8, 16, 32	Dn∨(EA)→Dn (EA)∨Dn→(EA) (EA)∨#xxx→(EA)
EOR	8, 16, 32	(EA)⊕Dy→(EA) (EA)⊕#xxx→(EA)
NOT	8, 16, 32	~(EA)→(EA)

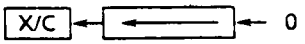
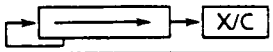
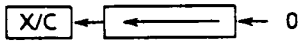
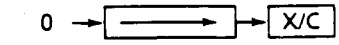
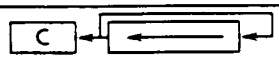
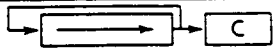
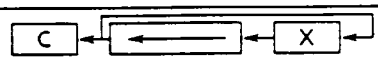
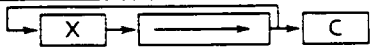
Notes: ~ = invert ∨ = logical OR
#xxx = immediate data ⊕ = logical exclusive OR
∧ = logical AND

3.4 SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by the arithmetic instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in a data register.

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates.

Table3.4 Shift and Rotate Operations

Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

3.5 BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instruction: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 3.5 is a summary of the bit manipulation operations. (Z is bit 2 of the status register.)

Table 3.5 Bit Manipulation Operations

Instruction	Operand Size	Operation
BTST	8, 32	~bit of (EA) →Z
BSET	8, 32	~bit of (EA) →Z 1 →bit of EA
BCLR	8, 32	~bit of (EA) →Z 0 →bit of EA
BCHG	8, 32	~bit of (EA) →Z ~bit of (EA) →bit of EA

Note : ~ = invert

3.6 BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD) , subtract decimal with extend (SBCD) , and negate decimal with extend (NBCD) . Table 3.6 is a summary of the binary coded decimal operations.

Table 3.6 Binary Coded Decimal Operations

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $-(Ax)_{10} + -(Ay)_{10} + X \rightarrow (Ax)$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $-(Ax)_{10} - -(Ay)_{10} - X \rightarrow (Ax)$
NBCD	8	$0 - (EA)_{10} - X \rightarrow (EA)$

Note : -( ) = indirect with predecrement

3.7 PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 3.7.

The conditional instructions provide setting and branching for the following conditions:

CC	carry clear	LS	low or same
CS	carry set	LT	less than
EQ	equal	MI	minus
F	never true	NE	not equal
GE	greater or equal	PL	plus
GT	greater than	T	always true
HI	high	VC	no overflow
LE	less or equal	VS	overflow

Table 3.7 Program Control Operations

Instruction	Operation
Conditional Bcc	Branch Conditionally (14 Conditions) 8-and 16-Bit Displacement
DBcc	Test Condition, Decrement, and Branch 16-Bit Displacement
Scc	Set Byte Conditionally (16 Conditions)
Unconditional BRA	Branch Always 8-and 16-Bit Displacement
BSR	Branch to Subroutine 8-and 16-Bit Displacement
JMP	Jump
JSR	Jump to Subroutine
Reterns RTR	Return and Restore Condition Codes
RTS	Return from Subroutine

3.8 SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 3.8.

Table 3.8 System Control Operations

Instruction	Operation
Privileged	
ANDI to SR	Logical AND to Status Register
EORI to SR	Logical EOR to Status Register
MOVE EA to SR	Load New Status Register
MOVE USP	Move User Stack Pointer
ORI to SR	Logical OR to Status Register
RESET	Reset External Devices
RTE	Return from Exception
STOP	Stop Program Execution
Trap Generating	
CHK	Check Data Register Against Upper Bounds
TRAP	Trap
TRAPV	Trap on Overflow
Status Register	
ANDI to CCR	Logical AND to Condition Codes
EORI to CCR	Logical EOR to Condition Codes
MOVE EA to CCR	Load New Condition Codes
MOVE SR to EA	Store Status Register
ORI to CCR	Logical OR to Condition Codes

4. SIGNAL AND BUS OPERATION DESCRIPTION

This section contains a brief description of the input and output signals. A discussion of bus operation during the various machine cycles and operations is also given.

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.

4.1 SIGNAL DESCRIPTION

The input and output signals can be functionally organized into the groups and the pin assignments is shown in Figure 4.1. The following paragraphs provide a brief description of the signals and a reference (if applicable) to other paragraphs that contain more detail about the function being performed.

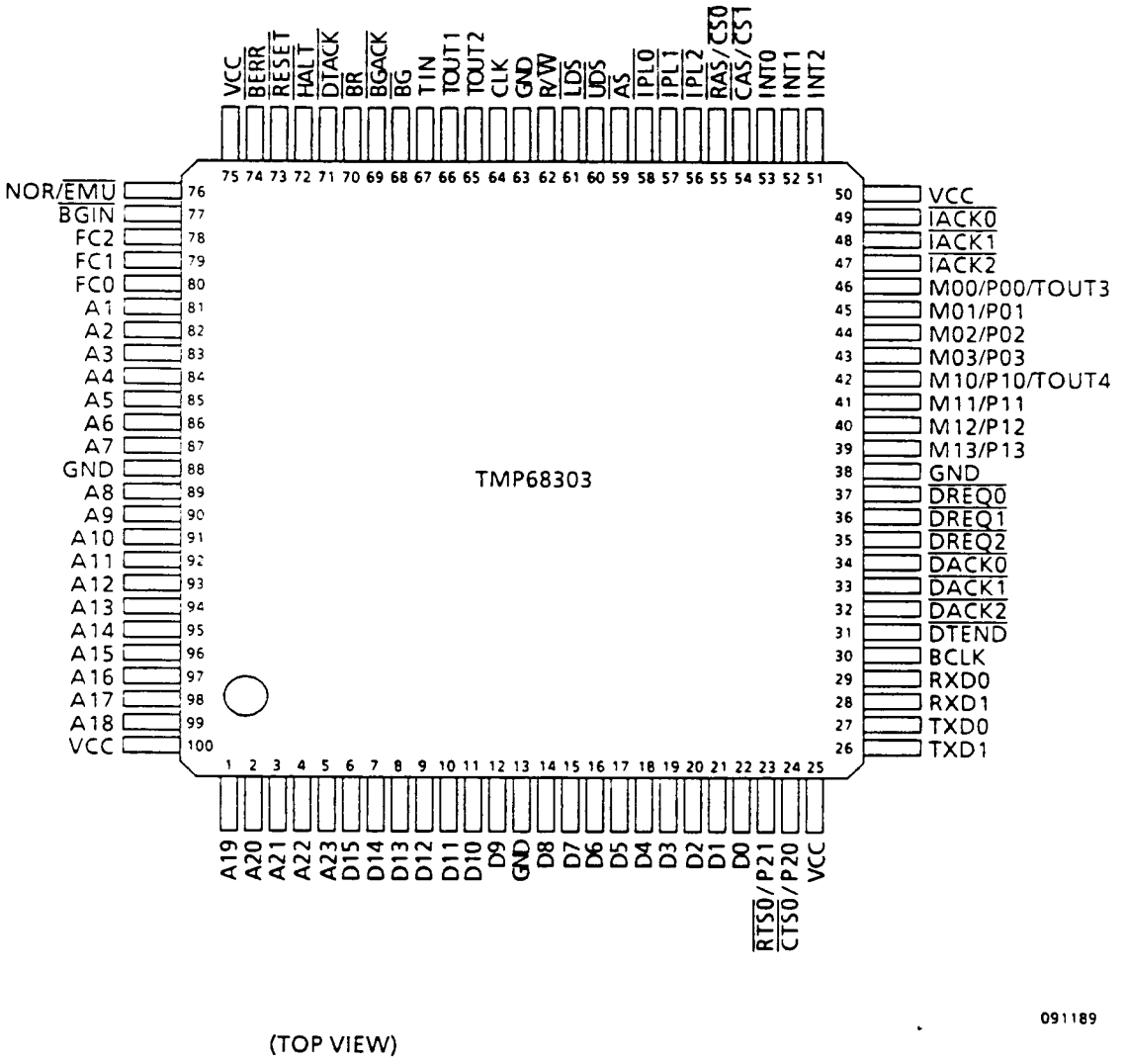


Figure 4.1 Input and Output Signals Pin Assignments

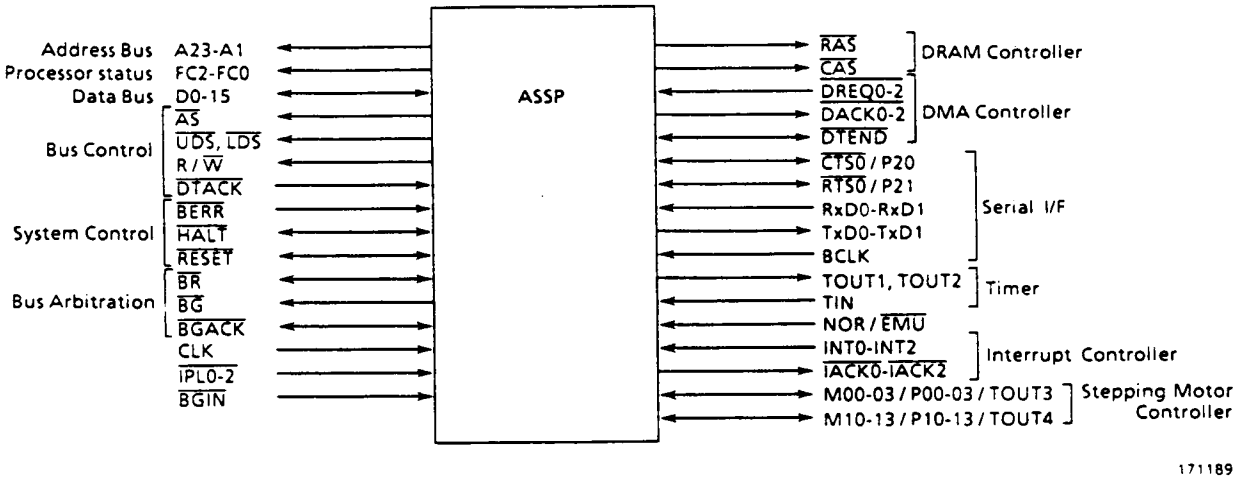


Figure 4.2 Normal Mode Signals Input/Output as Bus Master

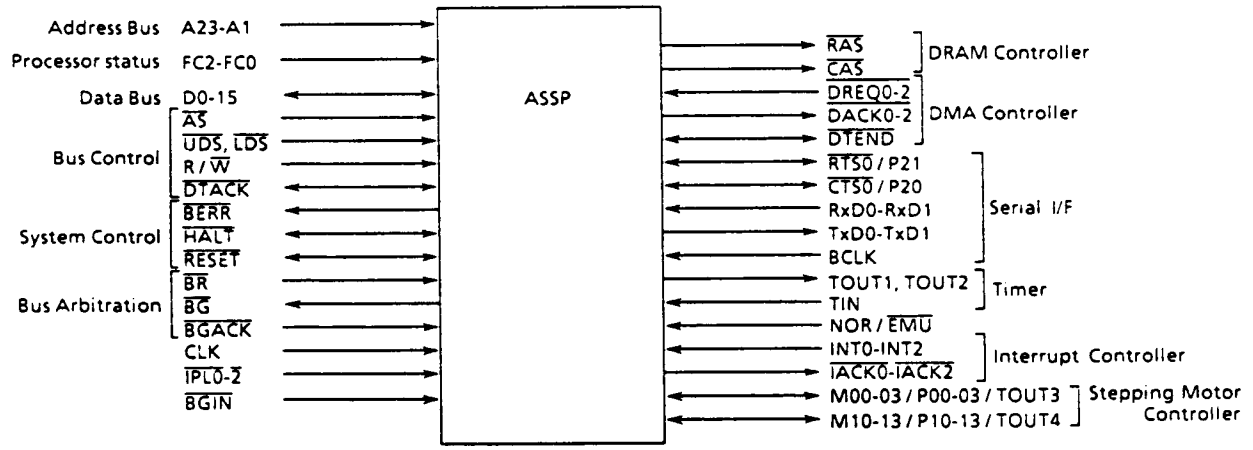


Figure 4.3 Normal Mode Signals Input/Output as not Bus Master

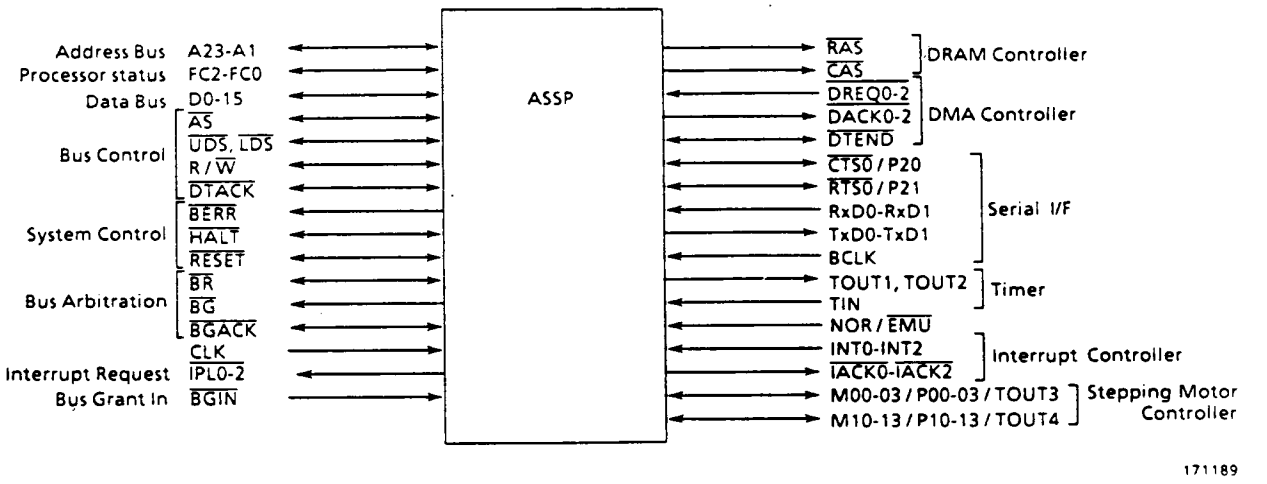


Figure 4.4 Emulation Mode Signals Input/Output

The following are brief explanations of the signals.

4.1.1 A1~A23: [Address Bus] (Output / [Input], Tristate)

This 23-bit bus can address up to 8M-words of data. Addresses are output for bus operation in all cycles except the interrupt acknowledge cycle. In the interrupt acknowledge cycle, A1~A3 indicate the interrupt service level and A4~A23 are all at high level. The signal changes to input when the bus is released and in the emulation mode.

4.1.2 D0~D15: [Data Bus] (Input / Output, Tristate)

This is a 16-bit general-purpose data bus that can transfer data in byte or word units. In an interrupt acknowledge cycle, the external device sends vector numbers to D0~D7. The data transfer direction is reversed in the emulation mode.

4.1.3 Asynchronous Bus Control Signals

4.1.3.1 $\overline{AS}$: [Address Strobe] (Output / [Input], Low Active, Tristate)

This signal indicates that there is a valid address on the address bus.
The signal changes to input in the emulation mode.

4.1.3.2 R/$\overline{W}$: [Read/Write] (Output / [Input], Tristate)

This signal indicates whether the data bus transfer is a read (High) or written (Low). As shown in Table 4.1, this signal is used in combination with $\overline{UDS}$ and $\overline{LDS}$. The signal changes to input in the emulation mode.

4.1.3.3 $\overline{UDS}$, $\overline{LDS}$: [Upper and Lower Data Strobes] (Output / [Input], Low Active, Tristate)

These signals control the flow of data on the data bus. The processor reads from the data bus when R/$\overline{W}$ is High and the core processor writes to the data bus when R/$\overline{W}$ is Low. The signal changes to input in the emulation mode.

4.1.3.4 $\overline{DTACK}$: [Data Transfer Acknowledge] (Input / [Output], Low Active)

This signal indicates the end of a data transfer. When the assertion of $\overline{DTACK}$ during the core processor read cycle is acknowledged, data is latched and the bus cycle ends. When $\overline{DTACK}$ is recognized during the write cycle, the bus cycle ends. (See 4.4 Asynchronous and synchronous operation.)

4.1.4 Bus Arbitration Control (Interrupt Request Output)

These three signals are used by the bus arbitration circuit to determine which device will be a bus master. In the emulation mode, interrupt requests are output by the interrupt controller.

4.1.4.1 $\overline{BR}$: [Bus Request] (Input / [Output], Low Active)

This signal is wire ORed connected with all other devices which could become the bus masters. This input signal indicates another device desire to become the bus master.

Table 4.1 Data Strobe Control of Data Bus

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

* : These conditions are result of current implementation and may not appear on future devices.

4.1.4.2 $\overline{BG}$: [Bus Grant] (Output, Low Active)

This signal indicates to all other devices which could become the bus master that the core processor will release the bus control at the end of the current bus cycle.

4.1.4.3 $\overline{BGACK}$: [Bus Grant Acknowledge] (Input / [Output], Low Active)

This signal indicates that another device has become the bus master. The signal should not asserted unless the following four conditions are satisfied.

(1) $\overline{BG}$ is asserted.

(2) $\overline{AS}$ is not asserted (that is, the core processor is not using the bus).

(3) $\overline{DTACK}$ is not asserted (that is, the memory and peripheral devices are not using the bus).

(4) $\overline{BGACK}$ is not asserted (no other device is requesting the bus).

4.1.5 System Control

4.1.5.1 $\overline{BERR}$: [Bus Error] (Input/[Output], Low Active)

This signal reports to the core processor that there is a problem in the current cycle in the following cases.

(1) There is no response from a device.

(2) Interrupt vector number acquisition failure.

(3) An invalid access request determined by a memory management unit is made.

(4) Other errors (differs according to the application).

$\overline{BERR}$, together with $\overline{HALT}$, determines whether exception processing should be performed or the current bus cycle is re-executed. (See item 4.2.4 Bus error and halt operation.)

4.1.5.2 $\overline{RESET}$: [Reset] (Input/Output, Low Active, Open Drain)

This signal resets the core processor (starts the system initialization sequence) in response to an external reset signal. In response to an internally generated reset (result of a $\overline{RESET}$ instruction), this signal also operates as an external device reset signal but does not change the internal status of the core processor. A total system resets (core processor and external devices) is applied by asserting both $\overline{HALT}$ and $\overline{RESET}$ at the same time from the outside. (See item 4.2.5 Reset operation.)

4.1.5.3 $\overline{HALT}$: [Halt] (Input/Output, Low Active, Open Drain)

When asserted from an external device, this signal halts the core processor when the current bus cycle ends. Using this input signal to halt the core processor negates all control signals from the core processor and sets all tristate signals to high impedance. (See item 4.2.4 Bus errors and halt operation concerning the relationship to $\overline{BERR}$.)

When the core processor halts the execution of an instruction such as in a double bus fault condition (see item 4.2.4.4 Double bus faults), $\overline{HALT}$ is driven to Low level by the core processor and the fact that the core processor has halted is reported to the external device.

4.1.6 FC0, FC1, FC2: [Function Code] (Output/[Input], High Active, Tristate)

These three signals indicate the core processor state (user or supervisor) and the current cycle type as shown in Table 4.2. The data indicated by these signals are valid when $\overline{AS}$ is asserted.

Table 4.2 Function Code Outputs

FC2	FC1	FC0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

The signals change to input in the emulation mode.

4.1.7 CLK: [Clock] (Input, High Active)

This signal is TTL compatible and is buffered internally for use in creating internal clocks. Clock input must not be gated off any time and the maximum and minimum pulse width conditions must be satisfied.

4.1.8 Peripheral Signals

4.1.8.1 P00~P03/M00~M03/TOUT3 [I/O Port] (Input/Output)

These are output signals from the 4-bit general-purpose I/O port or stepping motor control port 0. P00 is also used for the timer channel 3 output signal.

4.1.8.2 P10~P13/M10~M13/TOUT4 [I/O Port] (Input/Output)

These are output signals from the 4-bit general-purpose I/O port or stepping motor control port 1. P10 is also used for the timer channel 4 output signal.

4.1.8.3 $\overline{RAS}$ / $\overline{CS0}$ [Row Address Strobe / chip select 0] (Output, Low Active)

This is the row address strobe to the DRAM or address decoder signal.

4.1.8.4 $\overline{CAS}$ / $\overline{CS1}$ [Column Address Strobe/ chip select 1] (Output, Low Active)

This is the column address strobe to the DRAM or address decoder signal.

4.1.8.5 $\overline{DREQ0\sim2}$ [DMA Request] (Low Active)

Request signals which are input to start DMAC.

4.1.8.6 $\overline{DACK0\text{-}2}$ [DMA Acknowledge] (Output, Low Active)

Acknowledge signals output by DMAC to specify I/O devices.

4.1.8.7 $\overline{DTEND}$ [DMA End] (Input / Output, Low Active, Open Drain)

The input signal of prohibition for channel request, the end signal input to compulsorily end DMA operation or the output signal that show transfer complete.

4.1.9.8 $\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$ [Interrupt Priority Level] (Input, Low Active)

These pins are not connected in the normal mode. In the emulation mode, however, these pins are used to output interrupt requests from the interrupt controller.

4.1.8.9 TOUT1, TOUT2 [Timer Output] (Output)

The timer channels 1 and 2 output signals.

4.1.8.10 TIN [Timer Input] (Input, Low Active)

Input signals to 0,1 and 2 timer channels.

4.1.8.11 $\overline{\text{RTS0}}$/P21 [I/O Port] (Input/Output)

Used as the serial interface channel 0 request-to-send output signal pins and general-purpose I/O port pins.

4.1.8.12 $\overline{\text{CTS0}}$/P20 [I/O Port] (Input/Output)

Used as the serial interface channel 0 clear-to-send input signal pins and general-purpose I/O port pins.

4.1.8.13 RxD0, RxD1, [Receive Data] (Input)

The serial interface data inputs.

4.1.8.14 TxD0, TxD1, [Send Data] (Output)

The serial interface data outputs.

4.1.8.15 BCLK [Baud Rate, Clock] (Input)

The clock used to generate the serial interface baud rates.

4.1.8.16 INT0, INT1, INT2 [Interrupt Request] (Input)

Interrupt requests.

4.1.8.17 $\overline{\text{IACK0}}$, $\overline{\text{IACK1}}$, $\overline{\text{IACK2}}$ [Interrupt Acknowledge] (Output, Low Active)

Indicates IACK cycles.

4.1.9 NOR/$\overline{\text{EMU}}$ [Mode Switch] (Input)

The signal used to switch between the normal and emulation mode.

4.1.10 $\overline{\text{BGIN}}$ (Bus Grant In] (Input)

This signal is $\overline{\text{BG}}$ of CPU output which built-in DMAC accept at emulation mode.

4.1.11 Signal Summary

Table 4.3 is a summary of all the signals discussed in the previous paragraphs.

Table 4.3 Signal Summary

Signal Name	Mnemonic	Input / Output	Active State	3 State	Hi-Z	
					On $\overline{HALT}$	On $\overline{BG}$
Address Bus	A1~A23	Output (Input)	High	Yes	Yes	Yes
Data Bus	D0~D15	Input / Output (Outpu / Input)	High	Yes	Yes	Yes
Address Strobe	$\overline{AS}$	Output (Input)	Low	Yes	No	Yes
Read / Write	R/$\overline{W}$	Output (Input)	Read-High Write-Low	Yes	No	Yes
Upper and Lower Data strobes	$\overline{UDS}$, $\overline{LDS}$	Output (Input)	Low	Yes	No	Yes
Data Transfer Acknowledge	$\overline{DTACK}$	Input (Output)	Low	No	No[1]	No[1]
Bus Request	$\overline{BR}$	Input (Output)	Low	No	No[1]	No[1]
Bus Grant	$\overline{BG}$	Output	Low	No	No	No
Bus Grant Acknowledge	$\overline{BGACK}$	Input (Output)	Low	No	No[1]	No[1]
Interrupt Priority Level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	Input (Output)	Low	Yes	No	No
Bus Error	$\overline{BERR}$	Input (Output)	Low	No	No[1]	No[1]
Reset	$\overline{RESET}$	Input (Output)	Low	Yes	No[1]	No[1]
Halt	$\overline{HALT}$	Input (Output)	Low	Yes	No[1]	No[1]
Function Code Output	FC0, FC1, FC2	Output (Input)	High	Yes	No	Yes
Clock	CLK	Input	High	No	No	No
Power Input	V_{CC}	Input	–	–	–	–
Ground	GND	Input	–	–	–	–
Row Address Strobe	$\overline{RAS}$	Output	Low	–	–	–
Column Address Strobe	$\overline{CAS}$	Output	Low	–	–	–
DMA Request	$\overline{DREQ0~2}$	Input	Low	–	–	–
DMA Acknowledge	$\overline{DACK0~2}$	Output	Low	–	–	–
DMA End	$\overline{DTEND}$	Input / Output	Low	–	No[1]	No[1]
Clear to Send	$\overline{CTS0}$	Input	Low	Yes	–	–
Request to Send	$\overline{RTS0}$	Output	Low	Yes	–	–
Receive Data	RxD0-1	Input	–	–	–	–
Transfer Date	TxD0-1	Output	–	–	–	–
Baudrate clock	BCLK	Input	–	–	–	–
I/O Port	P00~03,10~13,20, 21	Input / Output	–	Yes	–	–
Timer Output	Tout 1~4	Output	–	–	–	–
Timer Input	TIN	Input	Low	–	–	–
Mode change	NOR / $\overline{EMU}$	Input	–	–	–	–
Interrupt Request	INT0~2	Input	High	–	–	–
Interrupt Acknowledge	$\overline{IACK0~2}$	Output	Low	–	–	–
SMC Output	M00~03,10~13	Output	–	–	–	–
Bus Grant In	$\overline{BGIN}$	Input	Low	Yes	No	No

Note :1. Open drain ( ) Emulation Mode

4.2 BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

4.2.1 Data Transfer Operations

Transfer of data between devices involves the following leads:

1. address bus A1~A23
2. data bus D0~D15
3. control signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the TMP68303 for interlocked multiprocessor communications.

4.2.1.1 Read Cycle

During a read cycle, the core processor receives data from the memory or a peripheral device. The core processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the core processor reads both upper and lower both simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the core processor uses an internal A0 bit to determine which byte to read and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. When the data is received, the core processor correctly positions it internally.

A word read cycle flowchart is given in Figure 4.2. A byte read cycle flowchart is given in Figure 4.3. Read cycle timing is given in Figure 4.4. Figure 4.5 details word and byte read cycle operations.

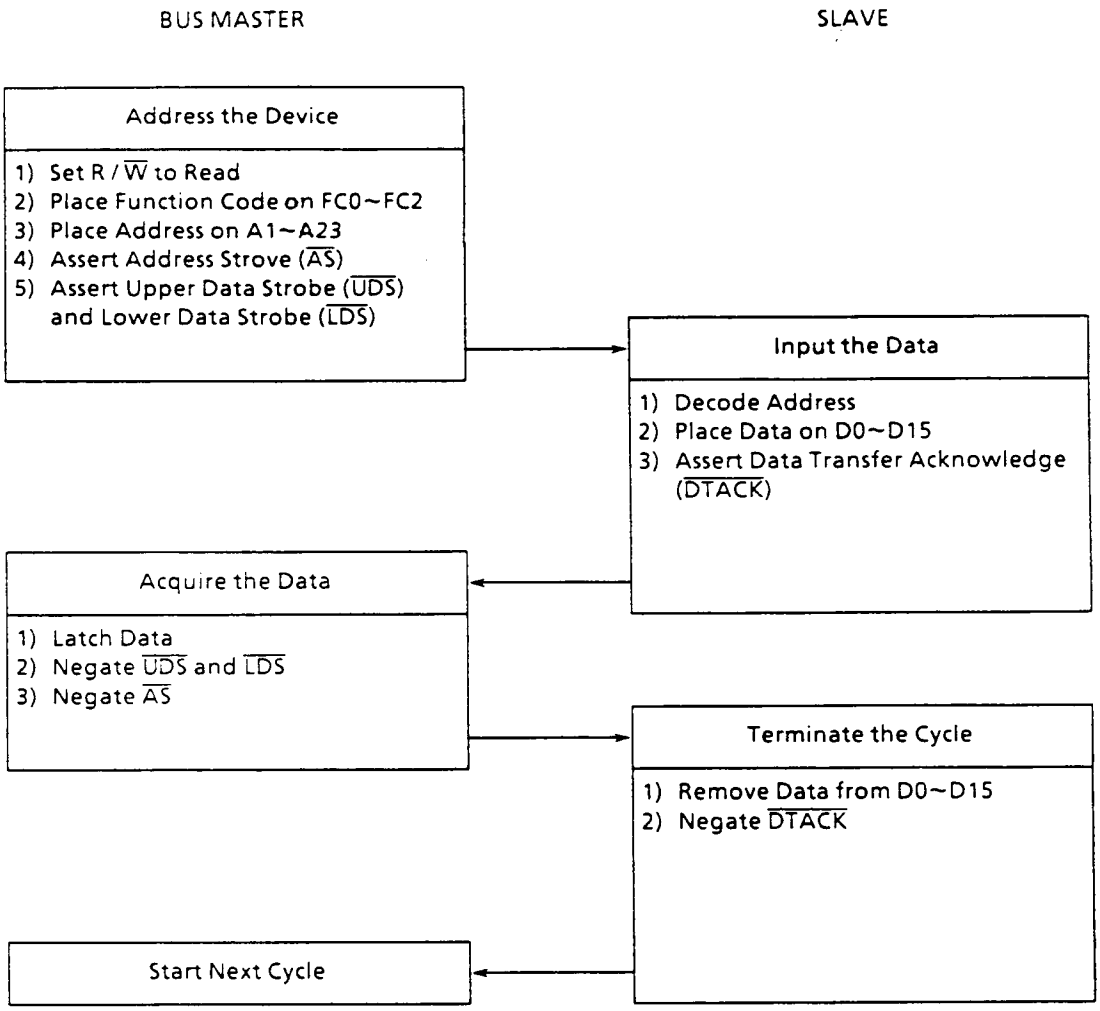


Figure 4.2 Word Read Cycle Flowchart

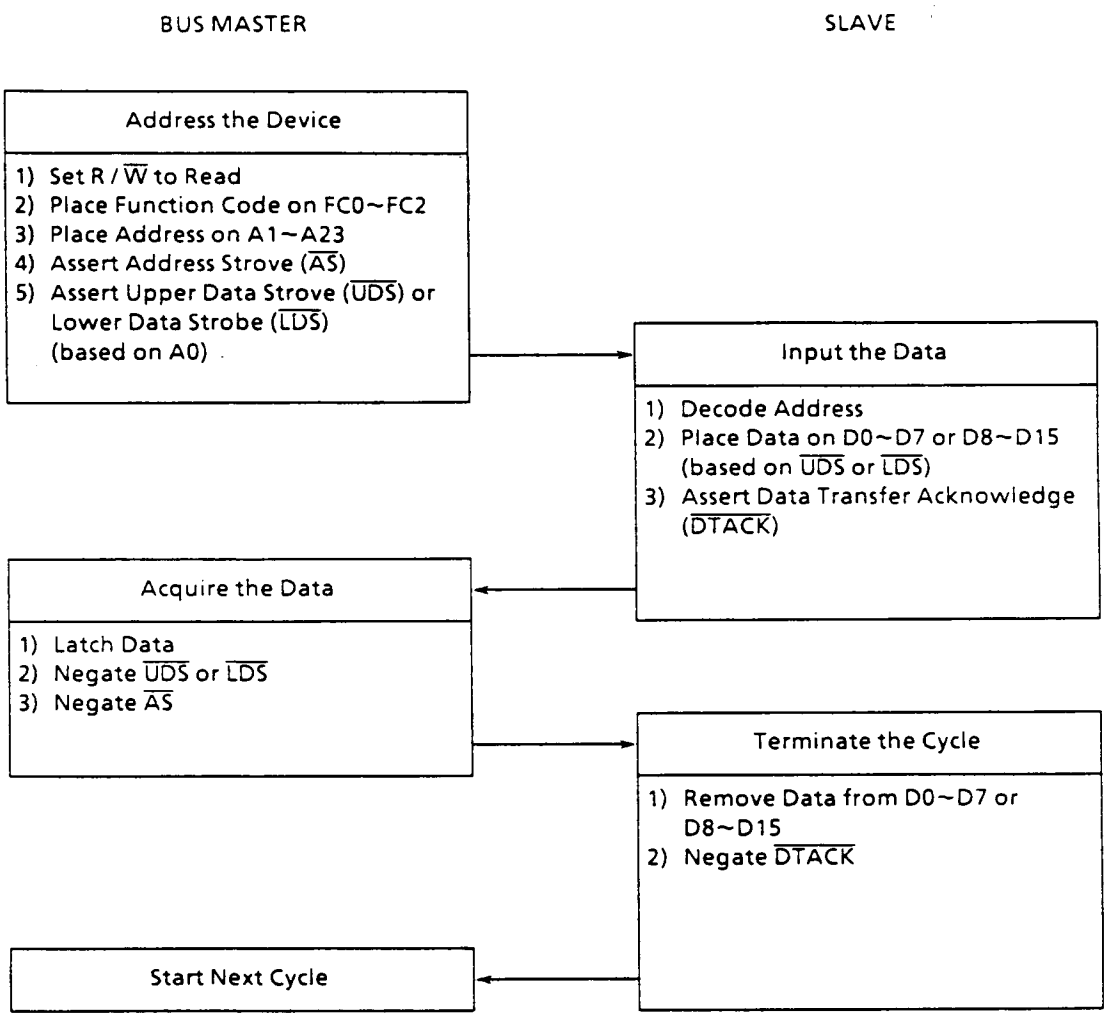


Figure 4.3 Byte Read Cycle Flowchart

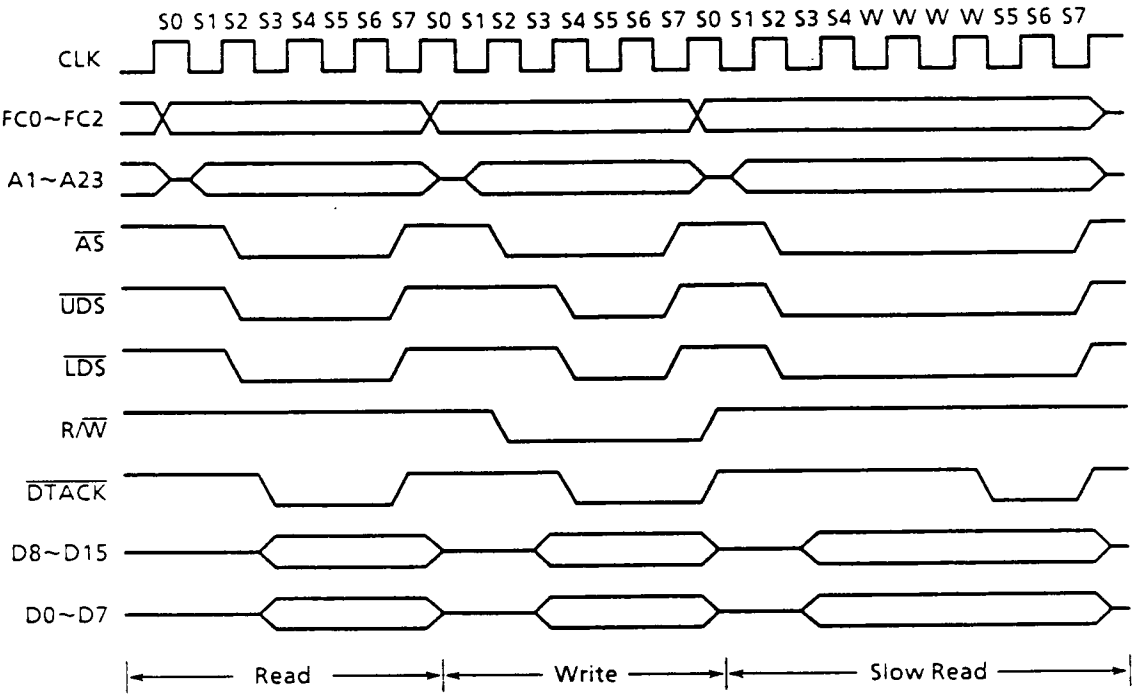


Figure 4.4 Read and Write Cycle Timing Diagram

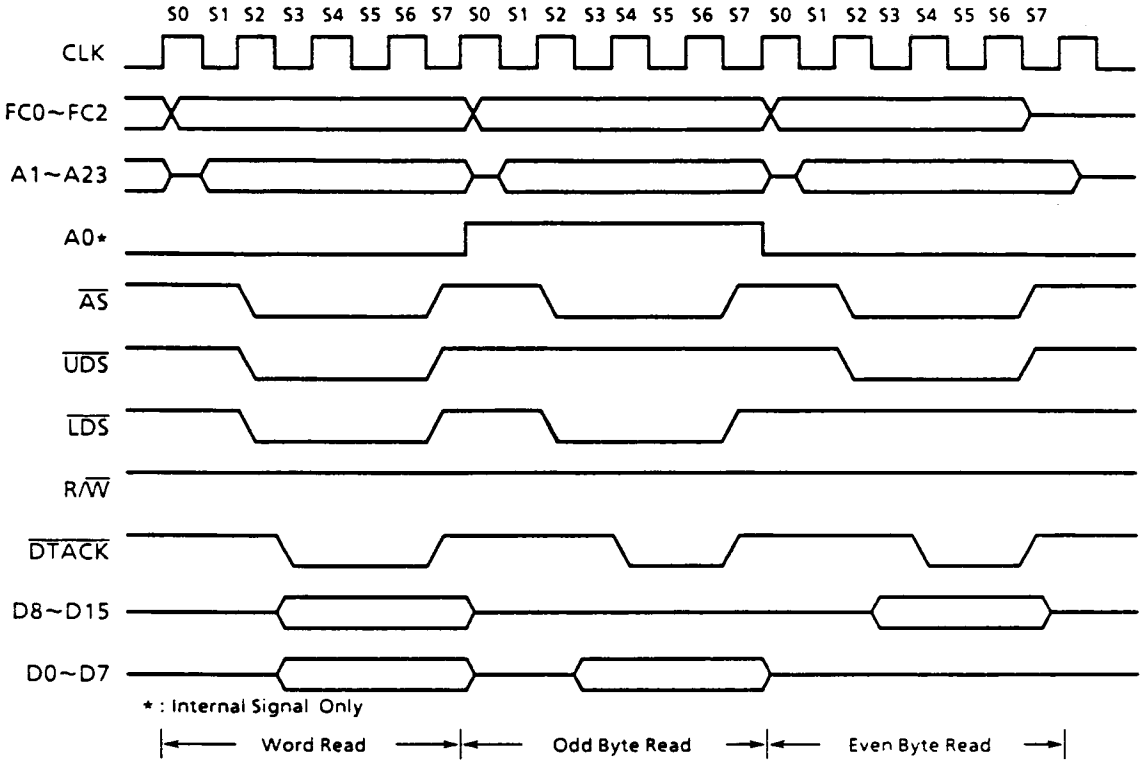


Figure 4.5 Word and Byte Read Cycle Timing Diagram

4.2.1.2 Write Cycle

During a write cycle, the core processor sends data to either the memory or a peripheral device. The core processor writes bytes of data in all cases. If the instruction specifies a word operation, the core processor writes both bytes. When the instruction specifies a byte operation, the core processor uses an internal A0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. A word write flowchart is given in Figure 4.6. A byte write cycle flowchart is given in Figure 4.7. Write cycle timing is given in Figure 4.4. Figure 4.8 details word and byte write cycle operation.

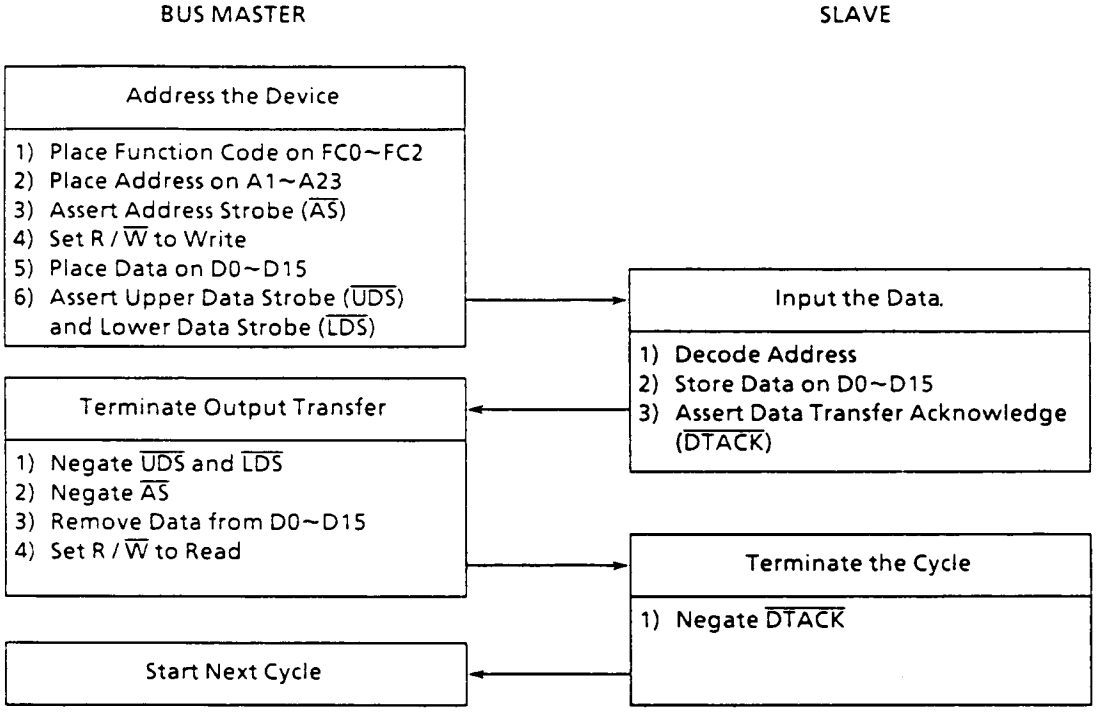


Figure 4.6 Word Write Cycle Flowchart

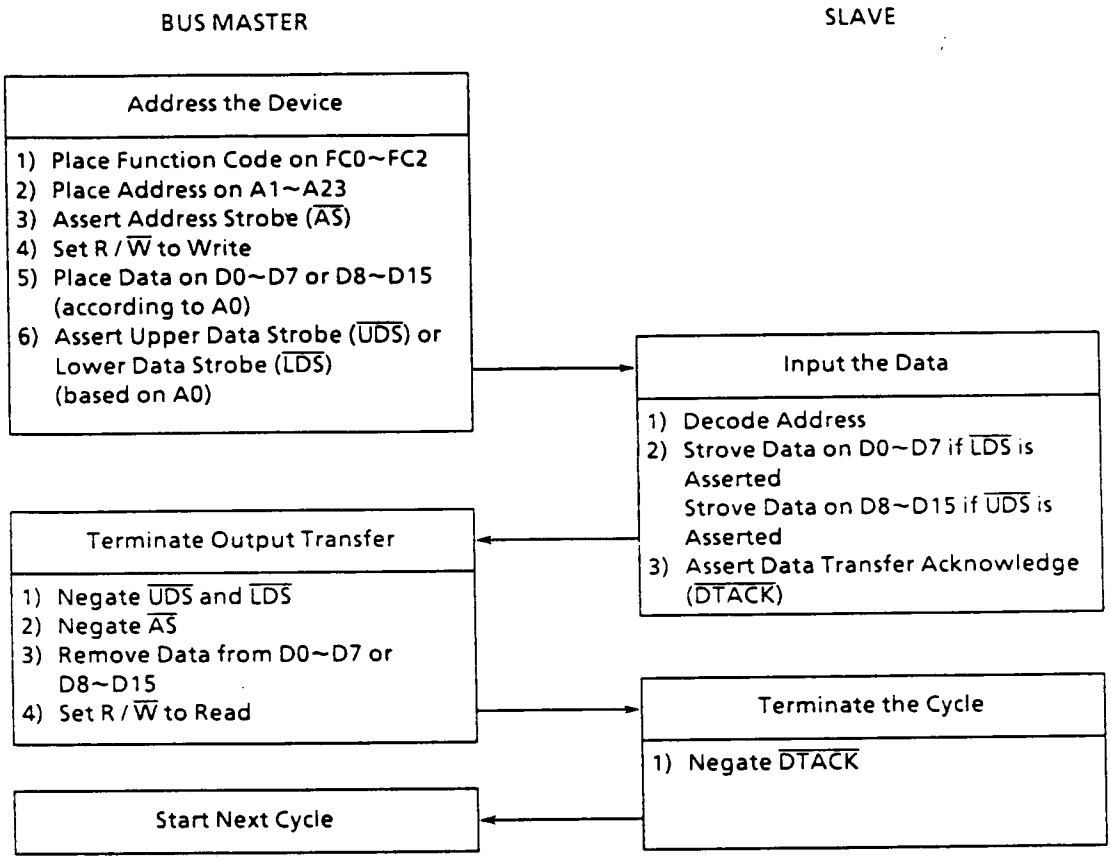


Figure 4.7 Byte Write Cycle Flowchart

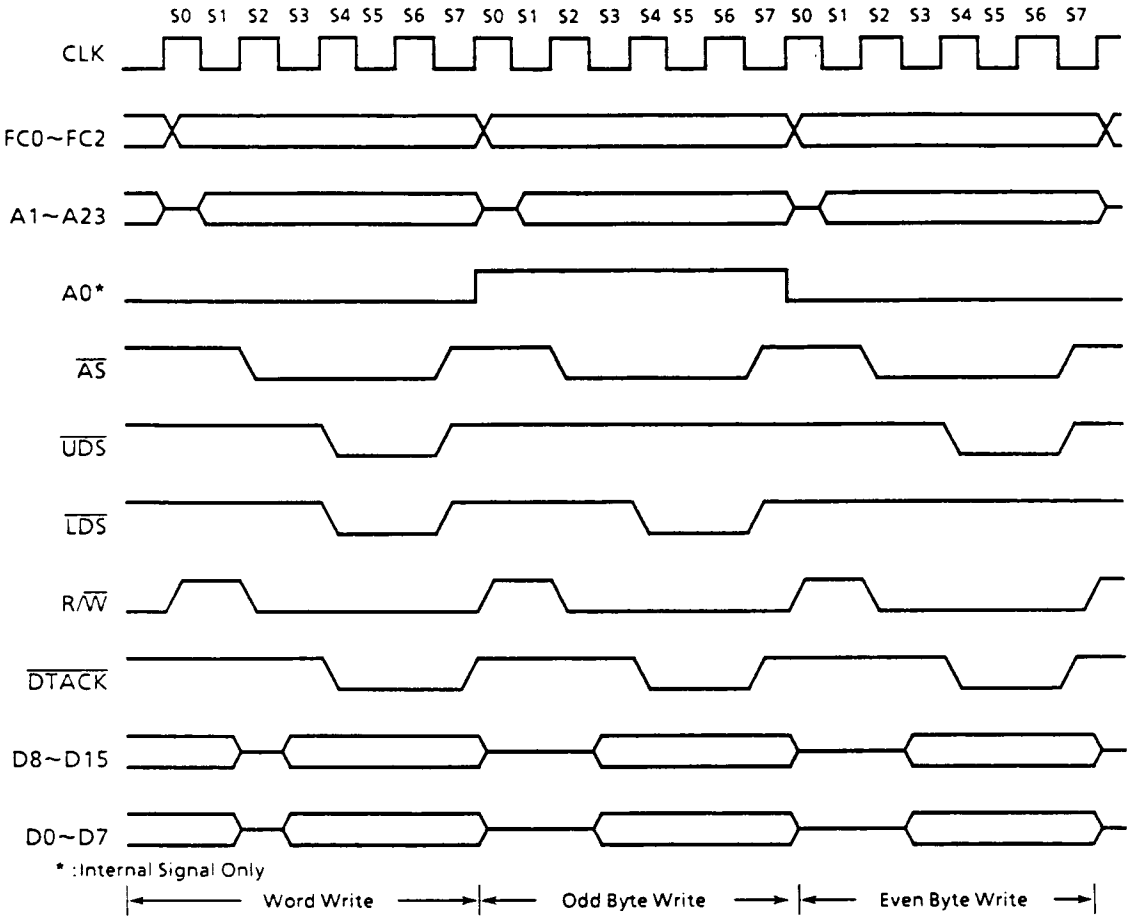


Figure 4.8 Word and Byte Write Cycle Timing Diagram

4.2.1.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the TMP68303, this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycles and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write flowchart is given in Figure 4.9 and a timing diagram is given in Figure 4.10.

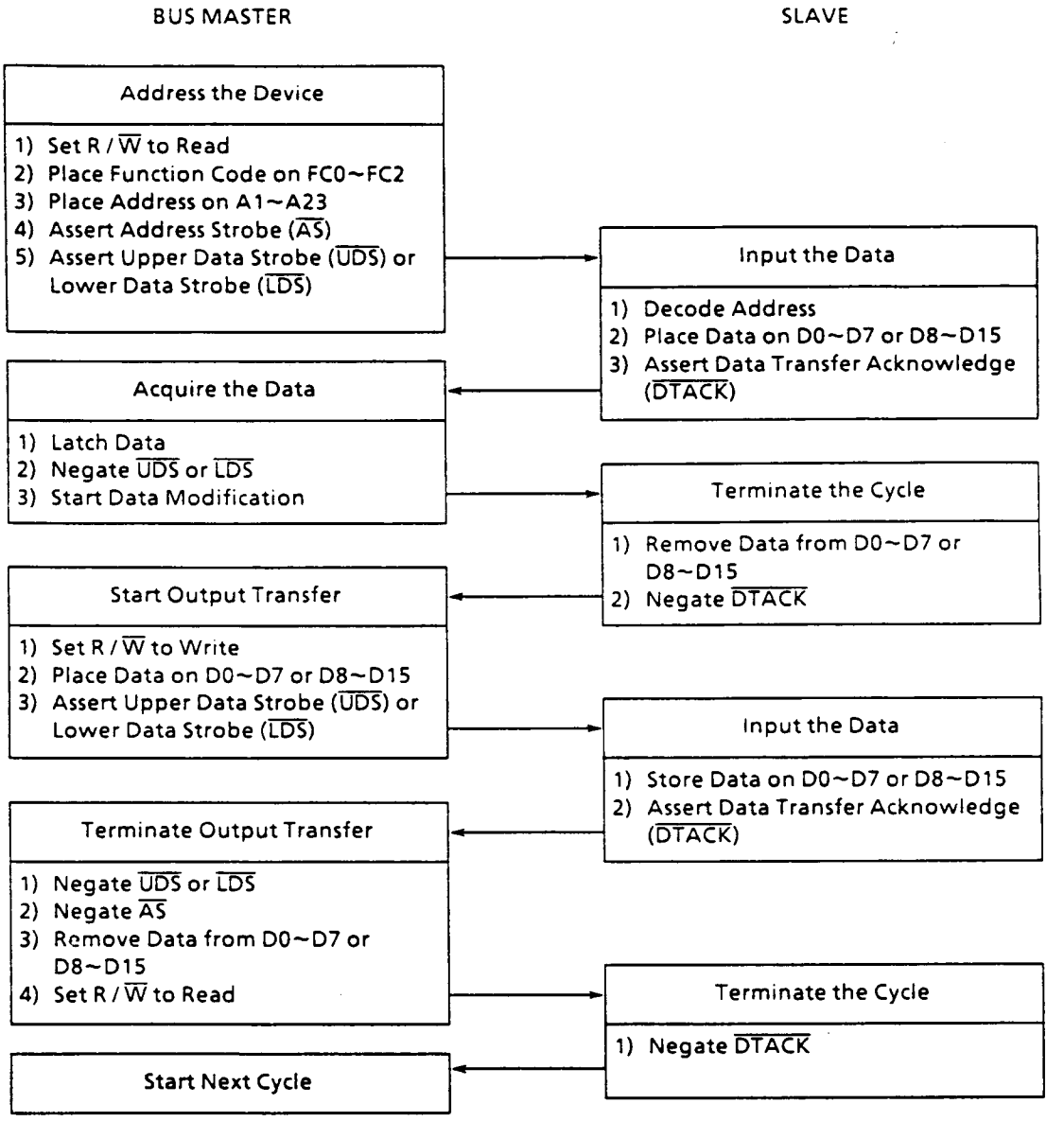


Figure 4.9 Read-Modify-Write Cycle Flowchart

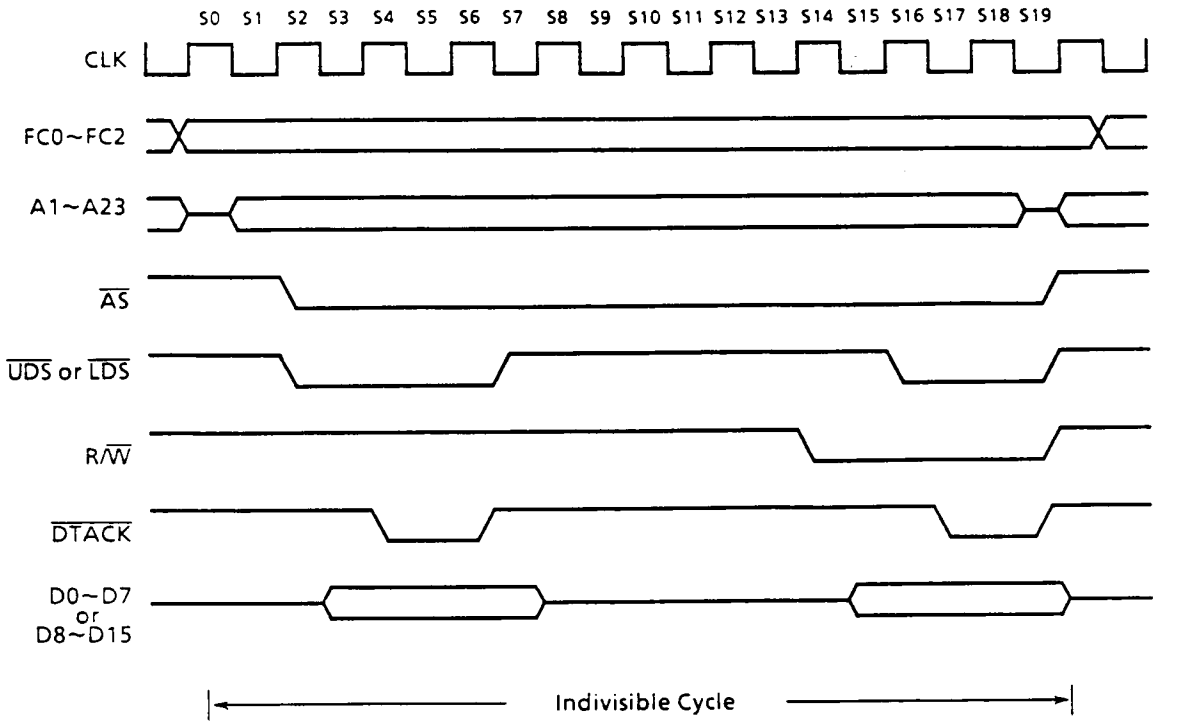


Figure 4.10 Read-Modify-Write Cycle Timing Diagram

4.2.2 Bus Arbitration

Bus arbitration is technique used by master-type devices to request, be granted, and acknowledge bus mastership. In its simplest form, it consists of the following:

1. asserting a bus mastership request
2. receiving a grant that the bus is available at the end of the current cycle
3. acknowledging that mastership has been assumed

Figure 4.11 is a flowchart showing the detail involved in a request from a single device. Figure 4.12 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles. The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the core processor and one device capable of bus mastership. In systems having a number of devices capable of the busmastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge ($\overline{BGACK}$) signal.

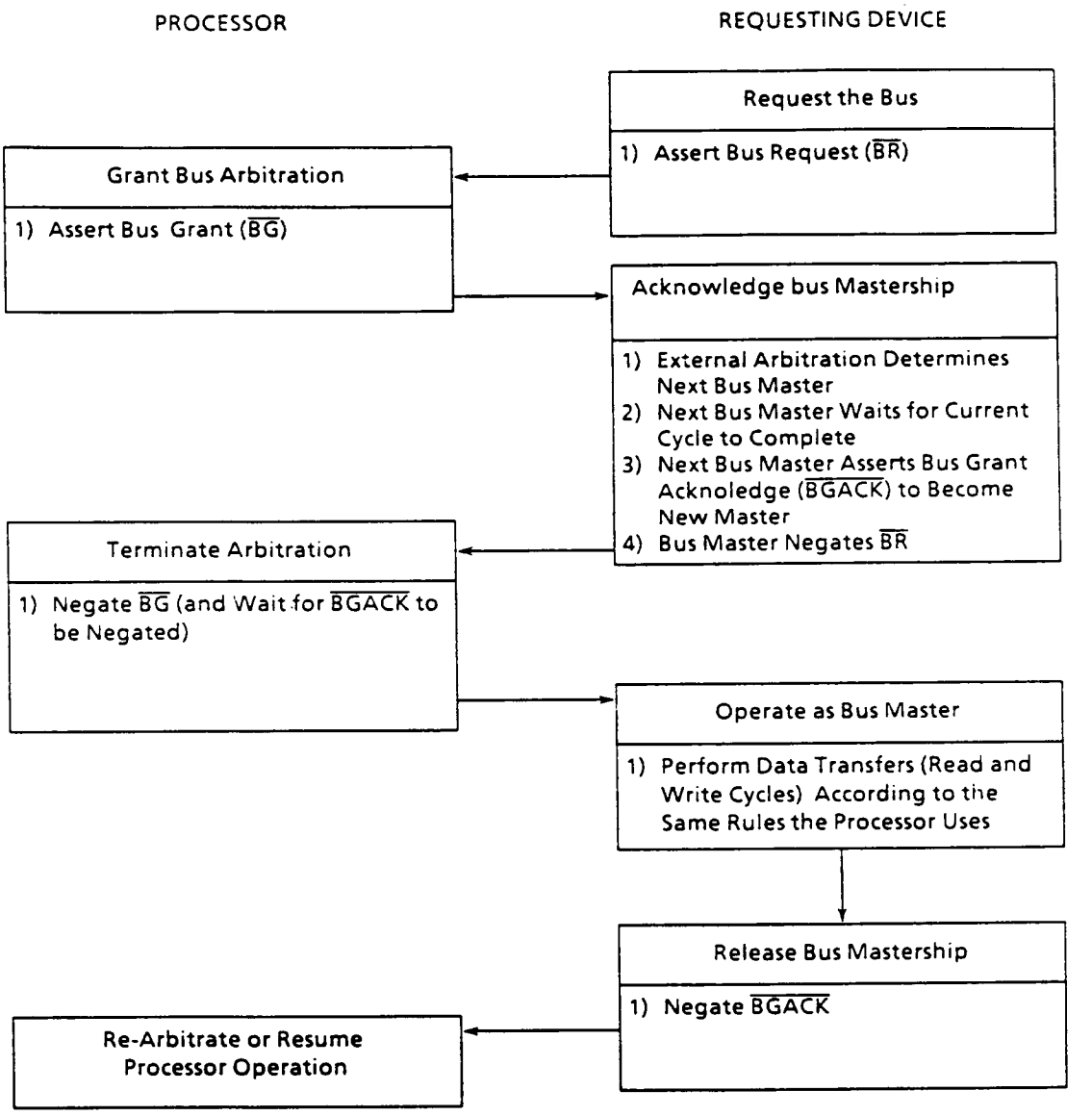


Figure 4.11 Bus Arbitration Cycle Flowchart

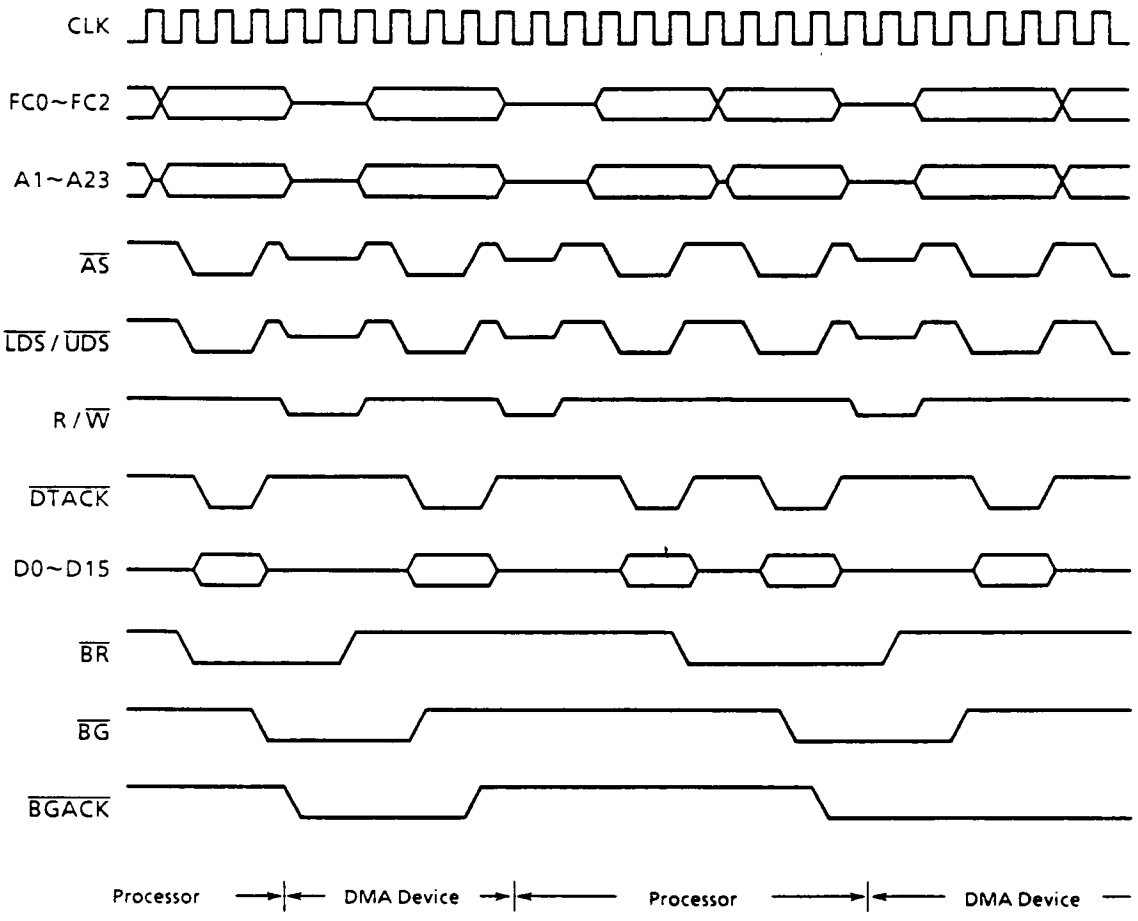


Figure 4.12 Bus Arbitration Cycle Timing Diagram

4.2.2.1 Requesting the Bus

External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{BR}$) signal. This is a wire-ORed signal (although it need not be constructed from open-collector devices) that indicates to the processor that some external device requires control of the external bus. The core processor is effectively at a lower bus priority level than the external device and will relinquish the bus after it has completed the last bus cycle it has started.

When no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

4.2.2.2 Receiving the Bus Grant

The core processor asserts bus grant ($\overline{BG}$) as soon as possible. Normally this is immediately after internal synchronization. The only exception to this occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe ($\overline{AS}$) signal. In this case, bus grant will be delayed until $\overline{AS}$ is asserted to indicate to external devices that a bus cycle is being executed.

The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The core processor is not affected by the external method of arbitration as long as the protcol is obeyed.

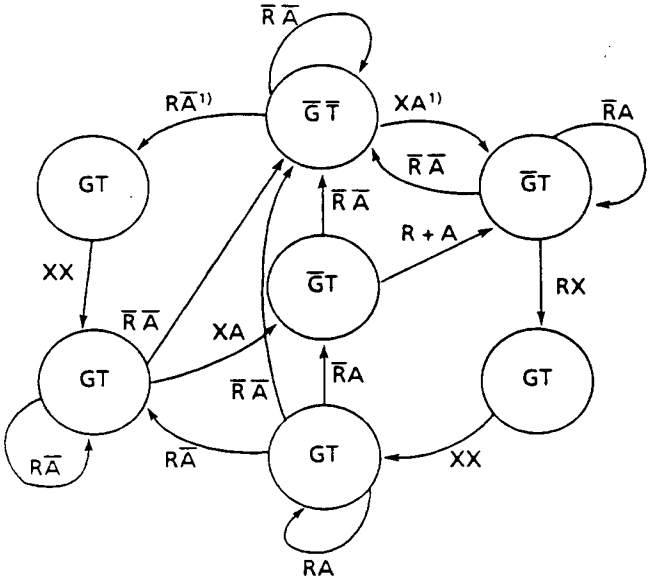
4.2.2.3 Acknowledgement of Mastership

Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowlegde, and bus grant acknowledge are negated before issuing its own $\overline{BGACK}$. The negation of the address strobe indicates that the previous master has completed its cycle; the negation of bus grant acknowledge indicates that the previous master has released the bus. (While address strobe is asserted, no device is allowed to "break into" a cycle.) The negation of data transfer acknowledge indicates the previous slave has terminated its connection to the previous master. Note that in some applications data transfer acknowledge might not enter into this function. General purpose devices would then be connected such that they were only dependent on address strobe. When bus grant acknowledge is issued, the device is a bus master until it negates bus grant acknowledge. Bus grant acknowledge should not be negated until after the bus cycle(s) is (are) completed. Bus mastership is terminated at the negation of bus grant acknowledge.

The bus request from the granted device should be dropped after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of the bus grant. Refer to "4.2.3 Bus Arbitration Control". Note that the core processor does not perform any external bus cycles before it re-asserts bus grant.

4.2.3 Bus Arbitration Control

The bus arbitration control unit in the TMP68303 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 4.13. All asynchronous signals to the TMP68303 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 4.14). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.



R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic[2)]
X = Don't Care

Notes :
1) State machine will not change if the bus is S0 or S1. Refer to "4.2.3 Bus Arbitration Control".
2) The address bus will be placed in the high-impedance state if T is asserted and $\overline{AS}$ is negated.

Figure 4.13 TMP68303 Bus Arbitration Unit State Diagram

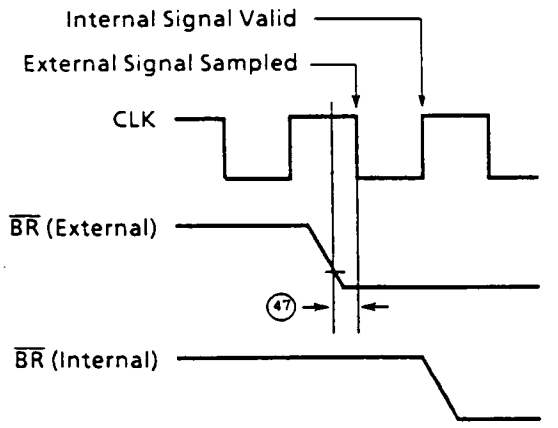


Figure 4.14 Timing Relationship of External Asynchronous Inputs to Internal Signals

As shown in Figure 4.13, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is labeled G and the internal three-state control signal T. If T is true, the address, data, and control buses are placed in a 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 after the internal signal is valid.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 4.15. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 4.16.

If a bus request is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0) , $\overline{BG}$ will not be asserted on the next rising edge. Instead, $\overline{BG}$ will be delayed until the second rising edge following its internal assertion. This sequence is shown in Figure 4.17.

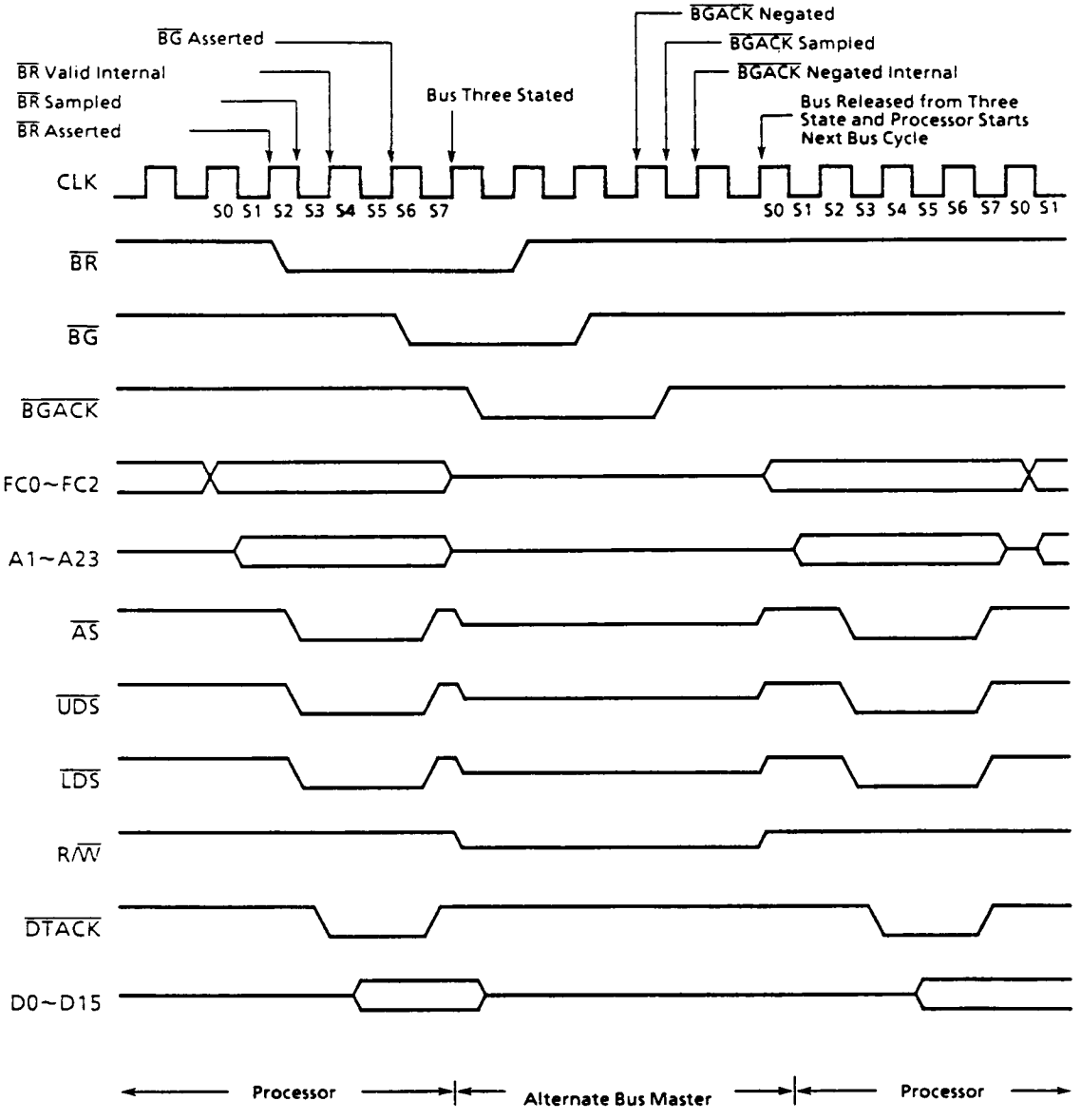


Figure 4.15 Bus Arbitration Timing Diagram — Processor Active

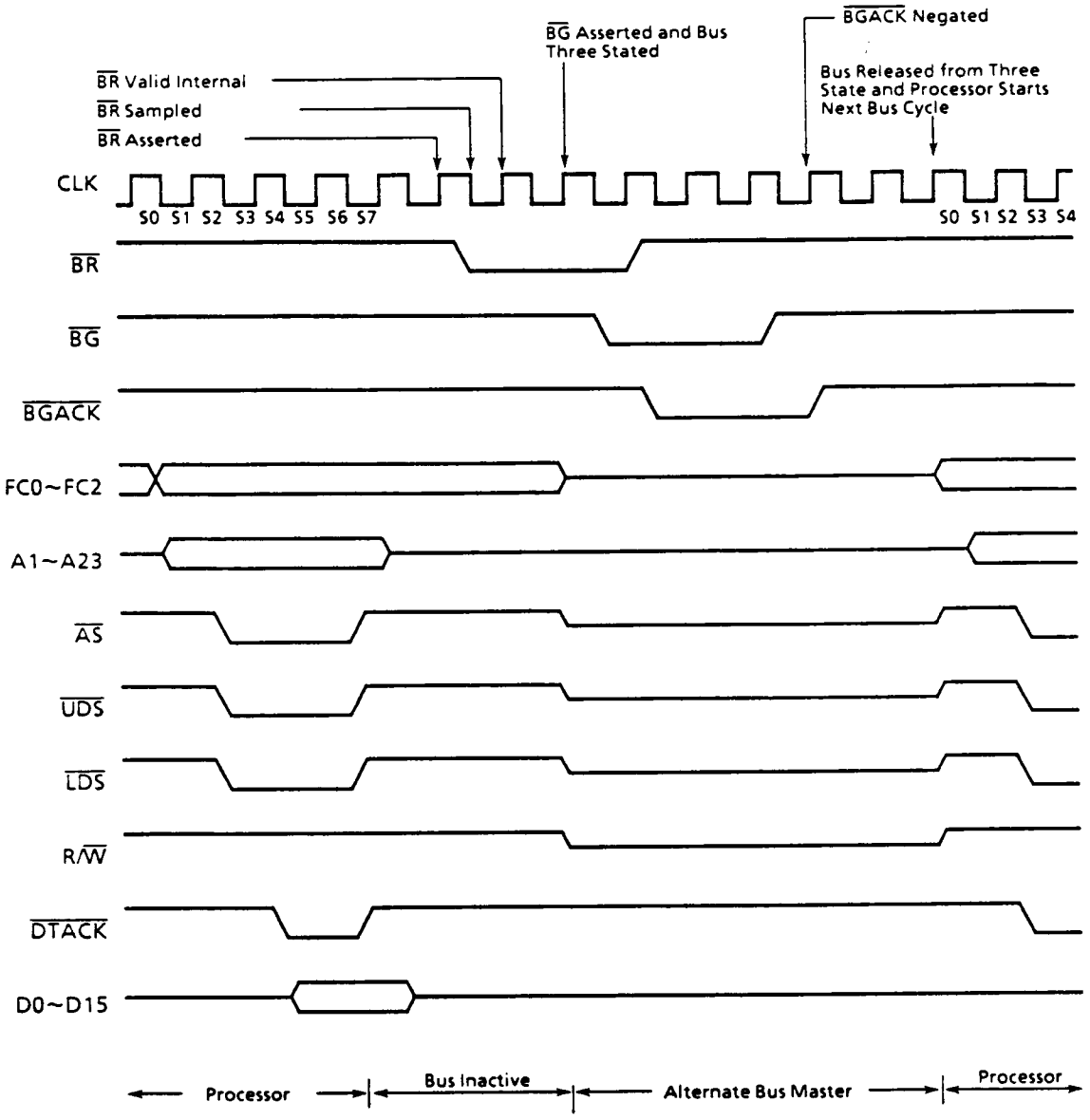


Figure 4.16 Bus Arbitration Timing Diagram – Bus Inactive

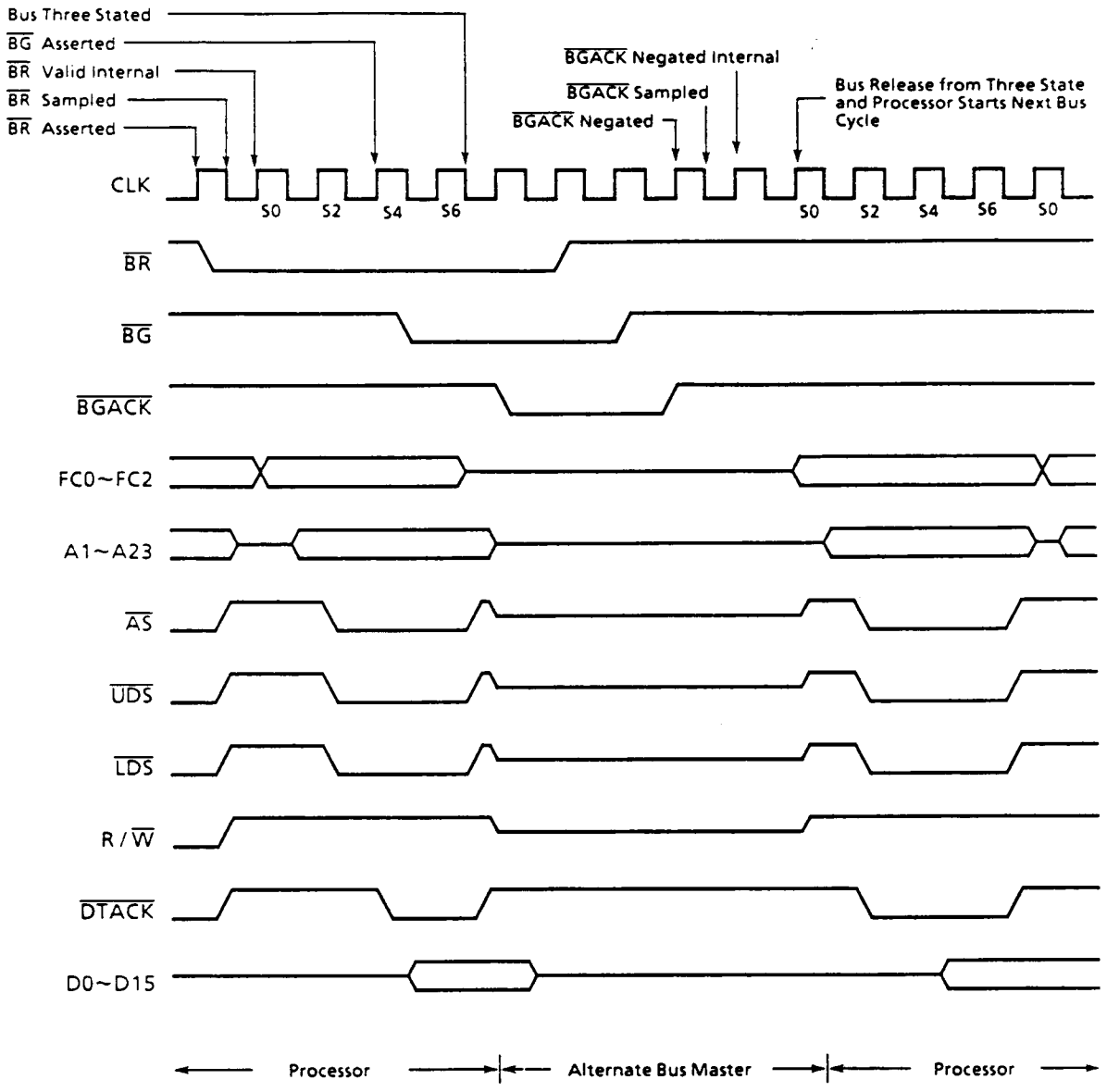


Figure 4.17 Bus Arbitration Timing Diagram – Special Case

4.2.4 Bus Error and Halt Operation

In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided. External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error signal is received, the processor has two options: initiate a bus error exception sequence or try running the bus cycle again.

4.2.4.1 Bus Error Operation

When the bus error signal is asserted, the current bus cycle is terminated. If $\overline{BERR}$ is asserted before the falling edge of S2, $\overline{AS}$ will be negated in S7 in either a read or write cycle. As long as $\overline{BERR}$ remains asserted, the data and address buses will be in the high-impedance state. When $\overline{BERR}$ is negated, the processor will begin stacking for exception processing. Figure 4.18 is a timing diagram for the exception sequence. The sequence is composed of the following elements:

1. stacking the program counter and status register
2. stacking the error information
3. reading the bus error vector table entry
4. executing the bus error handler routine

The stacking of the program counter and status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address $000008. The core processor loads the new program counter from this location. A software bus error handler routine is then executed by the core processor. Refer to "5.2 EXCEPTION PROCESSING" for additional information.

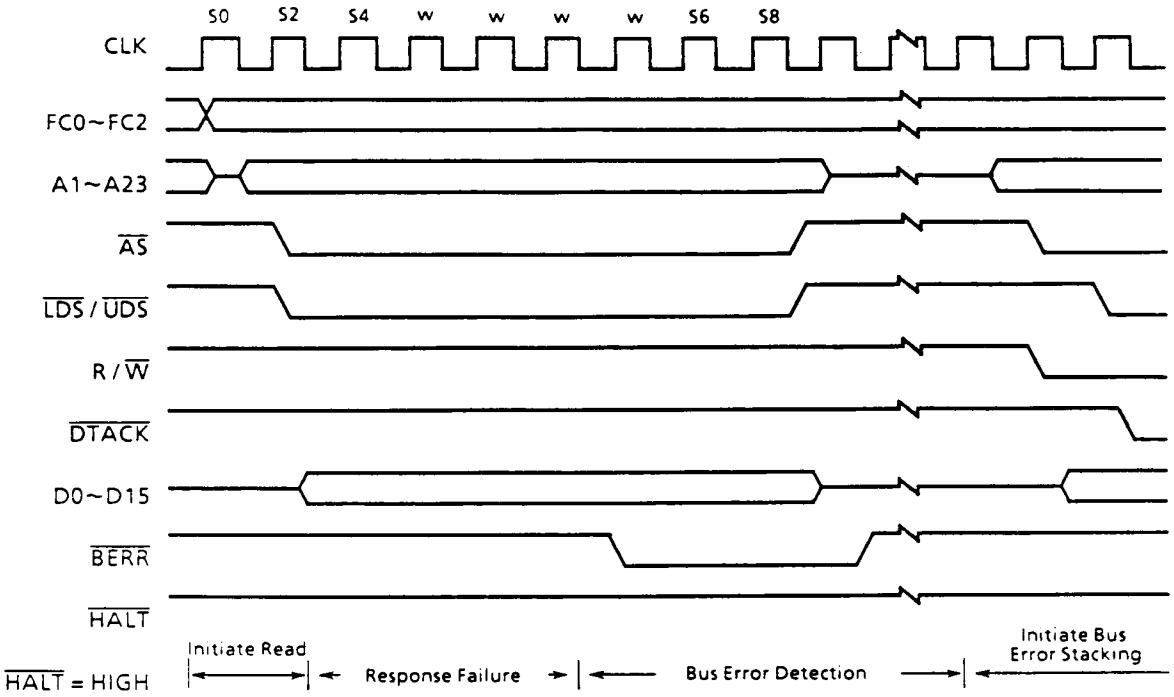


Figure 4.18 Bus Error Timing Diagram

4.2.4.2 Re-Run Operation

When, during a bus cycle, the core processor receives a bus error signal and the halt pin is being driven by an external device, the core processor enters the re-run sequence. Figure 4.19 is a timing diagram for re-running the bus cycle.

The core processor terminates the bus cycle, then puts the address and data output lines in the high-impedance state. The core processor remains "halted", and will not run another bus cycle until the halt signal is removed by external logic. Then the core processor will re-run the previous cycle using the same function codes, the same data (for a write operation) , and the same controls. The bus error signal should be removed at least one clock cycle before the halt signal is removed.

Note : The core processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a test-and-set operation is performed without ever releasing $\overline{AS}$. If $\overline{BERR}$ and $\overline{HALT}$ are asserted during a read-modify-write bus cycle, a bus error operation results.

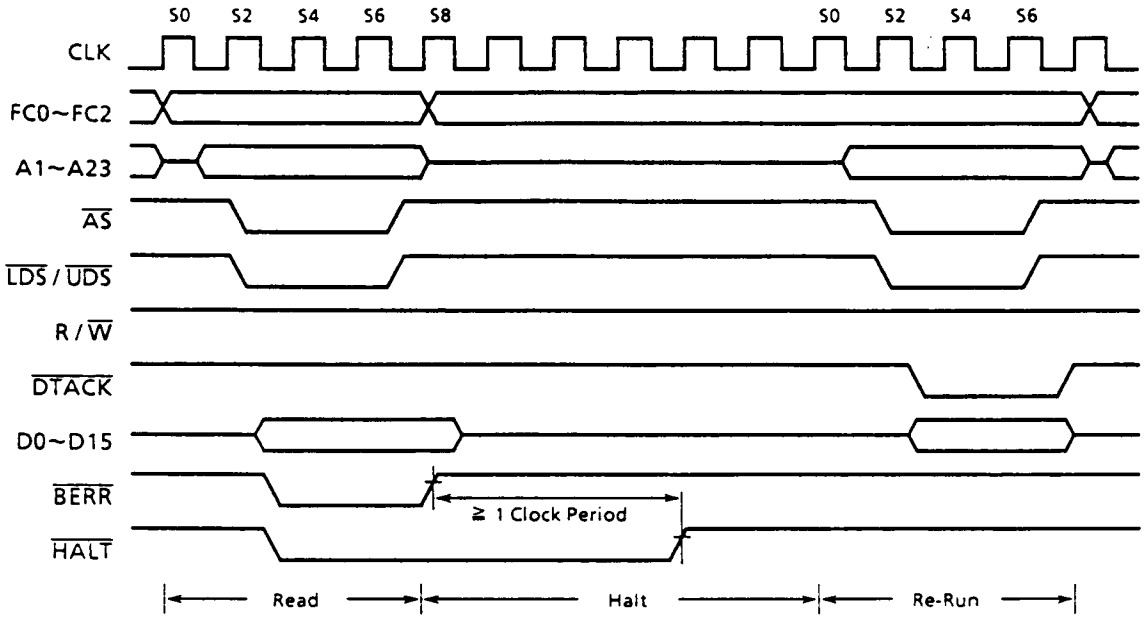


Figure 4.19 Re-Run Bus Cycle Timing Diagram

4.2.4.3 Halt Operation

The halt input signal to the TMP68303 performs a halt/run/single-step function in a similar fashion to the 6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the core processor "halts" (does nothing) and when the halt signal is constantly inactive the core processor "runs" (does something) .

This single-step mode is derived from correctly timed transitions on the halt signal input. It forces the core processor to execute a single bus cycle by entering the run mode until the core processor starts a bus cycle then changing to the halt mode. Thus, the single-step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 4.20 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single-cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

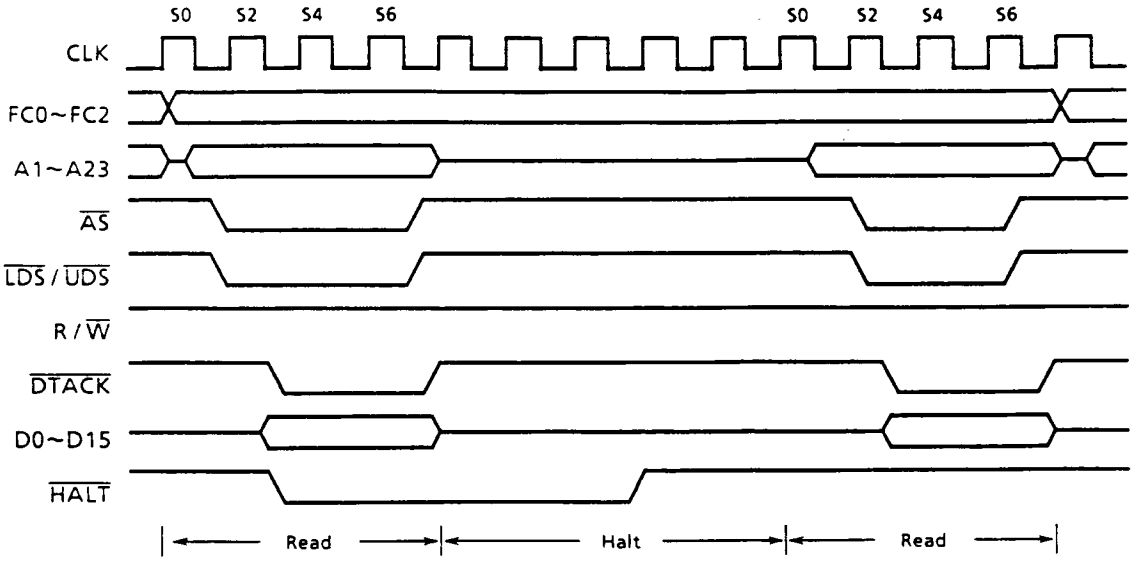


Figure 4.20 Halt Processor Timing Diagram

When the core processor completes a bus cycle after recognizing that the halt signal is active, most three-state signals are put in the high-impedance state, these include:

1. address line
2. data lines

This is required for correct performance of the re-run bus cycle operation.

While the core processor is honoring the halt request, bus arbitration performs as usual. That is, halting has no effect on bus arbitration. It is the bus arbitration function that removes the control signals from the bus.

The halt function and the hardware trace capability allow the hardware debugger to trace single bus cycles or single instructions at a time. These processor capabilities, along with a software debugging package, give total debugging flexibility.

4.2.4.4 Double Bus Faults

When a bus error exception occurs, the core processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus error in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs, the core processor will halt. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

Note that a bus cycle which is re-run does not constitute a bus error exception and dose not contribute to a double bus fault. Note also that this means that as long as the external hardware requests it, the core processor will continue to re-run the same bus cycle.

The bus error pin also has an effect on processor operation after the core processor receives an external reset input. The core processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed) , the core processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

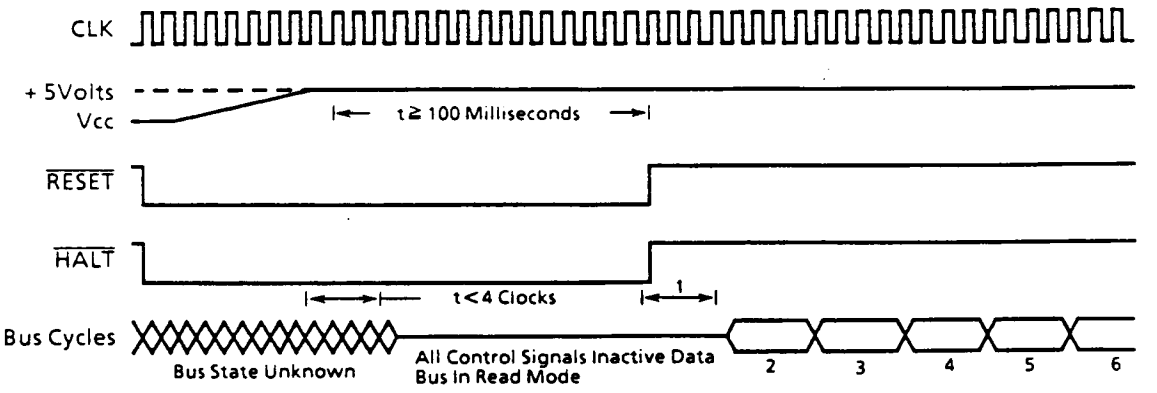
4.2.5 Reset Operation

The reset signal is a bidirectional signal that allows either the core processor or an external signal to reset the system. Figure 4.21 is a timing diagram for the reset operation. Both the halt and reset lines must be asserted to ensure total reset of the core processor.

When the reset and halt lines are driven by an external device, it is recognized as an entire system reset, including the core processor. The core processor responds by reading the reset vector table entry (vector number zero, address $000000) and loads it into the supervisor stack pointer (SSP). Vector table entry number one at address $000004 is read next and loaded into the program counter. The core processor initializes the status register to an interrupt level of seven. No other registers are affected by the reset sequence.

When a reset instruction is executed, the core processor drives the reset pin for 124 clock periods. In this case, the core processor is trying to reset the rest of the system. Therefore, there is no effect on the internal state of the core processor. All of the core processor's internal registers and the status register are unaffected by the execution of a reset instruction. All external devices connected to the reset line will be reset at the completion of the reset instruction.

Asserting the reset and halt lines for 12 clock cycles will cause a processor reset, except when Vcc is initially applied to the core processor. In this case, an external reset must be applied for at least 100 milliseconds.



Notes :

(1) Internal start-up time
(2) SSP High read in here
(3) SSP Low read in here
(4) PC High read in here
(5) PC Low read in here
(6) First instruction fetched here

Figure 4.21 Reset Operation Timing Diagram

4.3 THE RELATIONSHIP OF $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, AND $\overline{\text{HALT}}$

In order to properly control termination of a bus cycle for a re-run or a bus error condition, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ should be asserted and negated on the rising edge of the TMP68303 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the TMP68303. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 4.4) :

Normal Termination : $\overline{\text{DTACK}}$ occurs first (case 1) .

Halt Termination : $\overline{\text{HALT}}$ is asserted at the same time or before $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ remains negated (cases 2 and 3) .

Bus Error Termination : $\overline{\text{BERR}}$ is asserted in lieu of, at the same time, or before $\overline{\text{DTACK}}$ (case 4) ; $\overline{\text{BERR}}$ is negated at the same time or after $\overline{\text{DTACK}}$.

Re-Run Termination : $\overline{HALT}$ and $\overline{BERR}$ are asserted in lieu of, at the same time, or before $\overline{DTACK}$ (cases 6 and 7) ; $\overline{HALT}$ must be held at least one cycle after $\overline{BERR}$. Case 5 indicates $\overline{BERR}$ may precede $\overline{HALT}$ which allows fully asynchronous assertion.

Table 4.4 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 4.5 ($\overline{DTACK}$ is assumed to be negated normally in all cases; for best results, both $\overline{DTACK}$ and $\overline{BERR}$ should be negated when address strobe is negated) .

Table 4.4 $\overline{DTACK}$, $\overline{BERR}$, and $\overline{HALT}$ Assertion Results

Case No.	control Signal	Asserted on Rising Edge State		Result
		N	N + 2	
1	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	A NA NA	S X X	Normal cycle terminate and continue
2	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	A NA A	S X S	Normal cycle terminate and halt. Continue when $\overline{HALT}$ removed.
3	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	NA NA A	A NA S	Normal cycle terminate and halt. Continue when $\overline{HALT}$ removed.
4	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	X A NA	X S NA	Terminate and take bus error trap.
5	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	NA A NA	X S A	Terminate and re-run.
6	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	X A A	X S S	Terminate and re-run when $\overline{HALT}$ removed.
7	$\overline{DTACK}$ $\overline{BERR}$ $\overline{HALT}$	NA NA A	X A S	Terminate and re-run when $\overline{HALT}$ removed.

Legend :

N : the number of the current even bus state (e.g., S4, S6, etc.)

A : signal is asserted in this bus state

NA : signal is not asserted in this state

X : don't care

S : signal was asserted in previous state and remains asserted in this state

Table 4.5 $\overline{BERR}$ and $\overline{HALT}$ Negation Results

Conditions of Termination in Table 4.4	Control Signal	Negated on Rising Edge of State			Results — Next Cycle
		N		N + 2	
Bus Error	$\overline{BERR}$ $\overline{HALT}$	● ●	or or	● ●	Takes bus error trap.
Re-run	$\overline{BERR}$ $\overline{HALT}$	● ●	or	●	Illegal sequence; usually traps to vector number 0.
Re-run	$\overline{BERR}$ $\overline{HALT}$	●		● ●	Re-runs the bus cycle.
Normal	$\overline{BERR}$ $\overline{HALT}$	● ●	or	●	May lengthen next cycle.
Normal	$\overline{BERR}$ $\overline{HALT}$	●	or	● none	If next cycle is started it will be terminated as a bus error.

● : Signal is negated in this bus state.

EXAMPLE A : A system uses a watch-dog timer to terminate accesses to unpopulated address space. The timer asserts $\overline{DTACK}$ and $\overline{BERR}$ simultaneously after time out (case 4).

EXAMPLE B : A system uses error detection on RAM contents. Designer may

(a) delay $\overline{DTACK}$ until data verified and return $\overline{BERR}$ and $\overline{HALT}$ simultaneously to re-run error cycle (case 6), or if valid, return $\overline{DTACK}$ (case 1)

(b) delay $\overline{DTACK}$ until data verified and return $\overline{BERR}$ at same time as $\overline{DTACK}$ if data in error (case 4).

4.4 ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION

4.4.1 Asynchronous Operation

To achieve clock frequency independence at a system level, the TMP68303 can be used in an asynchronous manner. This entails using only the bus handshake lines ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$, $\overline{BERR}$ and $\overline{HALT}$) to control the data transfer. Using this method, $\overline{AS}$ signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal ($\overline{DTACK}$) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{BERR}$, or $\overline{BERR}$ and $\overline{HALT}$, signal to abort or rerun the bus cycle.

The $\overline{DTACK}$ signal is allowed to be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{DTACK}$ may precede data is given as parameter #31 and it must be met in any asynchronous system to insure that valid data is latched into the core processor. Notice that there is no maximum time specified from the assertion of $\overline{AS}$ to the assertion of $\overline{DTACK}$. This is because the MPU will insert wait cycles of one clock period each until $\overline{DTACK}$ is recognized.

4.4.2 Synchronous Operation

To allow for those systems which use the system clock as a signal to generate $\overline{DTACK}$ and other asynchronous inputs, the asynchronous input setup time is given as parameter #47. If this setup is met on an input, such as $\overline{DTACK}$, the core processor is guaranteed to recognize that signal on the next falling edge of the system clock. However, the converse is not true - if the input signal does not meet the setup time it is not guaranteed not to be recognized. In addition, if $\overline{DTACK}$ is recognized on a falling edge, valid data will be latched into the core processor (on a read cycle) on the next falling edge provided that the data meets the setup time given as parameter #27. Given this, parameter #31 may be ignored. Note that if $\overline{DTACK}$ is asserted, with the required setup time, before the falling edge of S4, no wait states will be incurred and the bus cycle will run at its maximum speed of four clock periods.

Note : During an active bus cycle, $\overline{BERR}$ is sampled on every falling edge of the clock starting with S2. $\overline{DTACK}$ is sampled on every falling edge of the clock starting with S4 and data is latched on the falling edge of S6 during a read. The bus cycle will then be terminated in S7 except when $\overline{BERR}$ is asserted in the absence of $\overline{DTACK}$, in which case it will terminate one clock cycle later in S9, $\overline{VPA}$ is sampled only on the third falling edge of the system clock before the rising edge of the E clock.

5. PROCESSING STATES

This section describes the actions of the TMP68303 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the core processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the core processor on exception conditions are detailed.

The TMP68303 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the core processor enters when a stop instruction is executed. In this state, no further references are made.

The exception processing state is associated with interrupts, trap instructions, tracing, and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the core processor may handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the core processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

5.1 PRIVILEGE STATES

The core processor operates in one of two states of privilege: the "supervisor" state or the "user" state. The privilege state determines which operations are legal, are used to choose between the supervisor stack pointer and the user stack pointer in instruction references, and may by used by an external memory management device to control and translate accesses.

The privilege state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas, and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides sequrity by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user state programs.

5.1.1 Supervisor State

The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S bit of the status register; if the S bit is asserted (high), the core processor is in the supervisor state. All instructions can be executed in the supervisor state. The bue cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the core processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the setting of the S bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

5.1.2 User State

The user state is the lower state of privilege. For instruction execution, the user state is determined by the S bit of the status register; if the S bit is negated (low), the core processor is executing instructions in the user state.

Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the stop instruction or the reset instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole state register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are also privileged. The bus cycles generated by an instruction executed in the user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the core processor is in the user privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly, access the user stack pointer.

5.1.3 Privilege State Changes

Once the core processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current setting of the S bit of the status register is saved and the S bit is asserted, putting the core processor in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the core processor is in the supervisor privilege state.

5.1.4 Reference Classification

When the core processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor state, such as interrput acknowledge. Table 5.1 lists the classification of references.

Table 5.1 Bus Cycle Classification

Function Code Output			Reference Class
FC2	FC1	FC0	
L	L	L	(Unassigned)
L	L	H	User Data
L	H	L	User Program
L	H	H	(Unassigned)
H	L	L	(Unassigned)
H	L	H	Supervisor Data
H	H	L	Supervisor Program
H	H	H	Interrupt Acknowledge

Note: L : LOW H : HIGH

5.2 EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made and the status register is set for exception processing. In the second step the exception vector is determined and the third step is the saving of the current processor context. In the fourth step a new context is obtained and the core processor switches to instruction processing.

5.2.1 Exception Vectors

Exception vectors are memory locations from which the core processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 5.1), except for the reset vector which is four words. All exception vectors lie 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 which, when multiplied by four, gives the address of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 5.2) to the core processor on data bus lines D0~D7. The core processor translates the vector number into a full 32-but address, shown in Figure 5.3. The memory layout for exception vectors is given in Table 5.2.

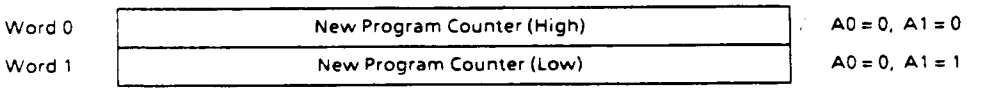


Figure 5.1 Format of Vector Table Entries

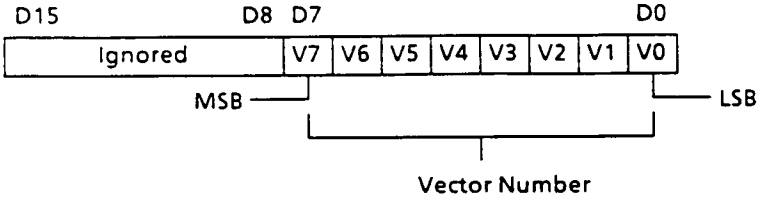


Figure 5.2 Vector Number Format

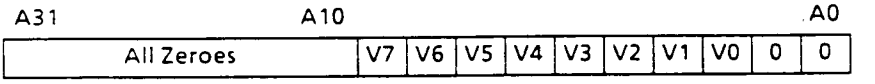


Figure 5.3 Exception Vector Address Calculation

As shown in Table 5.2, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

Table 5.2 Exception Vector Table

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset:Initial SSP
—	4	004	SP	Reset:Initial PC
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*	48	030	SD	(Unassigned, Reserved)
13*	52	034	SD	(Unassigned, Reserved)
14*	56	038	SD	(Unassigned, Reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16~23*	64 95	040 05F	SD	(Unassigned, Reserved) –
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32~47	128 191	080 0BF	SD	TRAP Instruction Vectors –
48~63*	192 255	0C0 0FF	SD	(Unassigned, Reserved) –
64~255	256 1023	100 3FF	SD	User Interrupt Vectors –

* Vector numbers 12, 13, 14, 16~23, and 48~63 are re-served for future enhancements. No user peripheral devices should be assigned these numbers.

5.2.2 Kinds of Exceptions

Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts and the bus error and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address errors or tracing. The trap (TRAP), trap on overflow (TRAPV), check data register against upper bounds (CHK), and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses, and privilege violations cause exceptions. Tracing behaves like a very high-priority internally-generated interrupt after each instruction execution.

5.2.3 Exception Processing Sequence

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S bit is asserted, putting the core processor into the supervisor privilege state. Also, the T bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch and classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer as shown in Figure 5.4. The program counter value stacked usually points to the next unexecuted instruction; however, for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The core processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

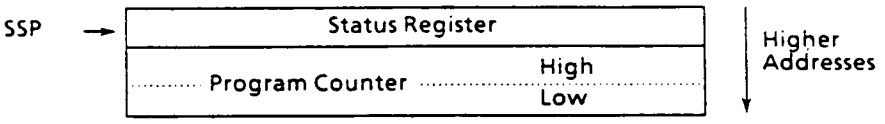


Figure 5.4 Exception Stack Order
(Groups 1 and 2)

5.2.4 Multiple Exceptions

These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted and the exception processing to commence within two clock cycles.

The group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instruction. These exceptions allow the current instruction to execute to completion, but pre-empt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The group 2 exception occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while group 2 exceptions have lowest 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, there is no priority relation within group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Thesefore, 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 is asserted, the trace exception has priority, and processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 5.3.

Table 5.3 Exception Grouping and Priority

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

5.3 EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources and each exception has processing which is peculiar to it. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

5.3.1 Reset

The reset input provides the highest exception level. The processing of the $\overline{\text{RESET}}$ signal is designed for system initiation and recovery from catastrophic failure. Any processing in progress at the time of the $\overline{\text{RESET}}$ is adorted and cannot be recoverd. The core processor is forced into the supervisor state and the trace state is forced off. The core processor interrupt priority mask is set at level seven. 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 supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, 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 progrom counter. The power-up/restart code should be pointed to by the initial program counter.

The reset instruction does not cause loading of the $\overline{\text{RESET}}$ vector, but does assert the reset line to reset external devices. This allows the software reset the system to a known state and then continue processing with the next instruction.

5.3.2 Interrupts

All interrupt requests are transmitted to the core processor through the interrupt controller. There are 7 interrupt priority levels numbered from 0 to 7, with number 7 being the highest level. There is a 3-bit mask field in the status register which indicates the current priority level. All interrupts below that priority level are disabled.

There are both external interrupt requests and requests from built-in devices. The interrupt level can be set by the interrupt controller for each interrupt request. Level 0 indicates "no interrupt requests". The interrupt controller sends the interrupt request with the highest priority level to the core processor. The core processor does not process the interrupt request immediately, but leaves it pending. Pending interrupts are sensed between instructions. If the priority of a pending interrupt is lower than the current interrupt priority level, the core processor proceeds to the next instruction and delays exception processing of the interrupt until later. (The situation is slightly different for the sensing of level 7 interrupts, which are explained next.)

If the priority level of a pending interrupt request is higher than the current interrupt priority level, the exception processing sequence is started. First, the contents of the status register are temporarily stored, the supervisor state is set, trace is disabled, and the priority level is set to the level being acknowledged by the core processor.

The core processor fetches a vector number from the interrupt controller or external device generating the interrupt. If a bus error is asserted during interrupt acknowledge, the interrupt is considered to be "spurious" and a spurious interrupt vector number is generated.

Next, the core processor advances to normal exception processing and stores the PC value and status register value to the stack. The stored PC value is the address of the instruction that should have been executed if the interrupt was not applied. The contents of the exception vector are loaded to the PC and normal instruction execution is started by the interrupt processing routine.

Figure 5.5 shows the interrupt acknowledge flowchart, Figure 5.6 shows the timing and Figure 5.7 shows the interrupt processing timing sequence.

Priority level 7 is a special case. Level 7 interrupts cannot be disabled by masks; therefore, they are called "non-maskable interrupts". An interrupt is generated each time a change is made from a lower level with an interrupt request to level 7. Note that, with level 7 interrupts, there is possibility of an interrupt being generated due to a level comparison when the request level is 7 and the core processor priority level is then set to a lower level by instruction.

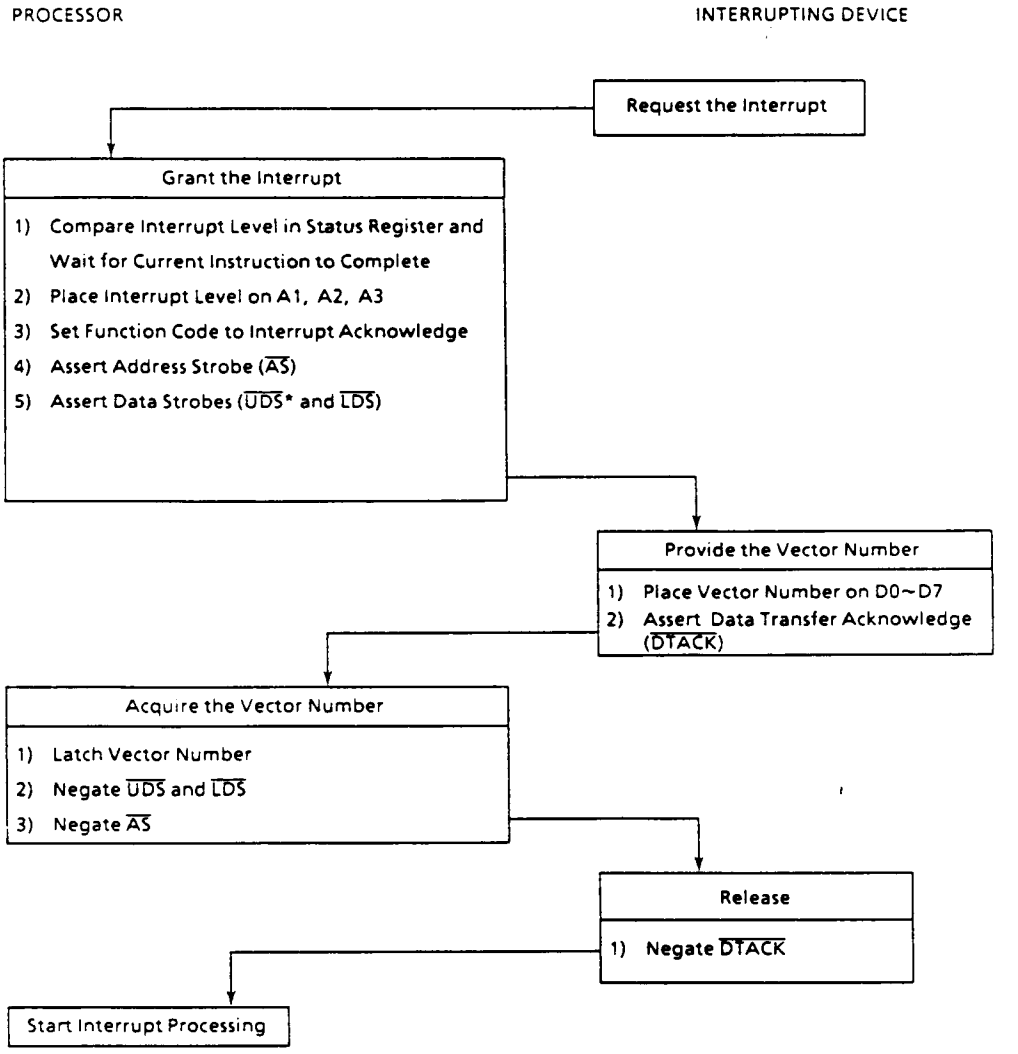


Figure 5.5 Vector Acquisition Flowchart

*: Although a vector 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 D8~D15 at this time.

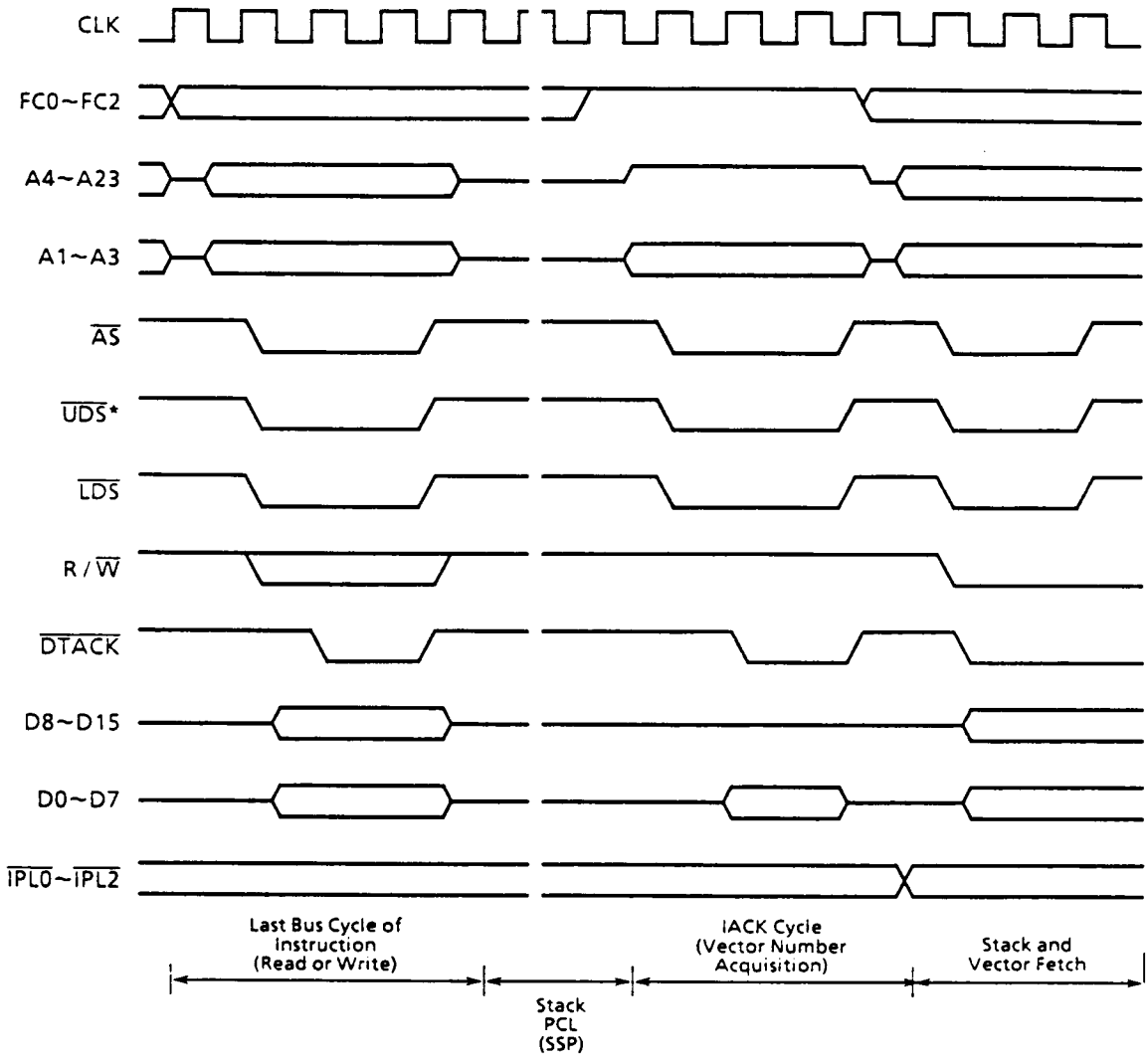
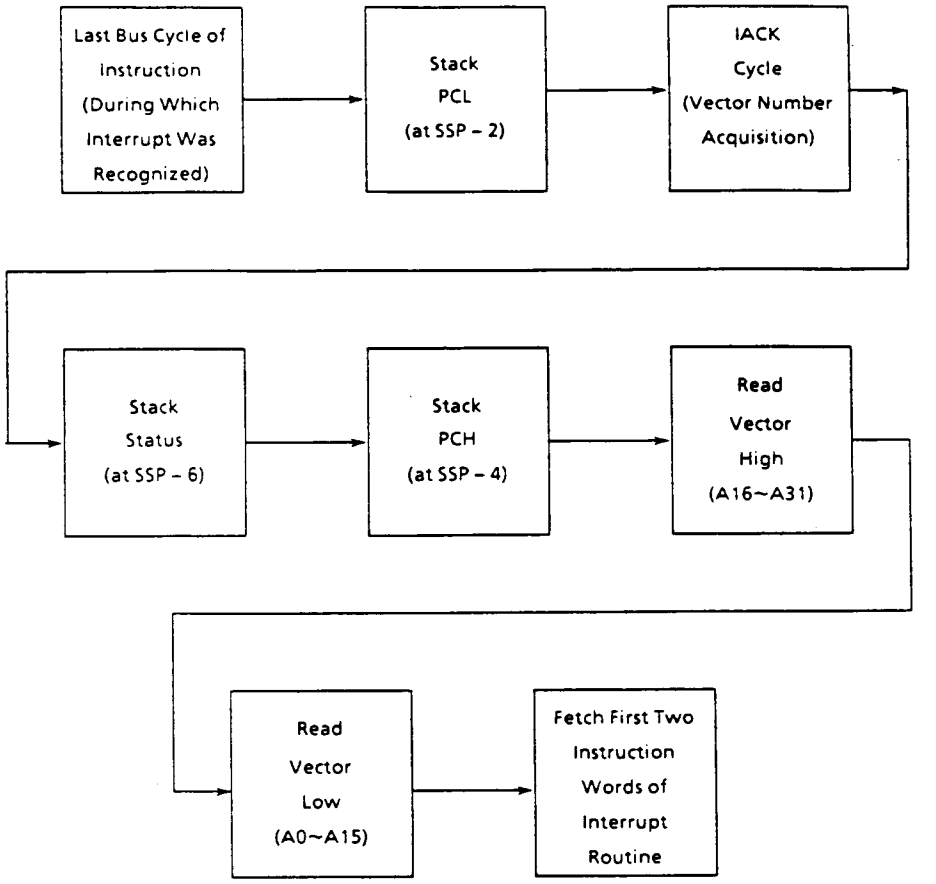


Figure 5.6 Interrupt Acknowledge Cycle Timing Diagram

* : 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 D8~D15 at this time.



Note : SSP refers to the value of the supervisor stack pointer before the interrupt occurs.

Figure 5.7 Interrupt Processing Sequence

5.3.3 Uninitialized Interrupt

An interrupting device asserts provides an interrupt during an interrupt acknowledge cycle to the TMP68303. If the vector register has not been initialized, the responding TLCS-68000 Family peripheral will provide vector 15, the uninitialized, interrupt vector. This provides a uniform way to recover from a programming error.

5.3.4 Spurious Interrupt

If during the interrupt acknowledge cycle no device responds by asserting $\overline{\text{DTACK}}$ or $\overline{\text{VPA}}$, the bus error line should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

5.3.5 Instruction Traps

Traps are exceptions caused by instruction. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

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 runtime error, which may be anarithemetic overflow or a subscript out of bounds.

The signed divide (DIVS) and unsigned (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

5.3.6 Illegal and Unimplemented Instructions

"Illegal instruction" is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs. Three bit patterns will always force an illegal instruction trap on all TLCS-68000 Family compatible microprocessors. They are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for the system. The third pattern, $4AFC, is reserved for customer use.

Word patterns with bits 15~12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

5.3.7 Privilege Violations

In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an

exception. The privileged instructions are:

STOP	AND Immediate to SR
RESET	EOR Immediate to SR
RTE	OR Immediate to SR
MOVE to SR	MOVE USP

5.3.8 Tracing

To aid in program development, the TMP68303 includes a facility to allow instruction-by-instruction tracing. In the trace state, after each instruction is executed an exception is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T bit in the supervisor portion of the status register. If the T bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken, or the instruction is illegal or privileged, the trance 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 indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above 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.

5.3.9 Bus Error

Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for the bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, 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 of course saved. The value saved for the program counter is advanced by some amount,

one to five words beyond the address of the first word of the instruction which 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. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved; whether it was a read or a write, whether or not the processor was processing an instruction, and the classification displayed on the 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 5.8 illustrates how this information is organized on the supervisor stack. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in vector number two. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

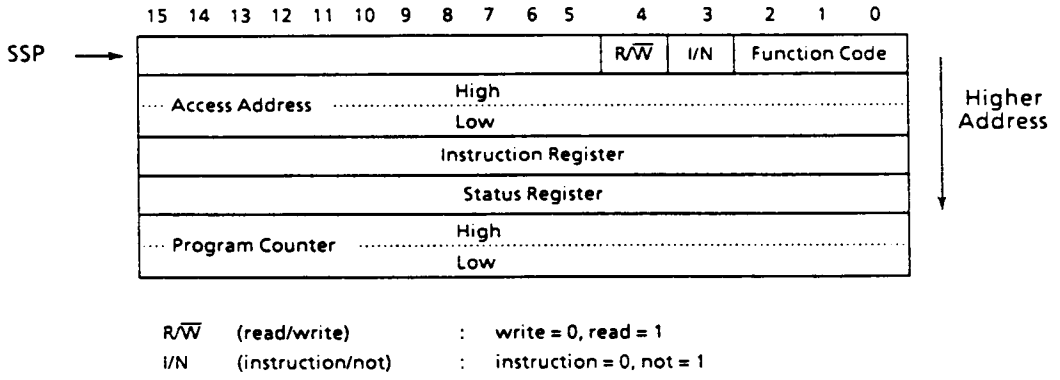


Figure 5.8 Exception Stack Order (Group 0)

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted and all processing ceases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroy any memory contents. Only the $\overline{\text{RESET}}$ pin can restart a halted processor.

5.3.10 Address Error

Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted and the processor ceases whatever processing it is currently doing and begins exception processing. After the exception processing commences, the sequence is the same as that for bus error including the

information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in Figure 5.9, an address error will execute a short bus cycle followed by exception processing.

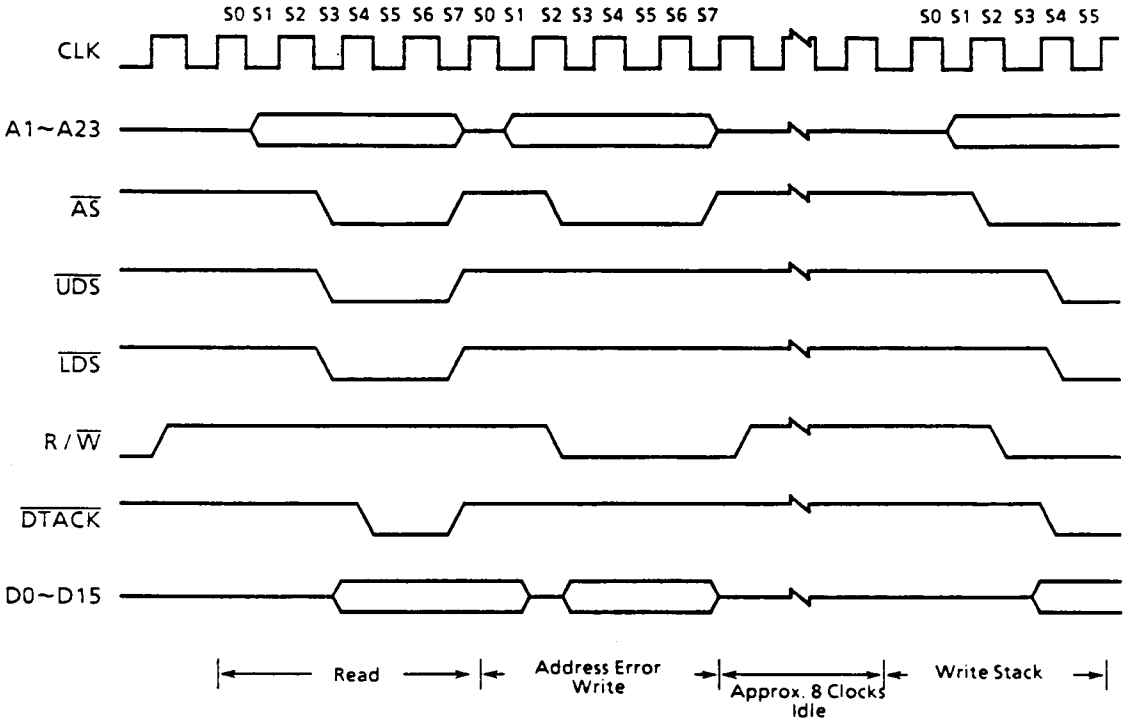


Figure 5.9 Address Error Timing Diagram

6. FUNCTION OF PERIPHERAL DEVICES

6.1 ADDRESS DECODER

6.1.1 Overview

This address decoder can select DRAM memory areas (hereafter called memory area) and register areas for the built-in devices (hereafter called register area). Areas can be located anywhere within the 16M-byte address space of the core processor. The sizes of memory areas are selectable.

6.1.1.1 Feature

Two external CS (Chip Select) pins
Selection of built in devices
Relocation of register area
Setting start address / area size of memory area
Generation of internal $\overline{\text{DTACK}}$ (setting waits of 0~7 clocks)
Detectation of bus errors ($\overline{\text{BERR}}$)

6.1.2 Area Selection

For each bus cycle, the address decoder decodes the address and determines whether two memory area or register area is being accessed.

If a memory area is being accessed, the chip select signal ($\overline{\text{CSn}}$) is asserted to the external, and assert the $\overline{\text{DTACK}}$ with setting waits to core processor. However chip select signal generation prohibit, when enable the DRAM CONTROLLER (DRAMC). $\overline{\text{CS0}}$ of internal signal is used for DRAM in this time.

If a register area is being accessed, a built-in device is selected and a data transfer is performed. Valid signals are output to the external pins at this time in the same way as when another area is accessed. During the read cycle, however, external data are ignored and data from the built-in device are read.

The start address of each area is specified by the memory address register or relocation register. If more than one area has been assigned to the same address, the area is selected in accordance with the following priority sequence.

Area	Priority sequence
Register area	High
Memory area (CS0)	⋮
Memory area (CS1)	Low

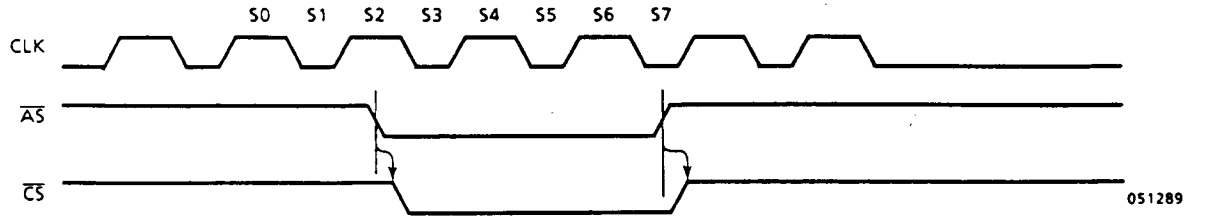


Figure 6.1 $\overline{CS}$ and $\overline{AS}$

6.1.2.1 Memory Area

Two memory area can be located anywhere within the 16M-byte address space. The start address of each memory area is specified by the memory address register.

For each bus cycle, the address decoder compares the values of the addresses on the bus and in the memory address register. If these values match, a memory area is considered to have been accessed. Addresses to be compared are specified by the address mask register and the size of a memory area is determined from the width of the address compared. Area size can be select from within table 6.1.1

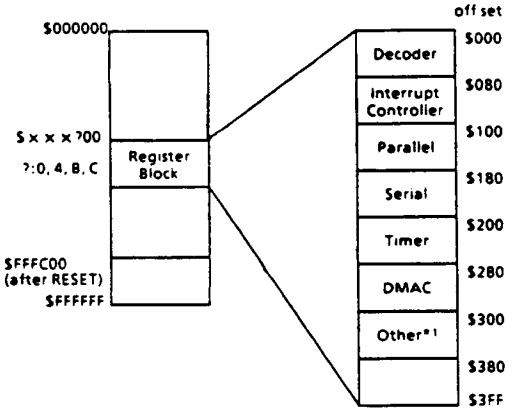
Address decoder can assert the $\overline{DTACK}$ of 0~7 clocks wait by setting control register.

Table 6.1.1 Memory Area and Memory Size

SIZE	256	512	32K	64K	128K	256K	512K	1M	2M	4M	8M
CS0	-	-	○	○	○	○	○	○	○	○	○
CS1	○	○	-	○	○	○	○	○	○	○	-

6.1.2.2 Register Areas

A register area is a collection of built-in device registers and includes the address decoder. The size of the register area is 1k-byte and the start address can be located anywhere in the address space with a 1k-byte boundary by specifying in the relocation register. The registers for the built-in devices are as shown below. After RESET, register area is placed at $FFFC00.



*1 : Other = DRAM Controller and Stepping Motor Controller.

6.1.2.3 Automatic DTACK Generation (Variable Number of Wait States)

The address decoder can generate $\overline{DTACK}$ for each of area.

The address decoder automatically generates $\overline{DTACK}$ with 0 wait states for register areas, * except particular areas. For the memory area, either the $\overline{DTACK}$ generated by an external circuit or $\overline{DTACK}$ generated by the address decoder, or both (the one asserted first), can be selected and used in the IACK cycle (when reading external vector numbers). Wait states of 0 to 7 clocks can be inserted in $\overline{DTACK}$ generated by the address decoder.

(*) Notes : The address decoder random generates $\overline{DTACK}$ within 0~3 wait states when access register for 8-bit timer, port and stepping motor controller

6.1.3 Bus Cycle Monitoring

For bus cycles in which $\overline{DTACK}$ is not returned as when accessing address space for which there is no memory, there is a function which monitors bus cycle length and issues a $\overline{BERR}$ signal. The bus cycle lengths at which a bus error is issued can be set to 32, 64, 128 or 256 clocks.

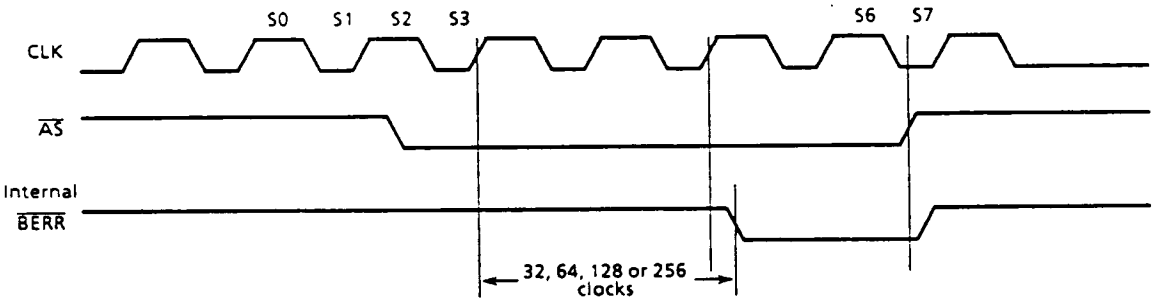


Figure 6.3 Internal $\overline{BERR}$

6.1.4 Memory Area Selection

For each bus cycle, the address decoder compares the values of the addresses on the bus and in the memory address register. In comparing the values, the bit(s) specified by the address mask register can be ignored. When a comparison shows that two values match, the specified area is considered as being accessed. If this area is enabled, generates $\overline{CS}$ signal. If a register area has been assigned to the same address space by the relocation register, that register area will be valid.

6.1.5 Register Configuration

6.1.5.1 Memory Address Register

This register specifies memory area start address. The address which can be specified is A16 - A23. Memory area start address can only be set in accordance with the memory area size boundaries. After RESET, memory address register 0 (use for CS0) becomes 00H, and memory address register 1 (use for CS1) becomes undefined.

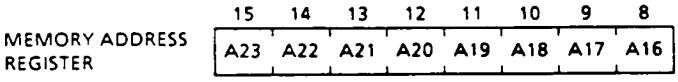


Figure 6.4 Memory Address Register

6.1.5.2 Address Mask Register

This register specifies address from A8 to A22 to be used for comparison. The memory area size varies depending on the width of the address to be compared. The mask addresses for each bit are as follows.

bit	mask address	
	CS0	CS1
0	M15	M8
1	M16	M9-M15
2	M17	M16
3	M18	M17
4	M19	M18
5	M20	M19
6	M21	M20
7	M22	M21

After a reset, address mask register 0 becomes FFH (memory area size is 8M-byte), and address mask register 1 becomes undefined.

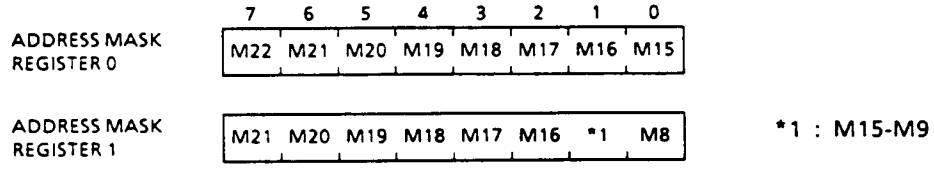


Figure 6.5 Address Mask Register

6.1.5.3 Area Control Register

This register sets area enable (1) / disable (0), the DTACK mode and number of wait states inserted.

When an area is disabled, the internal $\overline{CS}$ signal is not asserted and $\overline{DTACK}$ is not issued from the address decoder, even if the area is accessed.

bits 0 - 2	Specifies the number of wait states.
bit 3	Specifies address decoder $\overline{DTACK}$ enable (1)/disable (0).
bit 4	Specifies external $\overline{DTACK}$ enable (1)/disable (0).
bit 5	Specifies area enable (1)/disable (0).

Whether the external $\overline{DTACK}$ or address decoder $\overline{DTACK}$ is to be used is determined as follows.

Bits 4, 3	Mode
1 1	Use both
1 0	Use external $\overline{DTACK}$
0 1	Uses the address decoder $\overline{DTACK}$.
0 0	Not specified (becomes 01 if specified).

After RESET, area control register 0 (use for $\overline{CS0}$) becomes 3DH, and area control register 1 (use for $\overline{CS1}$) becomes undefined, and area control register 2 (use for Register) becomes 18H.

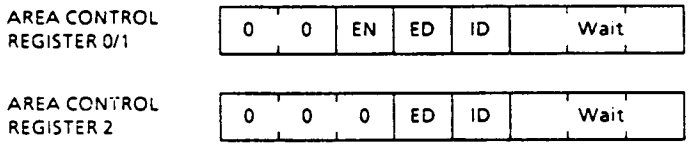


Figure 6.6 Area Control Register

6.1.5.4 Time Out Register

This register specifies the time that elapses before $\overline{BERR}$ is issued. This time can be set to 32, 64, 128 or 256. When more than one time is specified, the longest time will be selected and the other bits will be cleared. When none of these times are selected, $\overline{BERR}$ is not issued.

bit 0 $\overline{BERR}$ is issued after 32 clocks (1: on, 0: off)
bit 1 $\overline{BERR}$ is issued after 64 clocks (1: on, 0: off)
bit 2 $\overline{BERR}$ is issued after 128 clocks (1: on, 0: off)
bit 3 $\overline{BERR}$ is issued after 256 clocks (1: on, 0: off)
After RESET, this register becomes 08H.

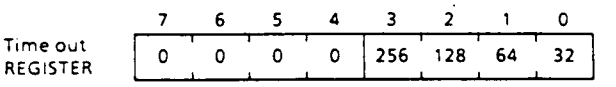


Figure 6.7 Time out Register

6.1.5.5 Relocation Register

This register specifies the start addresses of register blocks. The addresses which can be specified are A10 - A23; therefore, register block start addresses can only be set at 1k-byte boundaries. After RESET, this register becomes FFFCH.

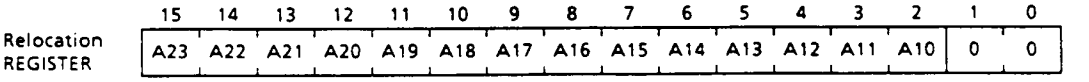


Figure 6.8 Relocation Register

6.1.6 Register Map

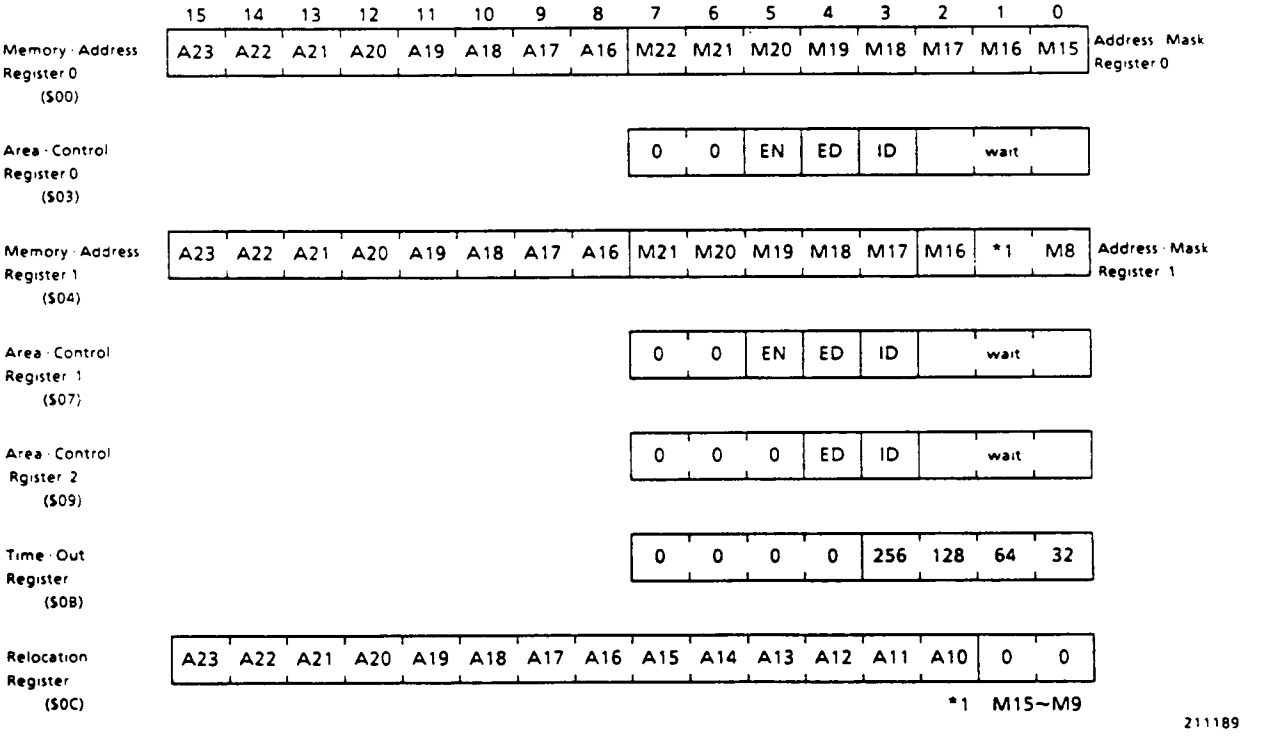


Figure 6.9 Address Decoder Register Map

6.2 INTERRUPT CONTROLLER

6.2.1 Overview

This interrupt controller has 12 interrupt channels. Nine of these channels are for built-in devices and the other three are for external requests. Interrupt levels can be set for each channel, thus, priority levels can be programmable. The interrupt controller issues an IACK signal for each channel during the IACK cycle. Also, vector numbers can be issued for external interrupts, and this feature can substitute the AUTO vector interrupt function.

6.2.1.1 Feature

Interrupt factor : 12ch (external 3ch, internal 9 ch)
Enable setting priority of interrupts
Auto vector generation
Interrupt request pin (Level/edge is programmable)
Interrupt acknowledge signal generation
Enable mask of interrupts
Interrupt monitor (pending, inservice bit)

6.2.2 Interrupt Request

When there is a request to interrupt, the interrupt controller uses internal IPL0 - 2 to issue an interrupt request to the core at the preset level. If requests are issued for more than one channel at the same time, the request with the highest priority level is used.

6.2.2.1 Request Input Mode

The following four input modes are selectable for external interrupt requests.

low level
high level
rising edge
falling edge

For the level modes, the interrupt request must continue until IACK is returned. (External request signals are sampled at the clock falling edges.) For the edge modes the same status must continue for at least 2 clocks after the edge.

(Note) When the mode is changed from the level mode to edge mode, the interrupt request (pending bit) which has already set is not cleared. To clear this bit as follows.
Mask the interrupt request ;
Change from the level to edge mode ;
Clear the pending bit ;
Mask off the interrupt request;

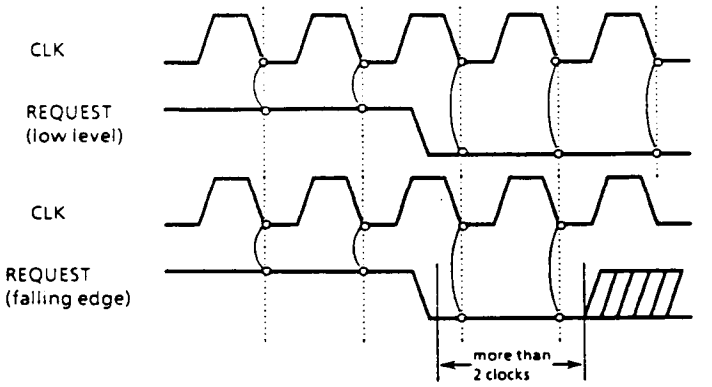


Figure 6.10 Interrupt request

6.2.3 Priority Between Channels

6.2.3.1 Channel Interrupt Request Level

The interrupt controller can set a level for each interrupt channel. The priority order depends on the level set (high 7 - 1 low).

6.2.3.2 Priority for Channels of the Same Level

If more than one channel is set to the same level, the priority order is as follows.

high	External 0
	Timer 1
	Serial 0
	Dmac 0
	External 1
	Timer 2
	Serial 1
	Dmac 1
	Timer 3
	External 2
	Dmac 2
low	Timer 4

6.2.4 IACK Cycle

6.2.4.1 IACK Signal

During the IACK cycle generated by the core, the interrupt controller decodes A1 - 3, detects the interrupt level received and issues the IACK signal for the corresponding channel. If more than one channel issue requests at the same level, the IACK signal is issued for the channel with the highest priority level as described in item 5.3.2. (IACK signal is asserted in the same timing as AS.)

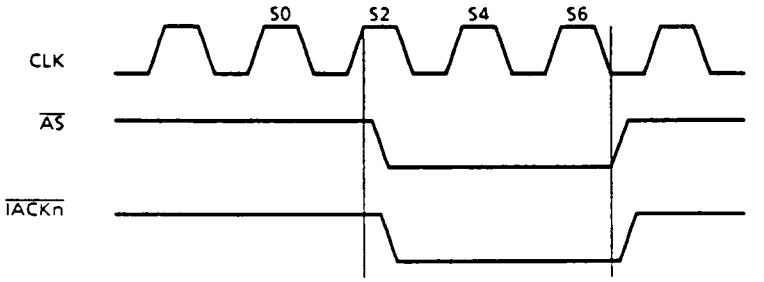


Figure 6.11 $\overline{\text{IACK}}$ signal

6.2.4.2 Vector Number

The interrupt controller generates vector numbers during IACK cycles (instead of auto vector interrupts). The 3 higher order bits of the vector number is set in the vector number register and the 5 lower order bits are determined channel as follows.

ch	vector No.	
external 0	00000	
external 1	00001	
external 2	00010	
timer 1	00100	
timer 2	00101	
timer 3	00110	
timer 4	00111	
serial 0	01000	error
serial 0	01001	buffer full
serial 0	01010	buffer empty
serial 1	01100	error
serial 1	01101	buffer full
serial 1	01110	buffer empty
DMAC 0	10000	error
DMAC 0	10001	service end
DMAC 1	10100	error
DMAC 1	10101	service end
DMAC 2	11000	error
DMAC 2	11001	service end

In care of external interrupts, the vector numbers can also be input externally. (The timing for external input of vector numbers is the same as for the 68000.)

6.2.5 Interrupt Status

The status of the channels is indicated in the mask, pending and in-service registers.

6.2.5.1 Mask Bit (M)

This bit enables the interrupt request.

1 : request will be ignored (masked)
0 : request will be received (not masked)
set : This bit is set by RESET signal or by the software (write "1").
reset : cleared by the software (write "0").

While the mask is applied, all interrupt requests are ignored. Interrupt requests issued while masked are accepted after the mask is reset.

Interrupt Control Register can be rewritten only while masked.

6.2.5.2 Pending Bit (P)

This bit indicates that an interrupt request has been made.

1 : indicates an interrupt request
0 : indicates no interrupt request
set : set by interrupt request
reset : cleared by interrupt request accept;
by the software (writes "0") or by RESET signal.

This bit cannot be set by the software.

For the software to be able to clear this bit during a level mode external interrupt request, the interrupt request must first be disabled.

6.2.5.3 In-service Bit (I)

This bit indicates acceptance of interrupt requests.

1 : interrupt request has been accepted
0 : interrupt request has not been accepted
set : interrupt request accepted
reset : cleared by the software (writes "0") or by RESET signal.

This bit cannot be set by the software.

Interrupt status according to the mask, pending and in-service bit values.

M	P	I	
0	0	0	no request
0	0	1	interrupt processing routine executing
0	1	0	interrupt request accepted
0	1	1	next interrupt request received while the interrupt processing routine is executing
1	0	0	no request
1	0	1	masked while interrupt processing routine executing
1	1	0	interrupt request received on the masked interrupt
1	1	1	interrupt request received on the interrupt masked by interrupt processing routine

M : mask bit
P : pending bit
I : in-service bit

6.2.6 Register Configuration

6.2.6.1 Control Register

6.2.6.1.1 Interrupt Control Registers 0 - 2

These are the interrupt control registers for external interrupt.
Bits 2 - 0 set the interrupt level. Bits 3, 4 set the request mode as shown below.

bit 4 R/F	bit 3 L/E	mode
0	0	falling edge
1	0	rising edge
0	1	Low level
1	1	High level

Bits 5 turns the auto vector function on (1) and off (0). After RESET, the initial value becomes 07H.

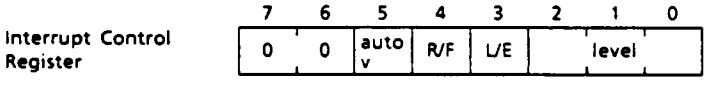


Figure 6.12 Interrupt Control Register (0~2)

6.2.6.1.2 Interrupt Control Registers 3 - 9

These are the interrupt control registers for internal interrupt. Each of the registers indicates the interrupt level for the internal devices (bits 0-2). The corresponding internal devices are as shown below.

REGISTER	DEVICE
3	Serial ch0
4	Serial ch1
5	DMAC ch0
6	DMAC ch1 / Ch2
7	Timer ch1
8	Timer ch2
9	Timer ch3 / Timer ch4

After resetting, the initial value becomes 7H.

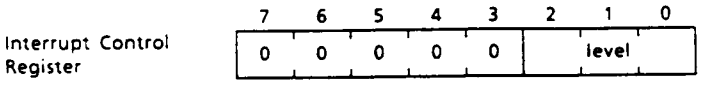


Figure 6.13 Interrupt Control Register (3~9)

6.2.6.2 Mask Register

This register sets masks for every channels. "1" means a mask (interrupts ignored) and "0" means no mask.

After RESET, the initial value becomes 7F37H.

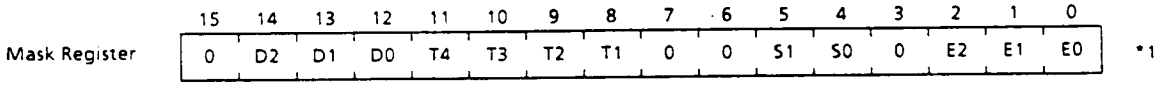


Figure 6.14 Interrupt Mask Register

6.2.6.3 Pending Register

This register indicates whether or not there is an interrupt request which has not yet been accepted by the core processor. "1" indicates that there is an interrupt request which has not been accepted by the core processor and "0" indicates that there is no interrupt request. Each bit is automatically cleared when the core processor accepts the interrupt request. Clearing by program cancels the interrupt request. After RESET, the initial value becomes 0000H.

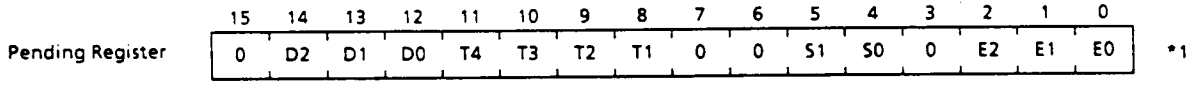


Figure 6.15 Interrupt Pending Register

6.2.6.4 In-service Register

This register indicates whether or not an interrupt request has been accepted by the core processor. "1" indicates that the request has been accepted and "0" indicates that a request has not been accepted.

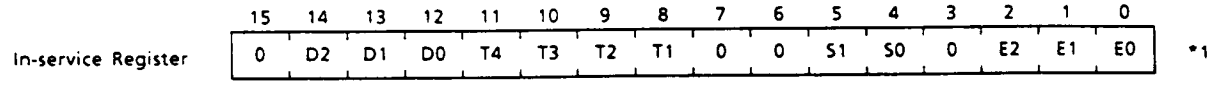


Figure 6.16 Interrupt Inservise Register

6.2.6.5 Vector Number Register

This register specifies the 3 upper order bits of vector numbers. The vector numbers of the 5 lower order bits are predetermined for each interrupt channel. Bits 7 - 5 specify the 3 upper order bits of vector numbers. The other bits are read as "0". After RESET the initial value becomes 00H.

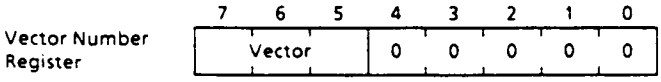


Figure 6.17 Vector Number Register

*1	T4	:	Timer ch4
	T3	:	Timer ch3
	T2	:	Timer ch2
	T1	:	Timer ch1
	S1	:	Serial ch1
	S0	:	Serial ch0
	E2	:	External ch2
	E1	:	External ch1
	E0	:	External ch0
	D2	:	Dmac ch2
	D1	:	Dmac ch1
	D0	:	Dmac ch0

6.2.6.6 Register Map

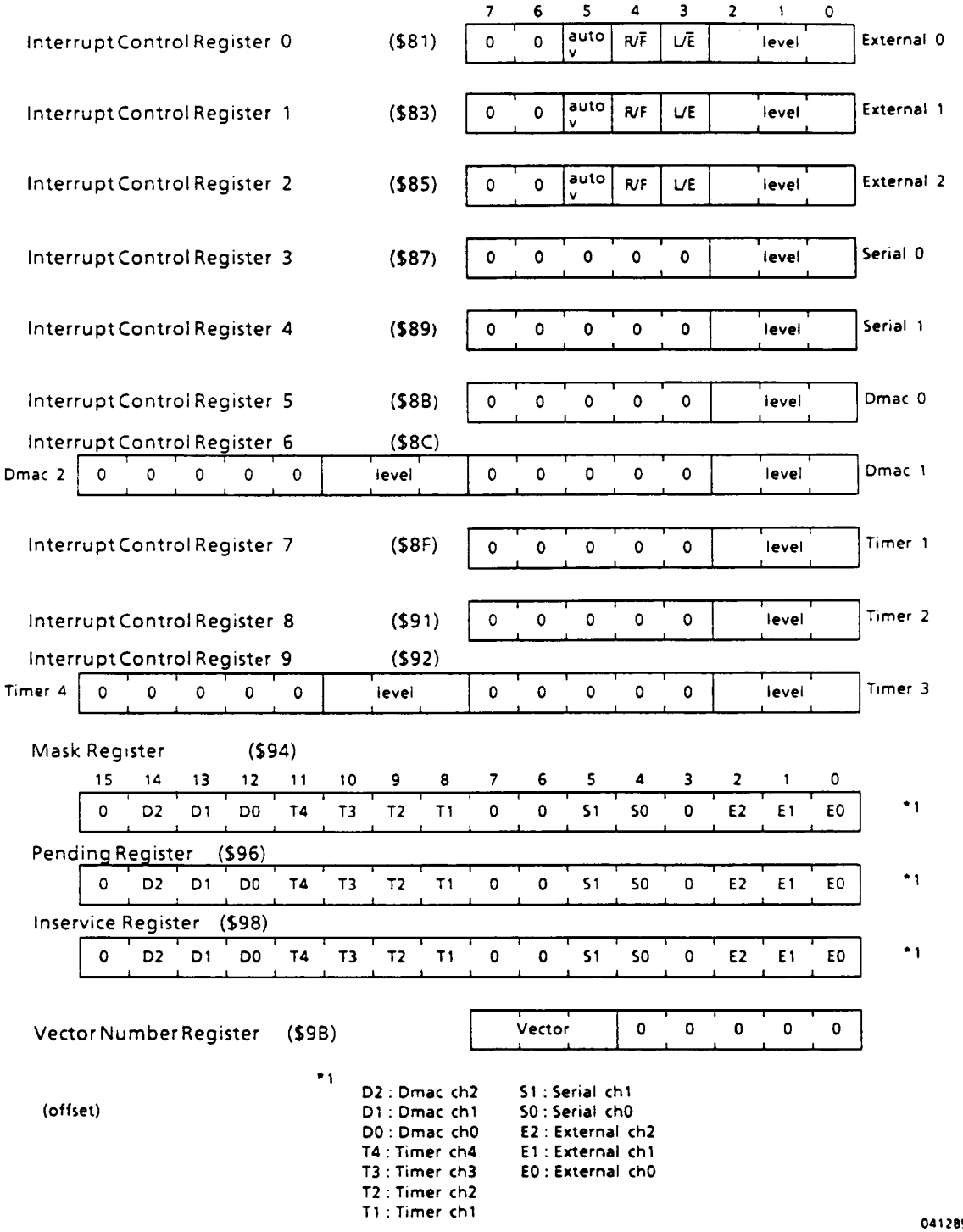


Figure 6.18 Interrupt Controller Register Map

6.3 SERIAL INTERFACE

6.3.1 Overview

This serial interface has a completely independent two-channel configuration and supports asynchronous communications. Interrupt requests can also be generated for each of these two channels.

6.3.1.1 Features

Asynchronous communications by two completely independent channels.
Interrupt requests can be generated for each channel.
5 - 8 bit character lengths are programmable.
Either 1 or 2 stop bits can be selected.
A parity bit can be appended (no parity, even parity, odd parity).
Parity, overrun and framing error detection.
False start bit detection.
Automatic break detection.
Full duplex communication, double buffer system.
Transfer rate: 1Mbps in maximum.
Baud rate generator included.

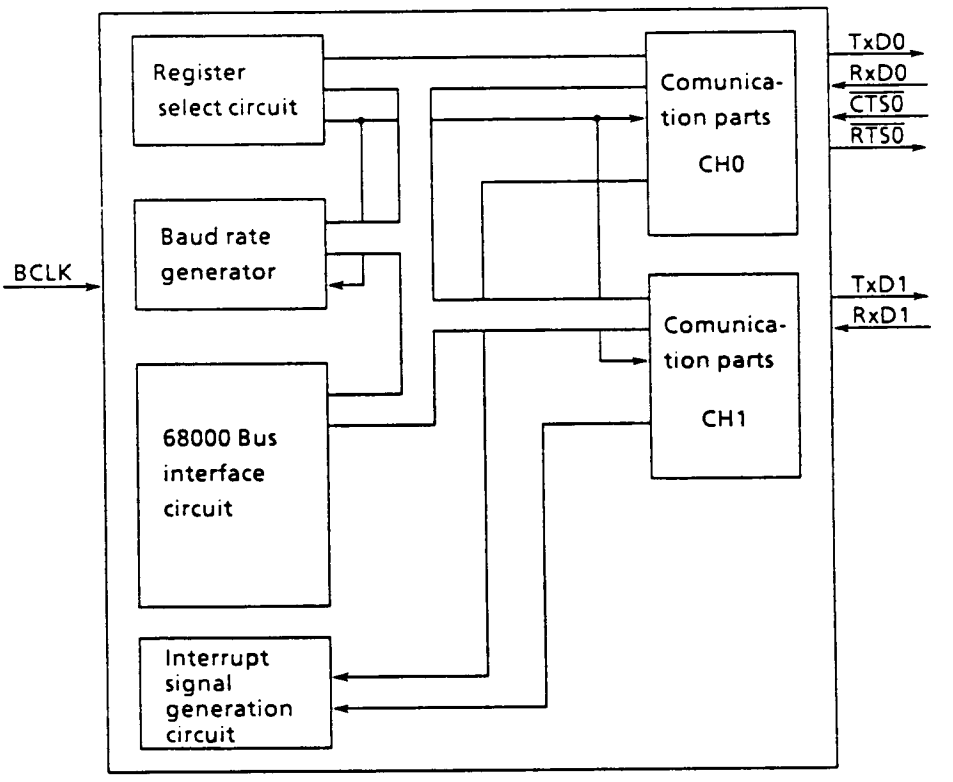


Figure 6.19 Serial Interface Block Diagram

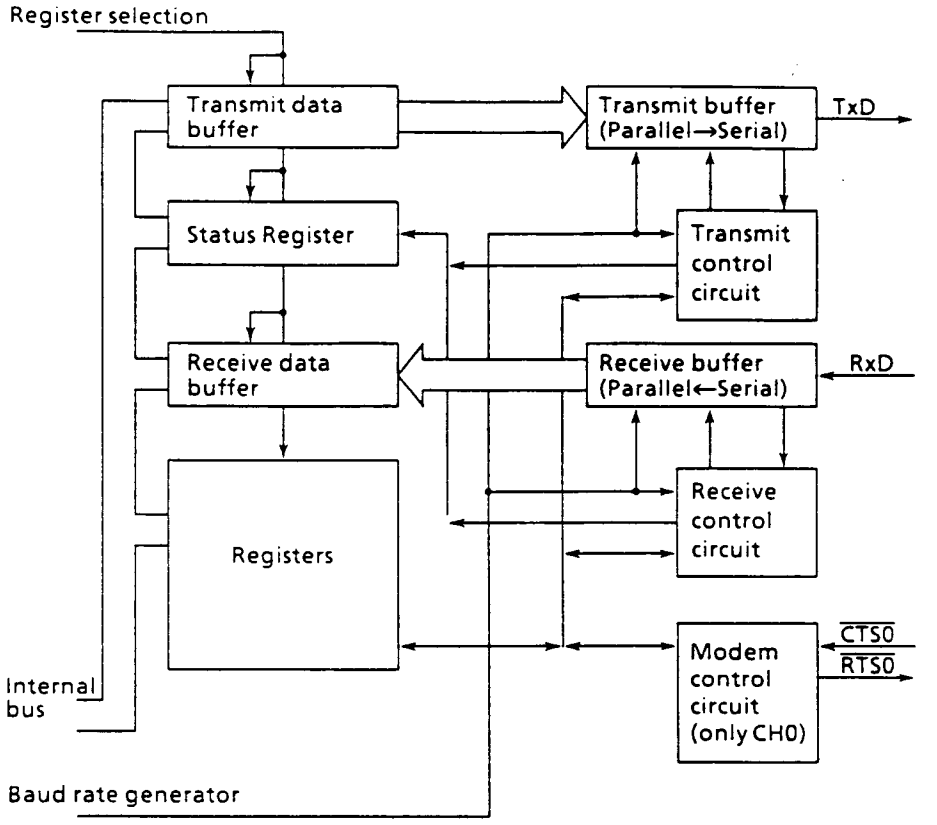


Figure 6.20 Communication Parts Block Diagram

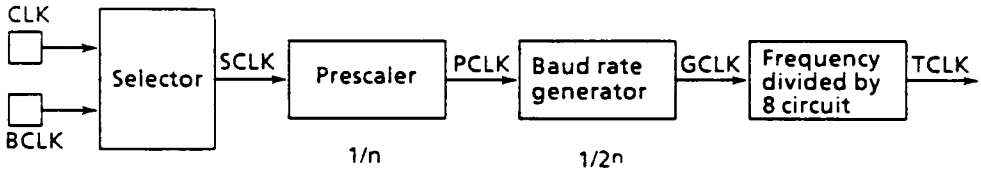


Figure 6.21 Baud Rate Generator Block Diagram

6.3.2 Communications Operation Overview

6.3.2.1 Data Format

When a data character is sent to the transmit data buffer, the serial interface automatically adds one start bit before (lower order first) and the specified number of stop bits after the data bits. When parity (even, odd) is specified by the mode register, a parity bit is inserted before the stop bits.

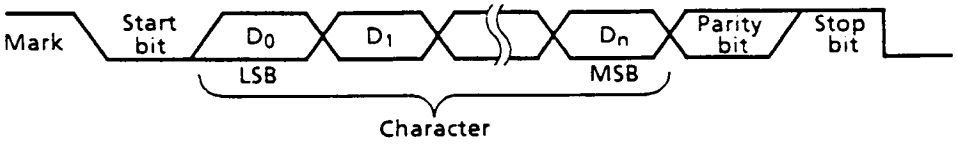


Figure 6.22 Data Frame Sequence

6.3.2.2 Data Sending

When a data character is received, it resets TxE, and checks TxEN, CTS and the transmit flag.

If TxEN = 1, CTS = 0 and the transmit flag is 0, the transmit controller shifts out the data bit at the rising edge of TCLK.

When sending is started and the start bit is sent, the transmit controller sets the transmit flag to 1 and TxRDY = 1 at the same time so that a data character to be sent next can be loaded into the transmit data buffer.

When the data character to be sent next is loaded, TxE is reset to 0, and the data character is held in the transmit data buffer until sending the current data character is completed.

After sending the start bit, the transmit controller automatically appends a parity bit and the stop bits to the data byte of the specified character length and sends this to the TxD line. When the data character to be sent next is in the transmit data buffer, sending continues with the start bit of the next data following the stop bit of the current data.

Once sending is started, the transmit controller continues sending the current data characters until the stop bits have been sent and the transmit flag is reset to 1 regardless of change of CTS level or TxEN level.

6.3.2.3 Data Receiving

The RxD line is normally at high level (mark status). The beginning of a start bit is detected when a falling edge enters the input.

For receiving, the first low level sampled at the GCLK rise is detected and, if the sampled data are low level at the GCLK rise four times in succession, the first low level is considered to be an effective start bit. The center point of the following data bits is then determined and each data bit is sampled at the TCLK falling edge.

When there is a parity bit, the center of the parity bit is sampled and compared to the parity of the received data and, if the two are not equal, the parity error flag is set to "1".

Concerning the stop bits, the center of the first stop bit is sampled regardless of the number of stop bits specified by the program and the framing error flag is set to "1" if this bit is not "1".

When the data bits for the number programmed have been sampled, these are sent together to the receive data buffer and the RxRDY flag is set to "1". Any unused upper order bits remaining at this time are reset to "0" and sent.

The RxRDY flag indicates that there is data to be read out from the receive data buffer. If sampling of the next data character has been completed and the data written before are not read out by the time these data have been sent to the receive data buffer, the contents of the receive data buffer are rewritten, thus erasing the previous data and setting the overrun error flag to "1".

If two or more bytes of data are input with the data bits, the parity bit if necessary, and the stop bits all "0", the receive controller sets the break detect flag to "1". At this time, the RxD initialize circuit operates to hold detection of the next start bit until the RxD line becomes "1". This "1" is held until all of the error flags, including the break detect flag, are reset by the error reset (ERS) command. However, the occurrence of an error will not affect the receive operation.

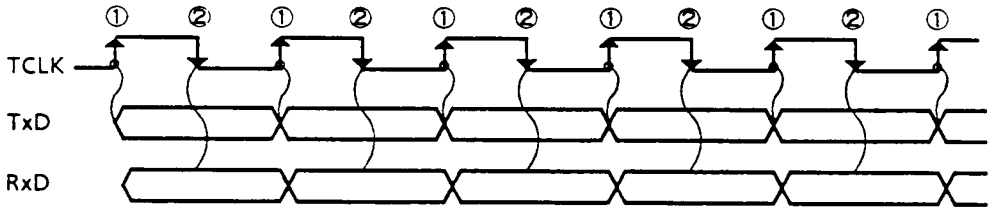


Figure 6.23 Data Transmit/Receive Timing Chart

6.3.3 Interrupt Control

The serial interface is interrupted for any of the following three reasons.

(1) When the transmit data buffer is ready to receive data.

(2) When data received in the receive channel are in the receive data buffer or a break character is detected.

(3) When an error occurs in the receive channel.

Each of these causes can be masked with the mode register. The condition for these reasons is RxEN for the receive side and TxEN for the transmit side.

In addition to individual masking of these causes, the occurrence of interrupt signals in the serial interface can be enabled and disabled with the control register.

$$INT = (INTM = 0) \times \{(RINT) + (TINT)\}$$
$$TINT = (TxEN = 1) \times (\overline{CTS} = 0) \times (TxRDY = 1) \times (TxINTM = 0)$$
$$RINT = (RxEN = 1)[(RxINTM = 0) \times \{(RxRDY = 1) + (RBRK = 1)\} + (ERINTM = 0) \times (FE + OE + PER)]$$

Interrupt Priority in the same channel is as follows :

High	Receive error
·	Receive buffer full
Low	Transmit buffer empty

6.3.4 Baud Rates

The serial interface determines the baud rate by frequency dividing the system clock or BCLK clock using an 8-bit prescaler and 8-bit baud rate generator in the baud rate generator block.

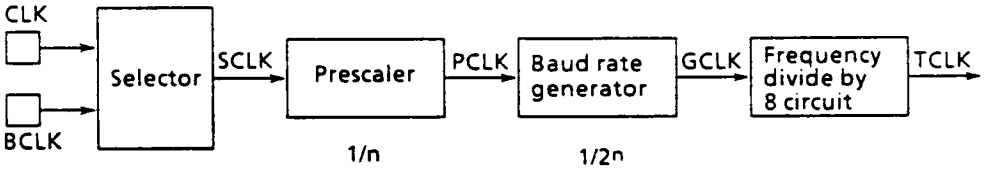


Figure 6.24 Baud Rate Generator Block

First, the selector is used to select either the system clock or the BCLK clock. If selected, SCLK is frequency divided by 1 - 1/256 by the prescaler.

The resulting PCLK is then frequency divided by 1 - 1/128 every $1/2^n$ by the baud rate generator. The resulting GCLK is then used as the clock for control of the transmit and receive controllers. GCLK is then frequency divided by 8 to create TCLK used for the shift out and sampling clock of data bits.

6.3.5 System Reset and Initialization

The serial interface can be initialized by either resetting the system or by using a program to set bit 5 of the control register to "1".

The above procedure sets the system reset flipflop of the serial interface to maintain the status until bit 5 is set to "0" by the program when writing to the control register next. This is convenient because it restricts unnecessary operations until the initialize values required by the initial setting program are set after the power is turned on.

The following registers and some of the internal logic are initialized by system resetting. Register initialization is as follows.

Register	7	6	5	4	3	2	1	0
Control Register	CKSE	×	RES	×	×	×	×	INTM
	1	×	1	×	×	×	×	1
Command Register	×	×	RTS	ERS	SBRK	RxEN	×	TxEN
	×	×	0	1	0	0	0	0
Status Register	×	RBRK	FE	OE	PE	TxE	RxRDY	TxRDY
	0	0	0	0	0	1	0	TxINTM

6.3.6 Internal Registers

6.3.6.1 Control Register

This register makes the basic serial interface settings.

bit 0 enables/disables interrupt signals

1 : disables interrupt signals

0 : enables interrupt signals

bit 5 software reset

1 : reset

0 : cancel

bit 7 selects the clock to be frequency divided

1 : system clock

0 : BCLK

7	6	5	4	3	2	1	0
CKSE	×	RES	×	×	×	×	INTM

× : Don't Care

Figure 6.25 Serial Control Register

6.3.6.2 Mode Registers 0 - 1

These registers are provided for each channel and are related to data format and masking of error causes.

bit 0 sets the number of stop bits
- 1 : 2 stop bits
- 0 : 1 stop bit

bit 1 controls interrupts with TxRDY
- 1 : Interrupt Masked
- 0 : Interrupt Enable

bits 3, 2 set character length

bits 3 2
- 0 0 : 5 bits character
- 0 1 : 6 bits character
- 1 0 : 7 bits character
- 1 1 : 8 bits character

bit 4 parity control
- 1 : with parity
- 0 : no parity

bit 5 parity selection
- 1 : odd parity
- 0 : even parity

bit 6 controls interrupts due to receive errors
- 1 : Interrupt Masked
- 0 : Interrupt Enable

bit 7 controls interrupts due to character receiving
- 1 : Interrupt Masked
- 0 : Interrupt Enable

7	6	5	4	3	2	1	0
Rx INTM	ER INTM	PEO	PEN	CL1	CL0	Tx INTM	ST

Figure 6.26 Serial Mode Register

6.3.6.3 Command Registers 0 - 1

These registers are provided for each channel and are related to transmit and receive control, and modem control line control.

bit 0 sets transmit status
- 1 : enables transmission
- 0 : disables transmission

bit 2 sets receive status
- 1 : enables receive
- 0 : disables receive

bit 3 sets break character transmission
- 1 : enables transmission
- 0 : disables transmission

bit 4 resets the receive error flag
- 1 : resets the PE, OE, FE, RBRK flags
- 0 : no operation

bit 5 output pin RTS control (channel 0 only)
- 1 : RTS is 0
- 0 : RTS is 1

7	6	5	4	3	2	1	0
×	×	RTS	ERS	SBRK	RxEN	0	TxEN

× : Don't Care

Figure 6.27 Serial Command Register

6.3.6.4 Status Registers 0 - 1

These registers are provided for each channel and indicate channel status.

bit 0 meaning differs depending on the value towhich TxINTM of the mode register is set

TxINTM = 1 TxRDY is "1" when the transmit data buffer is empty
TxINTM = 0 TxRDY is "1" when
(transmit data buffer is empty) x (CTS = 0) x (TxEN = 1)

bit 1 indicates receive data buffer status

"1" when the receive data buffer contains received characters.

bit 2 indicates transmit data buffer status

"1" when the transmit data buffer is empty and not transmitting.

bit 3 parity error bit

"1" when a receive data parity error is detected.

bit 4 overrun error

"1" when the data in the receive data buffer are erased before being read by receiving the next data.

bit 5 framing error

"1" when a stop bit is not detected after receipt of one character.

bit 6 break character detect

"1" when a break character is detected during receiving.

7	6	5	4	3	2	1	0
0	RBRK	FE	OE	PE	TxE	Rx RDY	Tx RDY

Figure 6.28 Serial Status Register

6.3.6.5 Prescaler Register

This register determines prescaler frequency divide ratios.

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

Figure 6.29 Serial Prescaler Register

Values from 0 to 255 can be set in these registers. The frequency divide ratios for each of these values are as follows.

set value	frequency divide ratio
0	×1/256
1	×1
2~255	×1/2~×1/255

6.3.6.6 Baud Rate Registers 0 - 1

These registers are provided for each channel and are used to determine how much the clock (PCLK) frequency divided with a prescalar will be further frequency divided. Only one bit can be set at one time.

The clock (GCLK) obtained by frequency dividing the PCLK clock selected with these registers is used to control the transmit and receive controller. The clock (TCLK) actually used for data bit shiftout and sampling is 1/8 the value of GCLK after being frequency divided by 8.

7	6	5	4	3	2	1	0
B7	B6	B5	B4	B3	B2	B1	B0

Figure 6.30 Baud Rate Register

B0 = 1	GCLK = PCLK × $1/2^0$ = PCLK × 1	TCLK = GCLK × 1/8 = PCLK × 1/8
B1 = 1	GCLK = PCLK × $1/2^1$ = PCLK × 1/2	TCLK = GCLK × 1/8 = PCLK × 1/16
B2 = 1	GCLK = PCLK × $1/2^2$ = PCLK × 1/4	TCLK = GCLK × 1/8 = PCLK × 1/32
B3 = 1	GCLK = PCLK × $1/2^3$ = PCLK × 1/8	TCLK = GCLK × 1/8 = PCLK × 1/64
B4 = 1	GCLK = PCLK × $1/2^4$ = PCLK × 1/16	TCLK = GCLK × 1/8 = PCLK × 1/128
B5 = 1	GCLK = PCLK × $1/2^5$ = PCLK × 1/32	TCLK = GCLK × 1/8 = PCLK × 1/256
B6 = 1	GCLK = PCLK × $1/2^6$ = PCLK × 1/64	TCLK = GCLK × 1/8 = PCLK × 1/512
B7 = 1	GCLK = PCLK × $1/2^7$ = PCLK × 1/128	TCLK = GCLK × 1/8 = PCLK × 1/1024

6.3.6.7 Transmit and Receive Data Buffer Registers 0 - 1

These registers are provided for each channel and temporarily hold data to be sent or data that has been received.

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0

Figure 6.31 Serial Date Register

The addresses of the transmit and receive data buffer registers are the same but each is combined with a read or write signal for identification.

6.3.6.8 Register Map

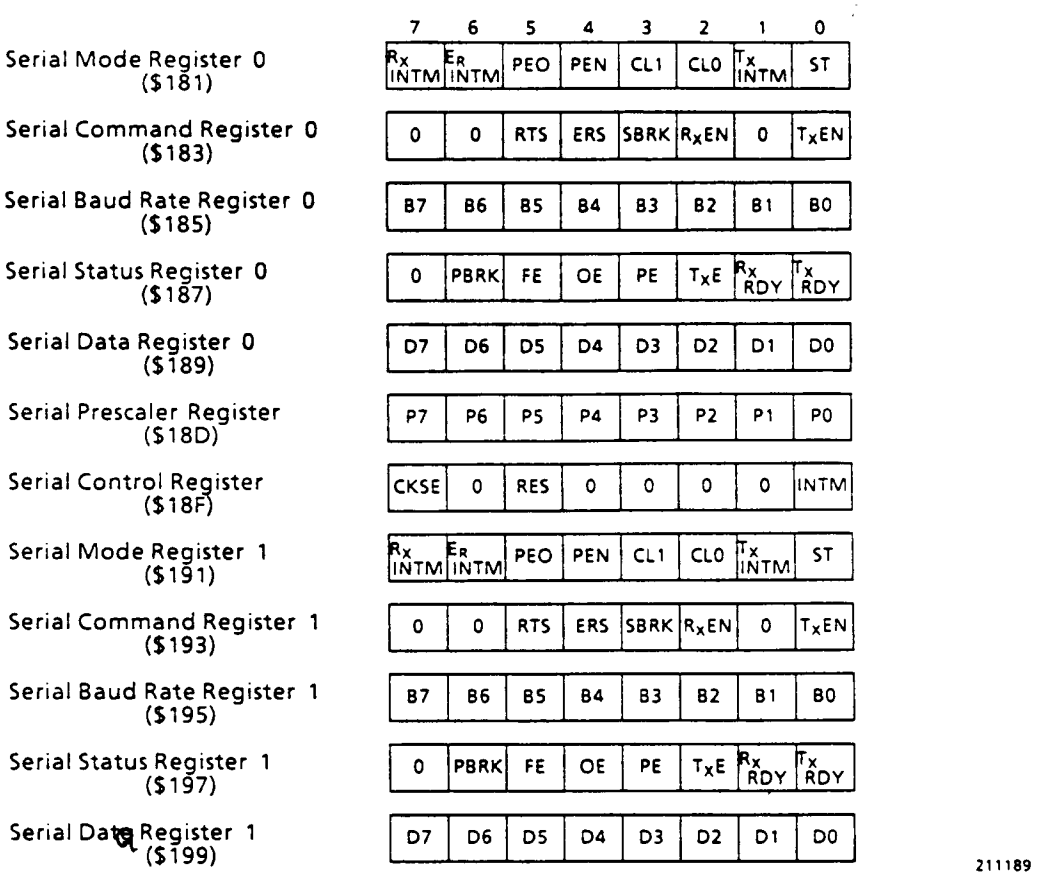


Figure 6.32 Serial Register Map

6.4 PARALLEL INTERFACE

6.4.1 Over view

This 10-bit general-purpose port is built in for use as the parallel interface. The 10-bit port is configured with two 4-bit ports (ports 0 and 1) and one 2-bit port (port 2).

Ports 0 and 1 are multiplexed to the stepping motor control ports (M00 - M03, M10 - M13). Port 2 is multiplexed to the built-in serial interface control pins ($\overline{CTS}$ and $\overline{RTS}$).

The control registers for these ports are located in the stepping motor controller (ports 0 and 1) and the serial interface (port 2).

6.4.1.1 Festure

10-bit general-purpose port
Enable setting on bit basis

6.4.2 Port 0 (P00~P03)

Port 0 is a 4-bit general-purpose I/O port (P0: memory address $333) whose function is specified by the control register (P01CR: bits 0~3 of memory address $337) for each bit. The control register is initialized to "0" by resetting, and Port 0 enters in the input mode. This port is also serviceable as a stepping motor control port channel 0 (M00~M03), so either the general-purpose I/O or the stepping motor control port can be selected by the control register (SMMOD: bits 0 and 1 of memory address $331). This port is served as the general-purpose I/O port by resetting.

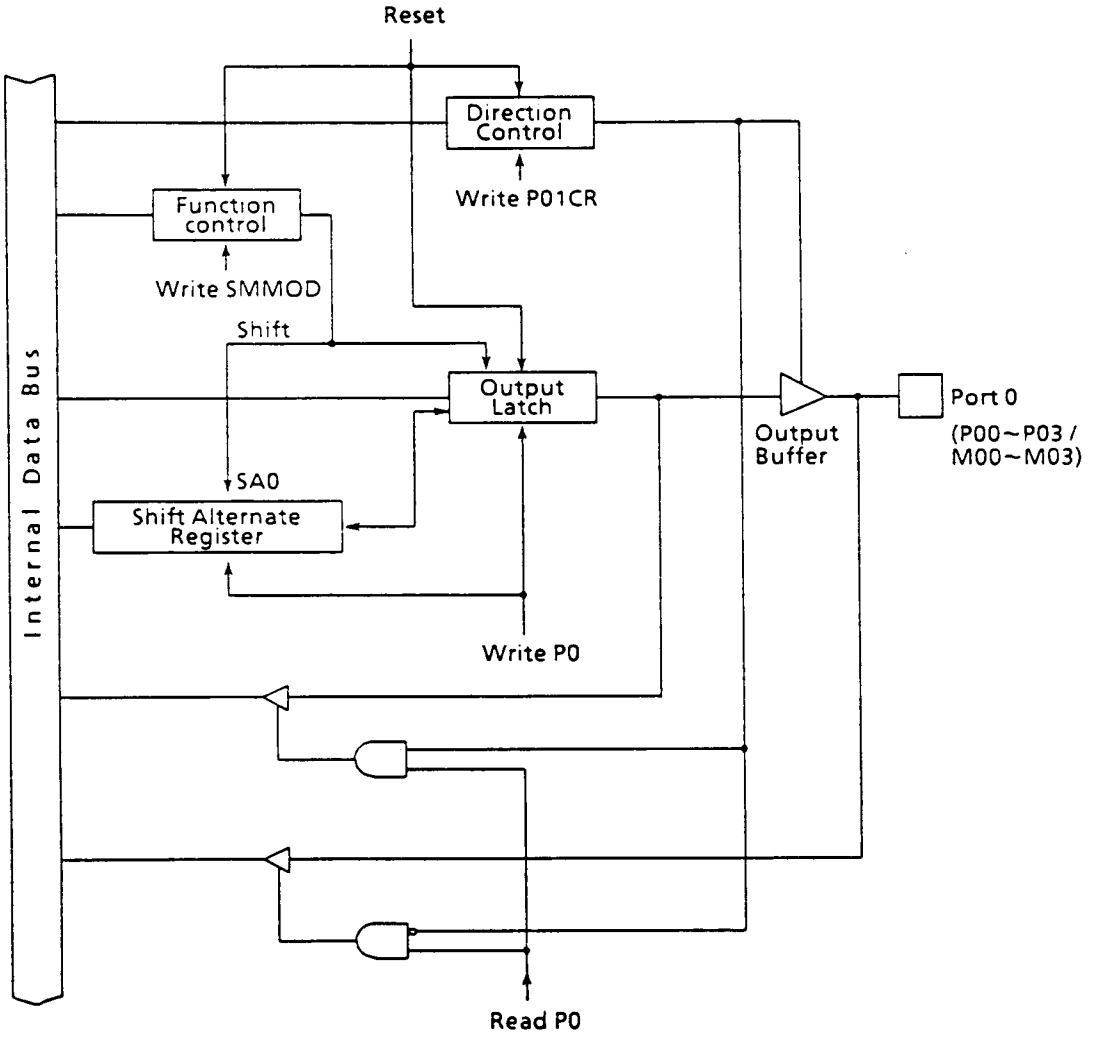


Figure 6.33 Port 0

6.4.2 Port 1 (P10~P13)

Port 1 is a 4-bit general-purpose I/O port (P1: memory address $335) whose I/O function is specified by the control register (P01CR: bits 4~7 of memory address $337). The control register is initialized to "0" by resetting, whereby Port 1 turns to the input mode.

This port is also serviceable as a stepping motor control port channel 1 (M10~M13), so either the general-purpose I/O or the stepping motor control port can be selected by the control register (SMMOD: bits 4 and 5 of memory address $331). It is served as the general-purpose I/O port by resetting.

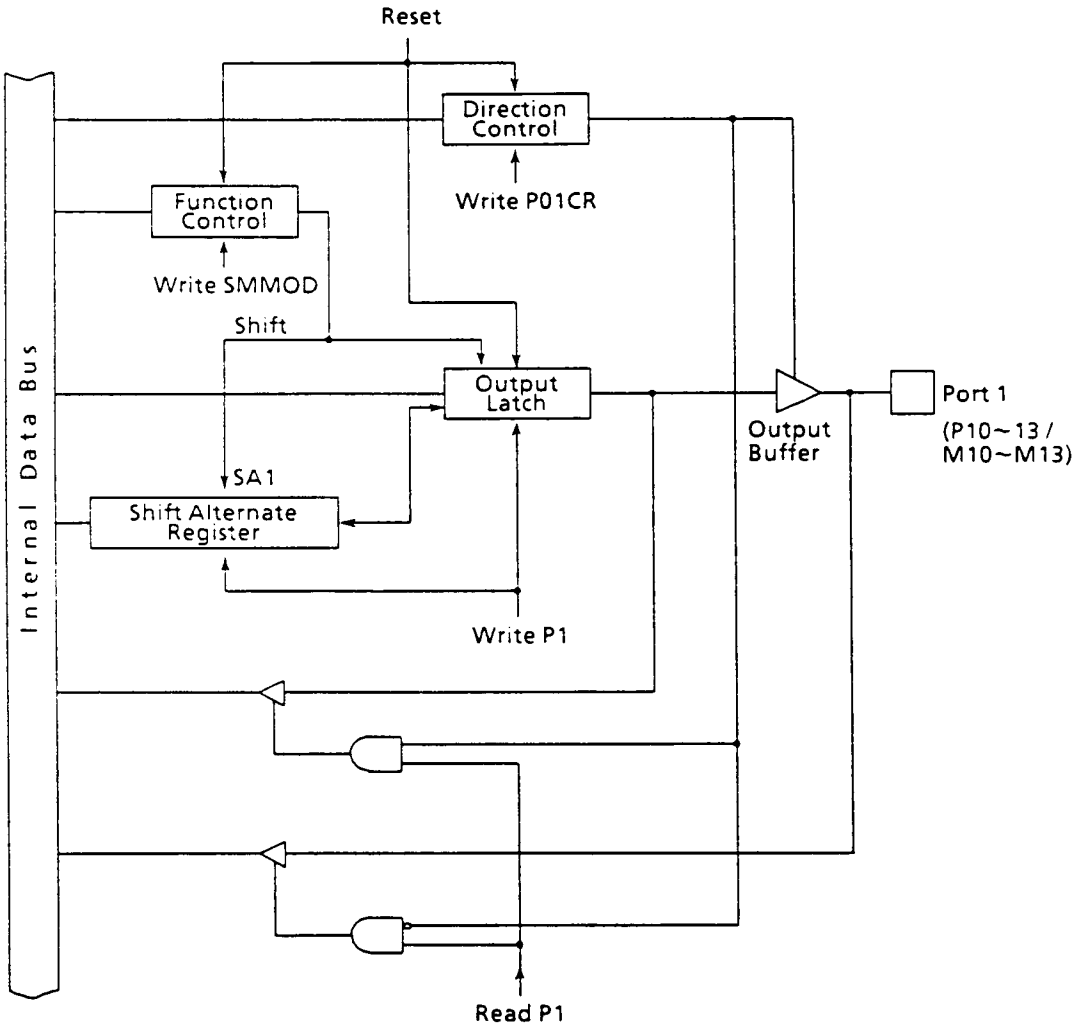


Figure 6.34 Port 1

P0 ($333)

7	6	5	4	3	2	1	0
SA03	SA02	SA01	SA00	P03	P02	P01	P00

P00-3 : Port 0
SA00-3 : Shifter alternate register 0 for stepping motor control

* initial value : X0H (X = undefined)
** SA00-3 always set at "FH" when read out

P1 ($335)

7	6	5	4	3	2	1	0
SA13	SA12	SA11	SA10	P13	P12	P11	P10

P10-3 : Port1
SA10-3 : Shifter alternate register 1 for stepping motor control

* initial value : X0H (X = undefined)
** SA10-3 always set at "FH" when read out

P01CR ($337)

P1				P0			
7	6	5	4	3	2	1	0
P13C	P12C	P11C	P10C	P03C	P02C	P01C	P00C

Select input/output of port 0 and 1

0 : Input port
1 : Output port

* initial value : 00H

201289

Figure 6.35 Registers for Port 0 and 1

	P1				P0			
	7	6	5	4	3	2	1	0
SMMOD ($331)	SM1M		P1C		SM0M		P0C	

P0C : Select Port 0 function

	P03	P02	P01	P00	SMC Trigger Signal
00	IN/OUT	IN/OUT	IN/OUT	IN/OUT	–
01	IN/OUT	IN/OUT	IN/OUT	IN/Tout3	–
10	IN/M03	IN/M02	IN/M01	IN / M00	TFF3
11					

P1C : Select Port 1 function

	P13	P12	P11	P10	SMC Trigger Signal
00	IN/OUT	IN/OUT	IN/OUT	IN/OUT	–
01	IN/OUT	IN/OUT	IN/OUT	IN/Tout4	–
10	IN/M13	IN/M12	IN/M11	IN / M10	TFF4
11					

IN : Input Port OUT : Output Port Tout3/Tout4 : Timer output port
M00-3/10-3 : Stepping motor control port 0/1

SMOM/SM1M : Stepping motor control port 0/1 mode

00	—
01	—
10	4-phase 1 or 2 excitation (Full-step)
11	4-phase 1-2 excitation (Half-step)

* initial value : 00H
** bit 3 and 7 always set at "1" when read out

151289

Figure 6.36 Register for Port 0 and 1

6.4.3 Port 2 (P20 and P21)

Port 2 is a general-purpose input/output port with which input or output can be specified in bit units. Input or output is specified using the control registers (P20CR bit and P21CR bit).

In addition to the input/output functions, port 2 also has a built-in serial interface ch0 input/output pin function. This is specified using the control register (EMSIO bit).

P20 is also used as the serial interface $\overline{\text{RTS0}}$ (Request To Send) input pin. and P21 is also used as the $\overline{\text{CTS0}}$ (Clear To Send) output pin. The $\overline{\text{CTS0}}$ or $\overline{\text{RTS0}}$ function are enabled by setting the EMSIO bit to "1". The value of the EMSIO bit has a higher priority than the values of the P20CR and P21CR bits.

Resets clear the port register and control register to "00" and enable the general-purpose input port mode.

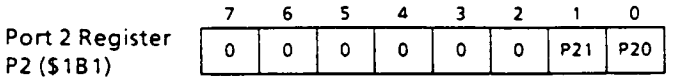


Figure 6.37 Port 2

* initial value : 00H

6.4.5 Port 2 Control Register

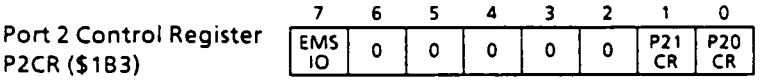


Figure 6.38 Port 2 Control Register

P20CR : Select input/output of port 20 (write only)
- 0 : Input mode
- 1 : Output mode

P21CR : Select input/output of port 21 (write only)
- 0 : Input mode
- 1 : Output mode

EMSIO : Select port 2 function
- 0 : General-purpose port
- 1 : Internal serial I/F control pin

* initial value : 00H

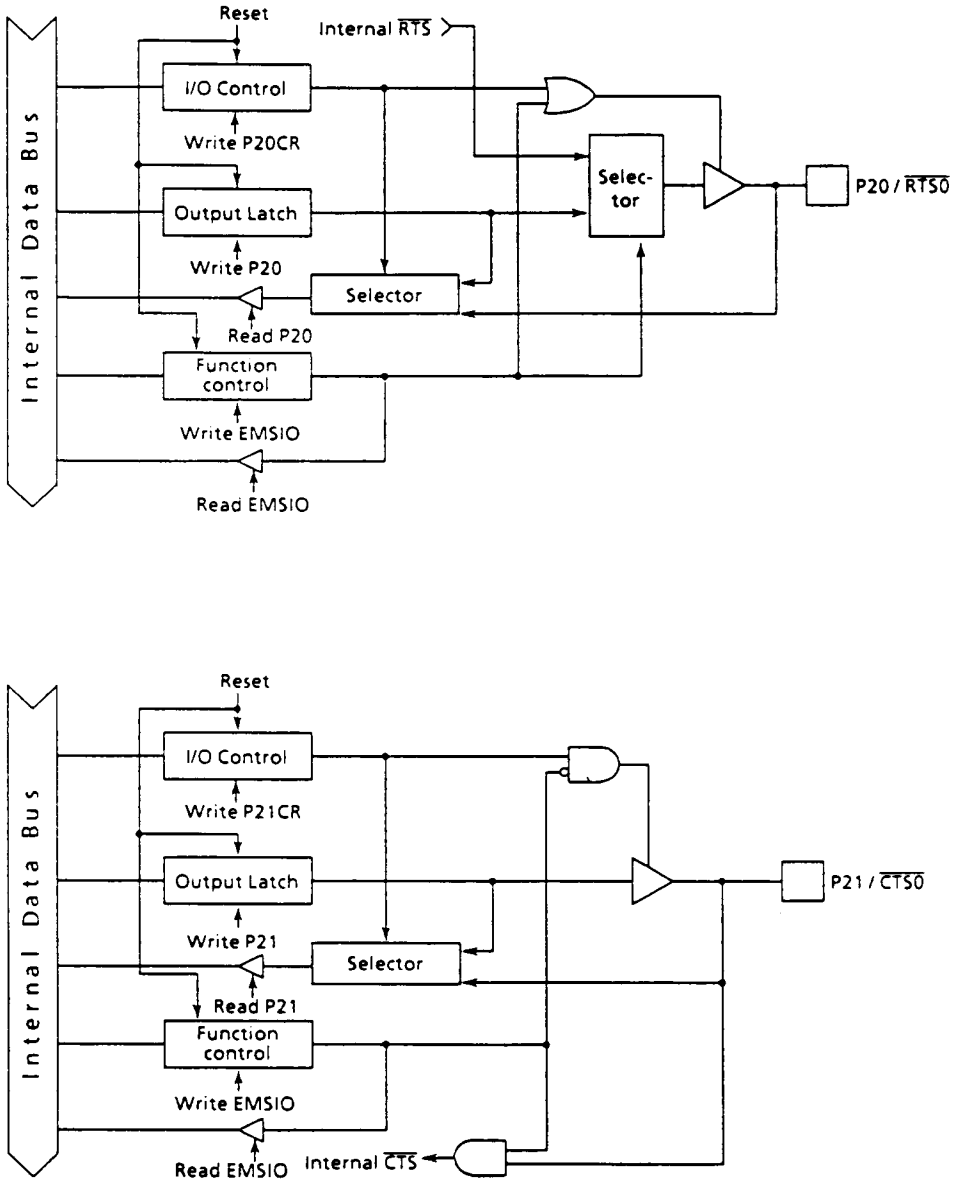


Figure 6.39 Port 2

6.5 16-BIT TIMER

6.5.1 Overview

This functional block is configured with three 16-bit counter channels. Each channel has an 8-bit prescalar (only the system clock is effective). Channels 1 and 2 generate interrupt requests. Two of the channels can also be cascade connected for long cycle counts. Channel 0 can be used for watchdog timer.

Three independent built-in 16-bit counter channels.
Count data read capability.
Interrupts from channel 1 and 2.
Software/hardware triggers.
Programmable operation modes.
Programmable pulse waveform output (channels 1 and 2 only).
One-shot pulse output (channels 1 and 2 only).
Event counting.
Maximum 8MHz count.
Watchdog timer (channel 0, Reset Output)

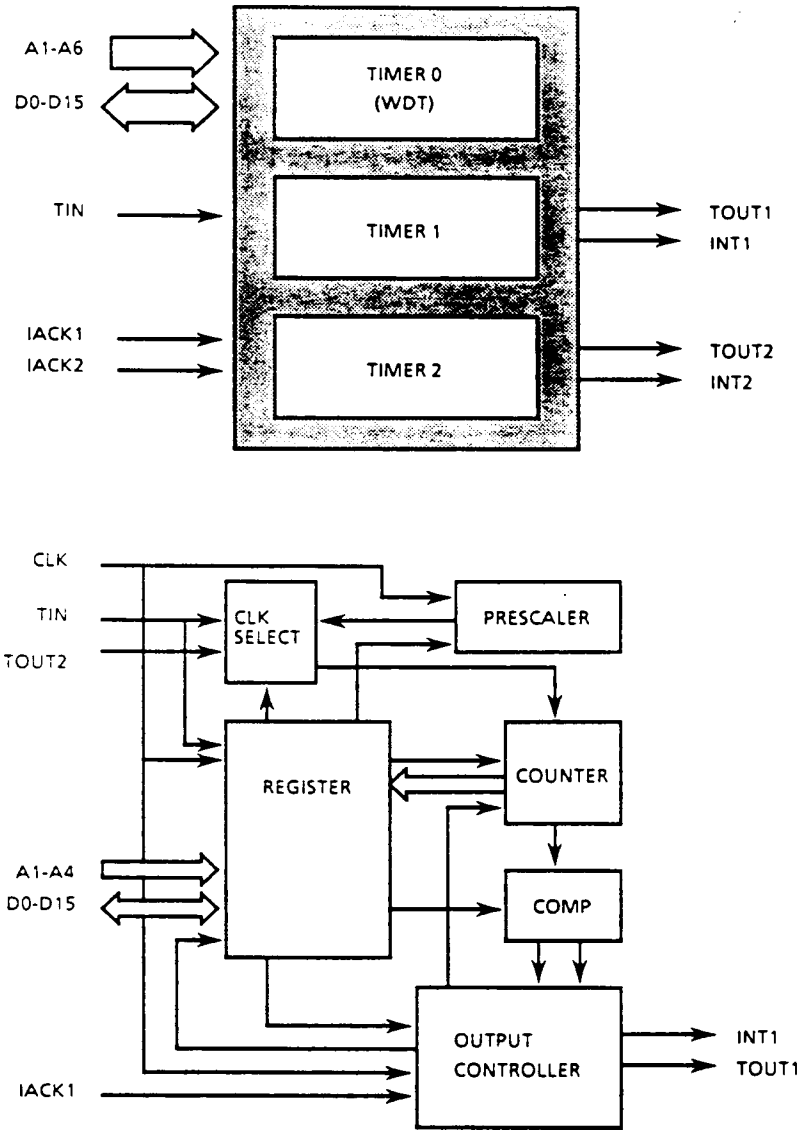


Figure 6.40 16-bit Timer Block Diagram

6.5.2 Operation

6.5.2.1 Channels 0 - 2

Each channel has a built-in 16-bit up counter. When the count value of a 16-bit counter reaches the maximum count register value, the count match signal is issued and the counter value is cleared. Channels 1 and 2 can issue interrupt request signals.

6.5.2.1.1 Channel 0

Channel 0 has three 16-bit registers: a control register, a count register and a MAX count register. External clock and signal can be input to this channel through the external input pin (common for channels 0 - 2). Channel 0 has a dedicated watchdog timer.

6.5.2.1.2 Channels 1, 2

Channels 1 and 2 have two MAX count registers, the control register and count register. These channels also have an external input pin (common for channels 0 - 2) and external output pin; therefore, external event counting and the output of optional waveforms are possible.

6.5.2.2 Setting and Changing the Maximum Count Value

The MAX count register contains the maximum count value.

The maximum count value can be set or changed by writing a value to the MAX count register. (Writing is possible even during operation.)

6.5.2.3 Count Value Read Out

It is possible to read out count value from the count register during counting. The contents of the count register are not affected by either read or write operations.

6.5.3 8-bit Prescaler

Count clock (only system CLK) frequency can be divided by 2, 4, 8, 16, 32, 64, 128 and 256. The frequency division ratio is set by the control register.

6.5.4 Interrupt Generation

6.5.4.1 Interrupt Generation

The interrupt generation mode is controlled by the control register.

6.5.4.2 Interrupt Generation Timing

Interrupt request signals are generated when the count match signal is generated.

6.5.5 Register Read and Write

Register read does not affect count operations and control signals. The count match signal (after comparison of a count value and MAX count register value) is ignored while writing to a register during a count operation; however, there is no influence on count operations for other channels .

6.5.6 Timer Operating Mode

6.5.6.1 Timer-in Function

External clock, trigger signal or control signal can be input to the external input pin (TIN). Signal input to TIN is sampled at the rise of the system clock (TIN). Clock or pulse input to TIN must have a pulse width wider than the system clock cycle.

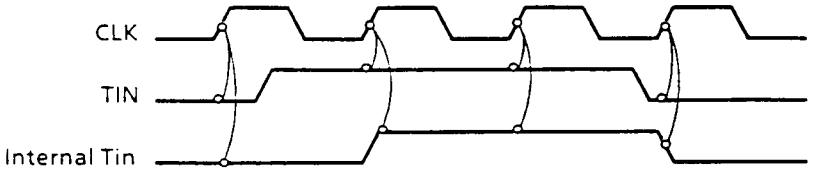


Figure 6.41 TIN Timming

6.5.6.1.1 External Clock

External clock can be used as count clock.

6.5.6.1.2 Count Control Signal

A count start function and count wait function are provided for count control.
These functions are selected with the control register.

Count start function

The trigger signal is input to the external input pin (TIN). The count starts when the trigger signal becomes low level (falling edge). (Second and following inputs are ignored.)

Count wait function

The control signal is input to the external input pin (TIN). The count continues while the control signal is at low level. Counting stops when the control signal becomes high level and restarts when the control signal goes to low level. (Unless the counter is cleared, counting will resume from the last value.)

6.5.6.2 Count Clock Selection

The maximum count frequency is 8MHz (target).
Either the system clock, external or another counter output is selected as the count clock.

6.5.6.3 Timer-out Function

The external output pin (TOUT) outputs either a pulse or a square waveform.

Signals output from the TOUT pin (pulse generation, output level inversion timing) are synchronized with the falling edge of the system clock.

Figure 6.42 TOUT Timming

6.5.6.4 MAX Count Register Setting

Each channel has one or two MAX count registers (here called MAX1 and MAX2). The channel can use either MAX1 or MAX2 alone, or both alternately. (Example given later.) The MAX count registers are selected by the control register.

6.5.6.5 Watchdog Timer

When enabled, channel 0 functions as a watchdog timer. When the timer reaches the count value specified with the MAX count register, $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are asserted for 12 clock cycles to reset the processor and external circuit.

6.5.7 Register Configuration

6.5.7.1 Count Register

This register has a 16-bit configuration.

Count value is read out from this register. The higher order bits or the lower order bits can be read out separately. This register is read only and the counting is not affected by writing.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COUNT REGISTER	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

Figure 6.43 Timer Count Register

6.5.7.2 MAX Count Register

This register has a 16-bit configuration.

The maximum count value is to be written to the MAX count register.

This register can be read and written.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAX COUNT REGISTER 1	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0

Figure 6.44 Timer MAX Count Register 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAX COUNT REGISTER 2	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0

Figure 6.45 Timer MAX Count Register 2

6.5.7.3 Control Register

This register has a 16-bit configuration.

The control register controls count operations and has the following bits.

15~14	CK2, CK1	input clock setting
13~10	P4, P3, P2, P1	prescaler value setting
9~8	T2, T1	external signal selection
7~6	1/N, R/P	output signal control (channels 1, 2 only)
5~4	MR2, MR1	MAX count register setting (channels 1, 2 only)
2	INT	interrupt request bit
1	CS	count clock input control
0	TS	timer operation setting

This register can be read and written.

CONTROL REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CK2	CK1	P4	P3	P2	P1	T2	T1	N/1	R/P	MR2	MR1	0	INT	CS	TS

Figure 6.46 Timer Control Register

CK2, CK1	:	00	–	System clk
		01	–	external clk
		10	–	other timer output 1
		11	–	other timer output 2
P4, P3, P2, P1	:	0000	–	(not use)
		0001	–	Prescaler 2
		0010	–	Prescaler 4
		0011	–	Prescaler 8
		0100	–	Prescaler 16
		0101	–	Prescaler 32
		0110	–	Prescaler 64
		0111	–	Prescaler 128
		1xxx	–	Prescaler 256
T2, T1	:	00	–	This channel select external clock and not use count start function and count wait function
		10	–	set count start func
		11	–	set count wait func
N/1	:	0	–	complete count operation by one time
		1	–	repeat count operation
R/P	:	0	–	generate pulse for output signal
		1	–	reverse output level
MR2, MR1	:	01	–	only use MAX1
		10	–	only use MAX2
		11	–	use both MAX1 and MAX2
INT	:	0	–	Not Generate interrupt request signal
		1	–	Generate interrupt request signal
CS	:	0	–	put count clock in counter (start count operation)
		1	–	not count clock in counter (stop count operation)
TS	:	0	–	clear counter
		1	–	restart counter

Note : Channel 0 bit 2,15 have no meanings because Channel 0 has no output.

6.5.7.4 Register Map

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONTROL REGISTER 0 [$200]	0	CK1	P4	P3	P2	P1	T2	T1	1	1	0	1	0	0	CS	TS
MAX COUNT REGISTER 10 [$204]	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
COUNT REGISTER 0 [$20C]	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
CONTROL REGISTER 1 [$220]	CK2	CK1	P4	P3	P2	P1	T2	T1	N/1	R/P	MR2	MR1	0	INT	CS	TS
MAX COUNT REGISTER 11 [$224]	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
MAX COUNT REGISTER 21 [$228]	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
COUNT REGISTER 10 [$22C]	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
CONTROL REGISTER 2 [$240]	CK2	CK1	P4	P3	P2	P1	T2	T1	N/1	R/P	MR2	MR1	0	INT	CS	TS
MAX COUNT REGISTER 12 [$244]	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
MAX COUNT REGISTER 22 [$248]	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
COUNT REGISTER 2 [$24C]	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

Figure 6.47 Timer Register Map

6.6 8-BIT TIMER

6.6.1 Over view

There are two 8-bit interval timers (timer 3 and 4), each of which operates independently.

Each of these 8-bit timers is configured with a main timer (timer 31 and 41) and sub timer (timers 30 and 40). Two 16-bit timers can also be configured by cascade connecting the main and sub timers. These 8-bit timers have the following four operating modes.

6.6.1.1 Feature

Two 8-bit interval timer modes

Two 16-bit interval timer modes

Two 8-bit programmable Pulse Generation (PPG) output modes

Two 8-bit PWM output modes

Figure 6.48 shows a block diagram of 8-bit timer 3 (timers 30 and 31). Timer 4 has the same circuit configuration as timer 3.

Each timer consists of an 8-bit up counter, 8-bit comparator, 8-bit timer register and timer flip-flop.

Of the clock sources input to each timer, T1, T16 and T256 are obtained from the 9-bit prescaler shown in Figure 6.49.

The 8-bit timer operation modes and timer flip-flops are controlled by five control registers (TCLK, TFFCR, TMOD, TIRCR and TRUN).

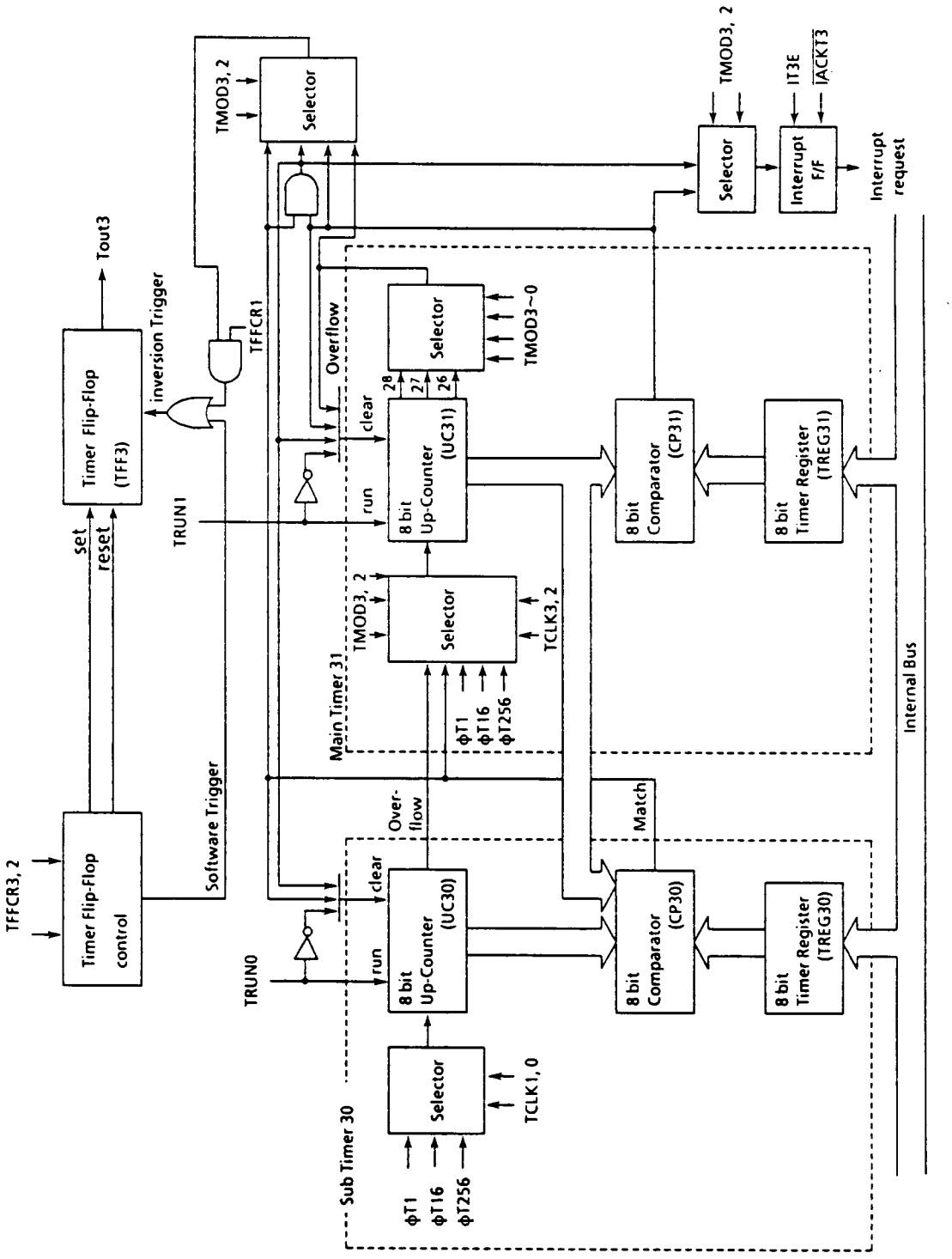


Figure 6.48 Block Diagram of 8-bit Timers (Timer 3)

6.6.2 Functional Description

(1) Prescaler

An 9-bit prescaler is provided to further divide the clock frequency already divided to a 1/4 of the frequency of the source clock (fc).

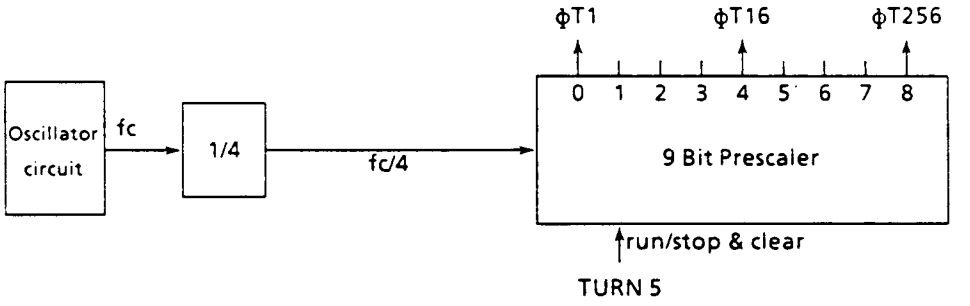
It generates a input clock pulse for the 8-bit timers.

For the 8-bit timers, three types of clock are generated (φT1, φT16 and φT256).

The prescaler can be run or stopped by using the 5th bit TRUN5 (PRRUN) of the timer control register (TRUN). Setting TRUN5 to "1" makes the prescaler to count, and setting it to "0" clears the prescaler to stop.

By resetting, TRUN5 is initialized to "0", making the prescaler clear and stop.

	Cycle	
Input clock (fc)	8MHz	16MHz
φT1	1.0μs	0.5μs
φT16	16μs	8μs
φT256	256μs	128μs



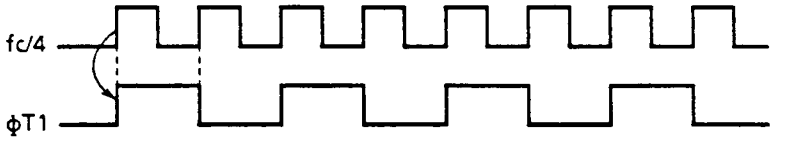


Figure 6.49 Prescaler

(2) Up-counter

This is an 8-bit binary counter that counts up by an input clock pulse specified by an 8-bit timer clock control register (TCLK) and an 8-bit timer mode register (TMOD).

The input clock pulse for Sub Timer 30 and 40 is selected from φT1, φT16 and φT256 according to the setting of the TCLK register.

However, φT256 cannot be selected as the input clock pulse for these timers in the 16-bit timer mode (TMOD3, 2 = 01/TMOD7, 6 = 01).

Example: When setting TCLK1, 0 = "01", φT1 is selected as the input clock pulse for Sub Timer 30.

The input clock pulse to Main Timer 31 and 41 is selected according to the operating mode. In the 16-bit timer mode, the overflow output of Sub Timer 30 and 40 is automatically selected as the input clock pulse to the main timer, regardless of the setting of the TCLK register.

In the other operating modes, the clock pulse is selected among the internal clocks φT1, φT16 and φT256, and the output of the Sub Timer 30 and 40 comparator (match signal).

Example: If TMOD 3, 2 = 01, the overflow output of Sub Timer 30 is selected as the input clock to Main Timer 31.
If TMOD 3, 2 = 00 and TCLK3, 2 = 01, φT1 is selected as the input clock to Main Timer 31.

The operating mode is selected by the TMOD register. This register is initialized to TMOD3, 2 = 00/TMOD7, 6 = 00 by resetting, whereby the up-counter is place in the 8-bit timer mode.

Functions, count, stop or clear of the up-counter can be controlled for each interval timer by the timer control register TRUN.

By resetting, all up-counters are cleared to stop the timers.

(3) Timer Registers

8-bit registers are provided to set the interval time. When the set value of a timer register matches that of an up-counter, the match signal of their comparators turn to the active mode. If "00H" is set, this signal becomes active when the up-counter overflows.

(4) Comparators

A comparator compares the values in an up-counter and a timer register. When they matches, the up-counter is cleared to "0", and an interrupt trigger is generated. If the timer flip-flop inversion is enabled by the Timer Flip-Flop control register, the Timer Flip-flop is inverted. The sub timers do not generate inversions of the interrupt and timer flip-flop signals.

(5) Timer Flip-flops (Timer F/Fs)

The status of the Timer Flip-flop is inverted by the match signal (output by comparator). Its status can be output to the timer output pin Tout3 (also used as P00) and Tout4 (used as P10).

A Timer F/F is provided to each of Timer 3 and Timer 4, and is called TFF3 and TFF4, respectively. The status of TFF3 is output to Tout3, and that of TFF4 to Tout4.

The Timer F/Fs are controlled by a timer flip-flop control register (TFFCR).

In the case of TFF3 (timer F/F for the Timer 3), the flip-flop operation is described as follows:

Always set TFFCR0 and TFFCR4 to "1" when using a timer flip-flop. TFFCR0 and TFFCR4 are cleared to "0" by resets.

TFFCR1 (FF3IE) controls the inversion of TFF3. Setting this bit to "1" enables the inversion and setting it to "0" disable.

FF3IE is initialized to "0" by resetting.

The bits TFFCR3 and TFFCR2 (FF3C) are used to set/reset TFF3 or enable its inversion by software. TFF3 is reset by writing "00", set by "01" and inverted by "10".

Similarly, TFF4 is controlled by TFFCR7-4.

(6) Interrupt Request Flip-flops (Interrupt F/Fs)

The Interrupt F/F generates interrupt request signal to the interrupt controller by interrupt trigger of each timers.

The Interrupt F/Fs are controlled by a timer interrupt request control register (TIRCR).

In the case of timer 3, the flip-flop operation is described as follows :

TIRCR 0 (IT3E) controls the generation of interrupt request signal. Setting this bit to "1" enables the generation and setting it to "0" disable. The generated interrupt request signal is cleared by acknowledge signal ($\overline{\text{IACKT3}}$) from interrupt controller, or setting this bit to "0" . IT3E is cleared to "0" by resetting. Similarly, timer 4 is controlled by TIRCR 1 (IT4E).

	Timer 4				Timer 3			
	7	6	5	4	3	2	1	0
TMOD ($26D)	T4M		PWM4		T3M		PWM3	

PWM3/PWM4 : Select PWM3/4 cycle

("Don't care" in non-PWM modes)

00 : —
01 : $2^6 - 1$
10 : $2^7 - 1$
11 : $2^8 - 1$

T3M/T4M : Set operating mode of Timer 3/4

00 : 8-bit timer
01 : 16-bit timer
10 : 8-bit PPG output
11 : 8-bit PWM output

* initial value : 00H

151289

Figure 6.50 8-bit Timer Mode Register (TMOD)

	Main Timer 41		Sub Timer40		Main Timer31		Sub Timer30	
	7	6	5	4	3	2	1	0
TCLK ($269)	T41CLK		T40CLK		T31CLK		T30CLK	

T30CLK : Sub Timer 30 input clock

	TMOD3, 2 ≠ 01	TMOD3, 2 = 01
00	—	—
01	Internal clock φT1	Internal clock φT1
10	Internal clock φT16	Internal clock φT16
11	Internal clock φT256	Not used

T40CLK : Sub Timer 40 input clock

	TMOD7, 6 ≠ 01	TMOD7, 6 = 01
00	—	—
01	Internal clock φT1	Internal clock φT1
10	Internal clock φT16	Internal clock φT16
11	Internal clock φT256	Not used

T31CLK : Main Timer 31 input clock

	TMOD3, 2 ≠ 01	TMOD3, 2 = 01
00	Sub Timer30 comparator output	Sub Timer30 overflow output
01	Internal clock φT1	
10	Internal clock φT16	
11	Internal clock φT256	

T41CLK : Main Timer 41 input clock

	TMOD7, 6 ≠ 01	TMOD7, 6 = 01
00	Sub Timer40 comparator output	Sub Timer40 overflow output
01	Internal clock φT1	
10	Internal clock φT16	
11	Internal clock φT256	

* initial value: 00H

151289

Figure 6.51 8-bit Timer Clock Control Register (TCLK)

TRUN ($26F)

7	6	5	4	3	2	1	0
0	0	PRRUN	0	T41RUN	T40RUN	T31RUN	T30RUN

Select Sub Timer 30/40, Main Timer 31/41 and prescaler operation

0 : Stop and clesr
1 : Count

* initial value : 00H

151289

Figure 6.52 Timer Control Register (TRUN)

TFFCR ($26B)

TFF4				TFF3			
7	6	5	4	3	2	1	0
FF4C		FF4IE	1 #	FF3C		FF3IE	1 #

FF31E/FF41E : Invert timer flip-flop TFF3/TFF4

0 : Disable
1 : Enable

FF3C/FF4C : Control timer flip-flop TFF3/TFF4 write only

00 : Clear TFF3/TFF4 to "0"
01 : Set TFF3/TFF4 to "1"
10 : Invert value of TFF3/TFF4 (Software inversion)
11 : Don't care (Always set at "11" when read out)

: Always write "1" when using a timer flip-flop.
The flip-flops are cleared to "0" by resets.

* initiasl value : XX00XX00B (X = undefined)

151289

Figure 6.53 8-bit Timer Flip-flop Control Register (TFFCR)

TIRCR ($271)

7	6	5	4	3	2	1	0
–	–	–	–	–	–	IT4E	IT3E

IT3E/IT4E : Control timer 3/4 interrupt request

0 : Not generate the interrupt request (clear)
1 : Generate the interrupt request

bit 2~7: No register

151289

* initial value : XXXXXX00B (X = undefined)

Figure 6.54 Timer Interrupt Request Control Register (TIRCR)

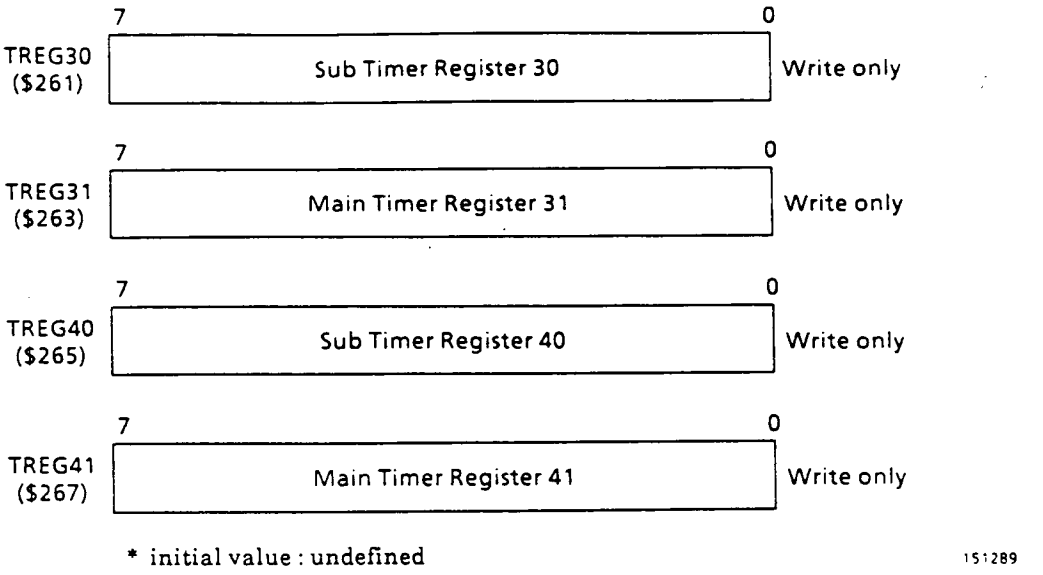


Figure 6.55 8-bit Timer Register

6.6.3 8-bit Timer Mode

The two interval timers 3 and 4 can operate independently as an 8-bit interval timer. Only the operation of Timer 3 is described because their operations are the same.

(1) Generating Interrupts at Specified Intervals

Periodic interrupts can be generated by using Timer 3 (Possible only with main timers 31 and 41) in the following procedure: Stop Timer 3, set the desired operating mode, input clock and cycle time in the registers TMOD, TCLK and TREG31. Then clear Timer 3 interrupt mask register, and start the counting of Timer 3.

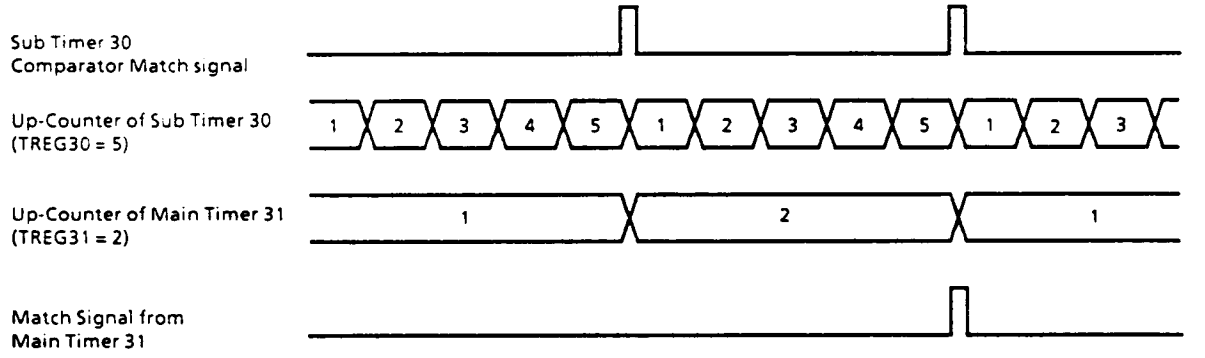
Use the following table for selecting the input clock:

Table 6.6.1 8-bit Timer Interrupt Cycle and Input Clock

Interrupt cycle @fc = 16 MHz	Resolution	Input clock
0.5μs ~ 127.5μs	0.5μs	φT1
8.0μs ~ 2.04ms	8.0μs	φT16
128.0μs ~ 32.64ms	128.0μs	φT256

(2) Making Main Timer 31 Count up by Match Signal from Sub Timer 30 Comparator.

Select the 8-bit timer mode, and set the comparator output of Sub Timer 30 as the input clock to Main Timer 31.



(3) Software Inversion

The timer flip-flops can be inverted by software independent of the timer operation.

Writing "10" into the bits TFFCR3, 2 inverts TFF3, and writing the same into TFFCR7, 6 inverts TFF4.

(4) Initial Setting of Timer Flip-Flops

The Timer Flip-flops can be initialized to either "0" or "1" without regard to the timer operation.

TFF3 is initialized to "0" by writing "00" into TFFCR3, 2, and "1" by writing "01" into these bits.

Note : Reading the data from the Timer Flip-flops and timer registers is prohibited.

6.6.4 16-bit Timer Mode

16-bit interval timers can be configured by cascade connecting the main and sub timers.

As both timer have the same function, only the Timer 3 is discussed.

Cascade connection of Sub Timer 30 and Main Timer 31 to use them as a 16-bit interval timer requires to set the bits 3 and 2 of the mode register TMOD to "01".

By selecting the 16-bit timer mode, the overflow output of Sub Timer 30 is automatically selected as the input clock to Main Timer 31, regardless of the set value of the clock control register TCLK. The input clock to Timer 30 is selected by TCLK. Note, however, that φT256 cannot be selected in the 16-bit timer mode. Table 6.6.2 shows the combinations of timer (interrupt) cycle and input clock.

Table 6.6.2 16-bit Timer (Interrupt) Cycle and Input Clock

Timer (interrupt) cycle @fc = 16MHz	Resolution	Input clock to Timer 0
0.5μs~32.77ms	0.5μs	φT1
8.0μs~524.28ms	8.0μs	φT16

To set the interrupt cycle, the lower eight bits of the timer cycle is set by TREG30 and the upper eight bits of that is set by TREG31. Note that TREG30 must be always set first (Writing data into TREG30 disables the comparator temporarily, which is restarted by writing data into TREG31).

Example : To generate interrupts at fc=16MHz every 0.5 second, the timer registers TREG30 and TREG31 should be set as follows:

As φT16 (=8μs @16MHz) is selected as the input clock,

0.5 sec/8μs = 62500 = F424H

Therefore,

TREG31 = F4H

TREG30 = 24H

The match signal is generated by Sub Timer 30 comparator each time the up-counter UC30 matches TREG30. In this case, the up-counter UC30 is not cleared.

Main Timer 31 comparator also generates the match signal each time the up-counter UC31 match TREG31. When the match signal is generated simultaneously from comparators of Sub Timer 30 and Main Timer 31, the up-counters UC30 and UC31 are cleared to "0", and the interrupt is generated. If the Timer Flip-flop inversion is enabled by the Timer Flip-flop control register, the timer flip-flop TFF3 is inverted at the same time.

Example: Given TREG31 = 04H and TREG30 = 80H

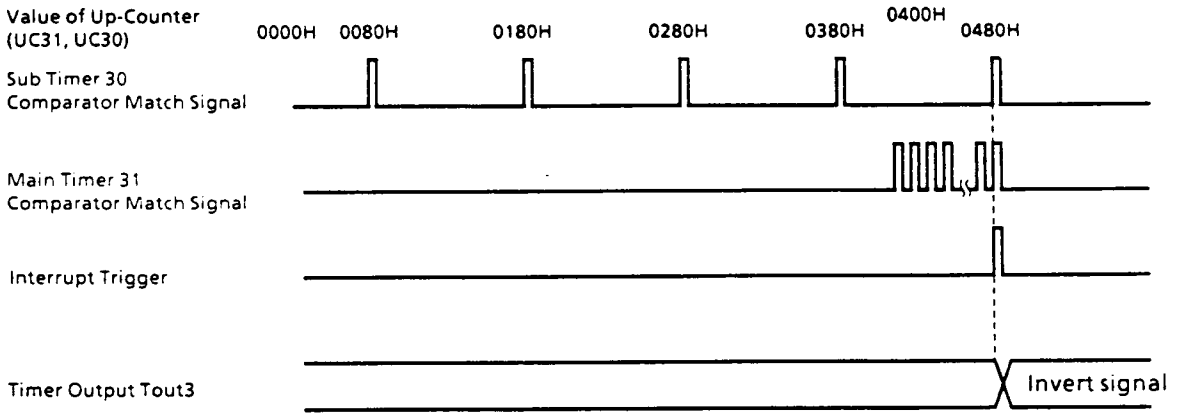
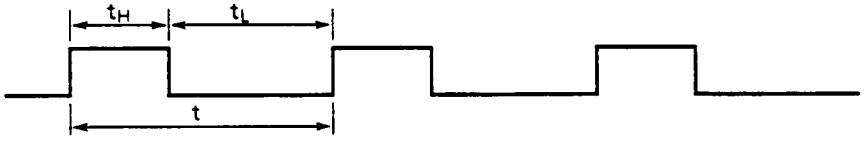


Figure 6.56

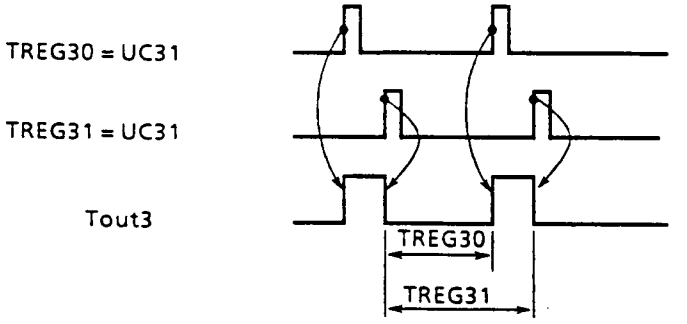
6.6.5 8-bit PPG (Programmable Pulse Generation) Mode

Pulse can be generated at any frequency and duty ratio by Main Timer 31 or Timer 41. The output pulse may be either low or high-active.

If Main Timer 31 is used, pulse is output to Tout3 (shared with P00), and the use of Main Timer 41 results in the output to Tout4 (shared with P10).



Following is the timing of Main Timer 3 (The operation is the same as when Main Timer 4 is selected):



In the 8-bit PPG mode, programmable pulse is generated by the inversion of the timer output each time the 8-bit up-counter 31 (UC31) matches the timer register TREG30 or TREG31.

Note that the set value of TREG30 must be smaller than that of TREG31.

In this mode, the up-counter UC30 of Sub Timer 30 can not be used (Set TRUN 0=1, and count the Timer 0).

The PPG mode can be illustrated as follows:

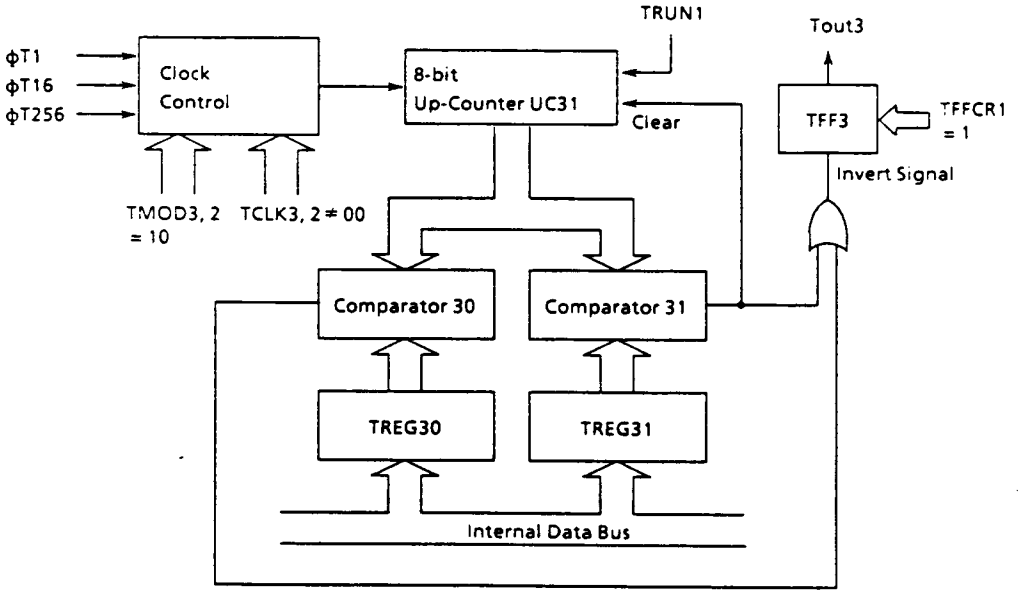
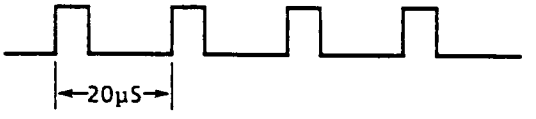


Figure 6.57 Block Diagram of 8-bit PPG Mode

Example : Generate pulse at 50kHz and 1/4 duty (@fc = 16MHz)



- Calculate the set values of the timer registers.

  To obtain the frequency of 50kHz, the pulse cycle t should be: 1/50kHz = 20μs.
  Given $\phi T1 = 0.5\mu s$ (@16MHz),
  $20\mu s/0.5\mu s = 40$
  Consequently, the Main timer register 31 (TREG31) should be set to 40 = 28H.
  Given a 1/4 duty, $t \times 1/4 = 20 \times 1/4 = 5\ \mu s$
  $5\ \mu s/0.5\ \mu s = 10$
  As a result, the Sub timer register 30 (TREG30) should be set to 10 = 0AH.

6.6.6 8-bit PWM (Pulse Width Modulation) Mode

This mode is only available for Main Timer 31 and Timer 41, and generates two 8-bit resolution PWM (PWM3 and PWM4).

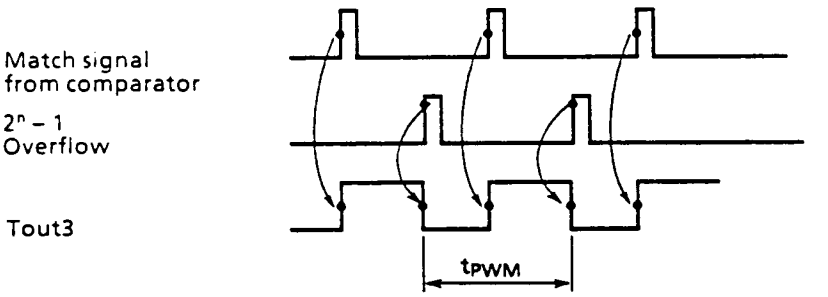
Pulse width modulation is output to Tout3 pin (shared with P00) when using Timer 3, and to Tout4 pin (shared with P10) when using Timer 4.

Following is the timing of Timer 3 (PWM3) (The operation is the same as when Timer 4 is selected):

The inversion of the timer output occurs when the up-counter (UC31) matches the set value of the timer register TREG31, as well as when an overflow of $2n-1$ (n = 6, 7 or 8 selected by TMOD1, 0) occurred at the counter. The up-counter UC31 is cleared by the occurrence of an overflow of $2n-1$.

The following condition must be obtained in this PWM mode:

(Set value of timer register) < (set overflow value of $2^n - 1$ counter)
(Set value of timer register) ≠ 0



The PWM mode can be illustrated as follows:

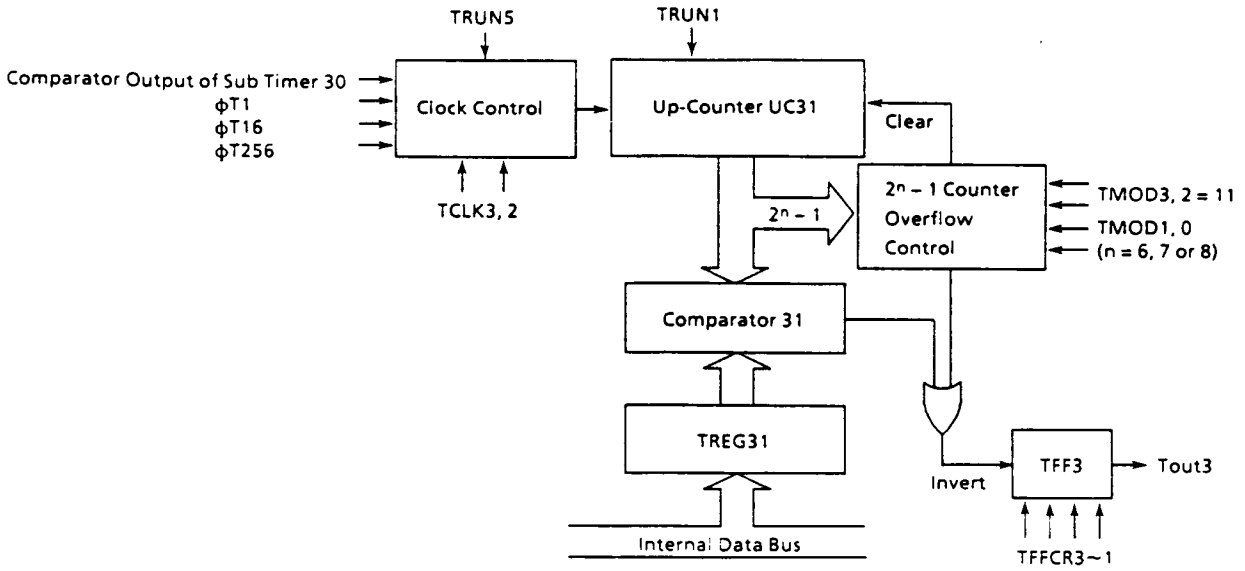
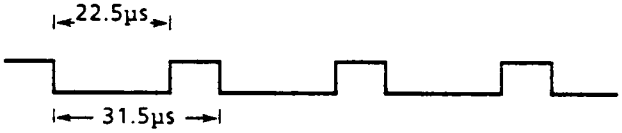


Figure 6.58 Block Diagram of 8-bit PWM Mode

Example : Generate the following PWM to the Tout3 pin at fc = 16MHz.



Assuming the PWM cycle is 31.5 μs when φT1 = 0.5 μs and @fc = 16MHz,
$31.5\mu s / 0.5\mu s = 63 = 2^6 - 1$
Consequently, n should be set at 6 (TMOD1, 0 = 01).
Given the "L" level period of 22.5μs, setting φT1 = 0.5μs results:
$22.5\mu s / 0.5\mu s = 45 = 2DH$
As a result, TREG31 should be set at 2DH.

Table 6.6 3 PWM Cycle and Selection of $2^n - 1$ Counter

	PWM cycle (@fc = 16MHz)		
	φT1	φT16	φT256
2^6-1	31.5μs	504μs	8.064ms
2^7-1	63.5μs	1016μs	16.256ms
2^8-1	127.5μs	2040μs	32.64ms

6.7 DMA CONTROLLER

6.7.1 Overview

This built-in DMA controller (DMAC) has three independent channels, a 24-bit memory address counter and 16-bit data transfer counter, and can transfer data at high speed without passing through the core processor.

Also, all channels of the external pin are provided with an input pin ($\overline{DREQ}$) for receiving DMA requests and an output pin ($\overline{DACK}$) for reporting responses. There is also a common input/output pin ($\overline{DTEND}$) for ending DMA operation and DMA service.

In the transfer mode, ~~two of the channels support single address transfers and one channel also supports dual address transfers.~~ CHANNEL 0 SUPPORTS ...

Features

Three independent DMA channels.
16M-byte maximum address space.
64k-byte maximum block transfer.
Byte and word transfer operands.
8M-byte/sec. maximum transfer speed.
Byte/demand/continuous bus modes.
Single and dual address modes.
Demand and continuous bus modes can use cycle steal mode.
Memory ↔ memory and memory ↔ I/O transfers.
Interrupts can be generated at end of DMA service and at error.

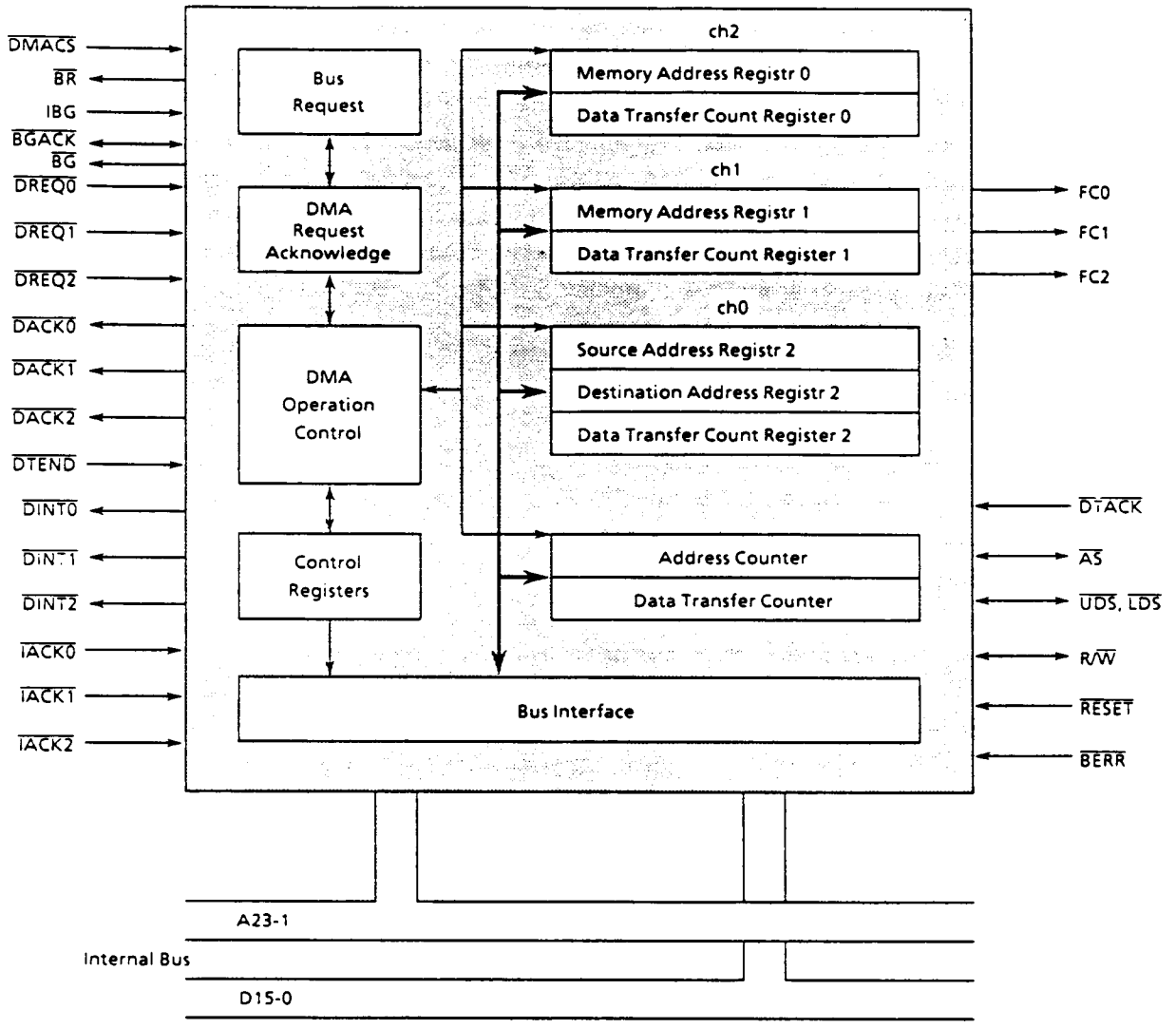


Figure 6.59 Internal Block of DMAC

6.7.2 DMAC Operation

6.7.2.1 Bus Arbitration

Bulit-in DMAC asserts $\overline{BR}$ (Bus request) signal to acquire a right of bus. The core checks $\overline{BR}$ (low level) signal and asserts $\overline{BG}$ (Bus Grant) signal. The DMAC accepts this $\overline{BG}$ signal and watches the external bus condition. Thereafter DMAC confirms bus release and asserts $\overline{BGACK}$ (Bus Acknowledge) signal.

6.7.2.2 Generating Transfer Requests

There are two methods for starting DMA transfers. One method is used to make requests by switching the $\overline{DREQ}$ signal to active status from the peripheral I/O device for transfers between memory and peripheral I/O devices. The other method is used for memory-to-memory transfers to set the DST bit of the channel control register (CHCR0) by programming and request the start of a transfer.

	Ch0	Ch1	Ch2
$\overline{DREQ}$	○	○	○
DST bit	○	×	×

6.7.2.2.1 External Requests

When transferring between memory and peripheral I/O devices, requests can be received by making the initial settings and then switching the transfer request signal ($\overline{DREQ}$) to active status from the peripheral I/O device. After that, the bus is captured and the DMA transfer is started by arbitration between the core processor and DMAC.

The transfer request signal ($\overline{DREQ}$) can only be accepted to start a DMA transfer when the DMAE bits of the channel control register (CHCR) and operation control register (OPCR) are set. Also, when the data transfer count register (DTCR) value becomes "0000H" after a DMA transfer ends, $\overline{DREQ}$ is ignored for the interval to which the OPC bit of the channel status register (CHSR) is set. The OPC bit is cleared when DTCR is overwritten.

Also, since the $\overline{DREQ}$ signal is sampled at system clock rises, a transfer request is assumed when Low level continues for at least one clock and then DMA sends a bus request to the core processor.

For the level request mode, the transfer request (DREQ) must continue until $\overline{BGACK}$ is returned.

6.7.2.2.2 Auto Request

Request is set by the software for memory-to-memory transfers. As with external requests, DMA requests are initiated when the DMAE bits of the operation control register (OPCR) and channel control register (CHCR0) are set and the CHCR0 DST bit is set.

When the bus is captured, the transfer operation continues until the data transfer count register (DTCR) value becomes "0000H".

The DST bit is cleared when DTCR reaches "0000H" or this channel generates interrupt of errors.

6.7.2.3 Ending Transfers

6.7.2.3.1 Ending Service

The transfer of a block by a channel is ended when the value of the data transfer count register (DTCR) reaches "0000H". The DMA transfer operation can be ended by setting the OPC bit of the channel status register (CHSR), outputting the service end interrupt signal and reporting to the built-in interrupt controller (the INTE bits of OPCR and CHCR must be set).

DMA transfers are not made while the OPC bit is set. This bit can only be cleared by writing new data to DTCR.

6.7.2.3.2 Compulsory Ending

The DMA operation of the channel currently being executed can be ended and requests canceled after the end of a transfer cycle by asserting the $\overline{\text{DTEND}}$ signal at the external input pin.

Compulsory ending of a DMA operation using the $\overline{\text{DTEND}}$ signal is effective during burst transfers (this function is effective in the burst transfer mode for channel 0 memory-to-memory transfers).

6.7.2.4 Channel Priority Sequence

The DMA request channel priority is fixed in the order ch0 > ch1 > ch2. The channels, for which a request is made before the bus grant ($\overline{\text{BG}}$) is asserted by the core processor, are handled in the above priority order.

Requests for a high priority channel during a demand mode burst transfer for a low-priority channel are held until the transfer is completed.

In the case of demand mode cycle steal transfers, however, high priority channels become effective when a request is made, and the current transfer is interrupted and then restarted when the high-priority channel transfer is ended.

6.7.2.5 Built-in DMAC and External Bus Master Priority Order

The priority of the built-in DMAC is higher an external bus master.

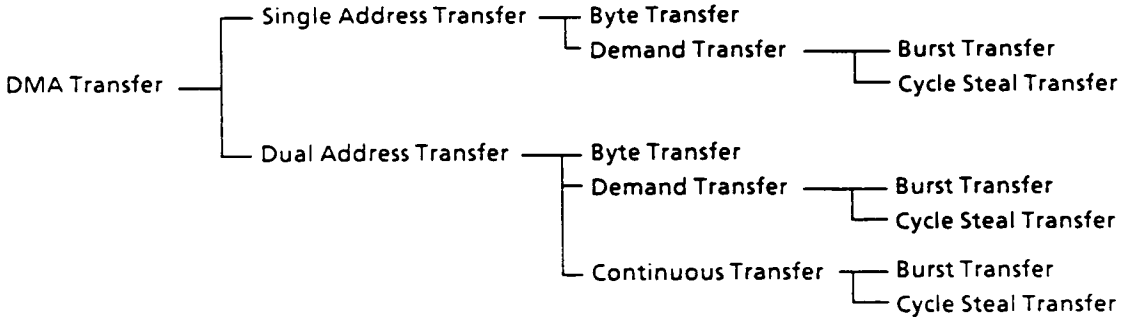
This priority order is fixed internally using the daisy chain method; therefore, an external bus master is held and $\overline{\text{BG}}$ is not asserted when the internal DMAC is executed.

6.7.2.6 Address Transfer Mode

There are two DMAC data transfer modes. One is the single address transfer mode in which memory read or write operation is made simultaneously with the assertion of the

$\overline{DACK}$ signal to the I/O device connected to the channel and thus, the transfer is made in one bus cycle.

The other is the dual address transfer mode. In this case, memory-to-memory (or memory - mapped I/O device) addressing is performed one operation at a time and transfers are made in two bus cycles. Data read from memory specified by a source address is stored in the DMAC temporary register.



6.7.2.6.1 Single Address Transfer Mode

This mode is effective for transfers between memory and I/O devices. Memory is accessed by the address but the I/O device is selected by the $\overline{DACK}$ signal from DMA, therefore, this mode can only be used for certain device connected to $\overline{DACK}$.

The transfer sequence is such that the transfer request signal $\overline{DREQ}$ is first asserted by the I/O device after the initial settings are completed and then bus arbitration is started. After the bus has been granted, the DMA operation is started and the $\overline{DACK}$ signal is asserted. A read or write cycle is then issued to memory at the same time. The read/write signal (R/$\overline{W}$) is used to control the memory; therefore, the R/$\overline{W}$ signal for I/O devices must be inverted by an external circuit.

Transfer requests are made in the external request mode.

6.7.2.6.2 Dual Address Transfer Mode

This mode is effective in memory-to-memory and memory-to-memory mapped I/O transfers, in which both devices are located in the memory address space for transfers between devices when addresses have been assigned to both.

DMAC transfers data in two steps (read cycle and write cycle). The data read are stored in the temporary register in the DMAC. The bus cannot be released between these two cycles. When a bus error occurs, the DMA cycle is ended immediately, even if the cycle is not completed.

Transfer requests are made in the auto request mode.

6.7.2.7 Transfer Mode

The bus modes are byte, cycle steal and burst transfer. In the byte transfer mode, only one DMA cycle is executed, even with the transfer request kept asserted. In the

cycle steal transfer mode, DMA transfers are made by alternating the core processor and DMAC bus cycles one at a time when the demand mode or continuous mode is set. In the burst transfer mode, the DMAC does not release the bus to the core processor and continues executing bus cycles until there is a transfer terminate request or the DTCR reaches "0000H".

6.7.2.7.1 Byte Transfer Mode

When a transfer request (regardless of the request mode) is sent to the DMAC in this mode, a DMA cycle begins after the bus is granted through bus arbitration, but ends always after one bus cycle and then, the bus is released to another bus master. Consequently, there is no DMA operation until the next transfer request is issued.

This mode cannot be used for memory-to-memory transfer.

6.7.2.7.2 Cycle Steal Transfer Mode

When a transfer request (regardless of the request mode) is sent to the DMAC in this mode, the transfer of one byte/word is made after the bus arbitration and then, the bus is always returned to the core processor. After, the core processor executes one bus cycle, the DMAC requests the bus again and repeats the DMA cycle as described above to make the transfer by alternating the bus.

6.7.2.7.3 Burst Transfer Mode

In this mode DMA cycles are executed when the DMAC is granted the bus from the core processor and continue until either the transfer end or transfer complete condition is satisfied. The bus will not be released to any other bus master during this time.

There are three cases of transfer end conditions as follows. One is for memory-to-memory transfers in which transfers continue until the data transfer count register (DTCR) reaches "0000H". The second is for transfers between memory and I/O devices with the demand transfer mode set. The DMA cycle continues as long as the $\overline{\text{DREQ}}$ signal is asserted but ends when $\overline{\text{DREQ}}$ is negated. The third is compulsory ending of the DMA cycle by asserting $\overline{\text{DTEND}}$ at an external pin.

6.7.2.7.4 Demand Transfer Mode

In this mode, transfers can only be made between memory and I/O devices.

When a transfer request is made the DMAC by asserting $\overline{\text{DREQ}}$ the DMA cycle is started after bus grant through the bus arbitration. The DMA cycle continues as long as $\overline{\text{DREQ}}$ is asserted.

This status, is the same as the burst transfers. When the cycle steal mode is set, however, the DMA cycle and processor cycle occur alternately. Negating $\overline{\text{DREQ}}$ ends the current DMA cycle and releases the bus to another bus master.

6.7.2.7.5 Continuous Transfer Mode

This mode is permitted only channel 0 for $\overline{\text{DREQ}}$ signal of demand transfer mode. Transfer request for the channel 0 starts at soft program setting. After all, this request is to set DSSH, CONTI and DST bit of CHCR0 and so DMA operation starts.

6.7.2.8 Issuing Interrupts

There are two sources for issuing DMAC interrupts. The first is the DMA service end interrupt signal issued when the data transfer count register (DTCR) reaches "0000H". The DMAC interrupt is sent to the built-in interrupt controller.

The second is error interrupts and there are a number of sources as follows.

(1) Bus error (with $\overline{\text{BERR}}$ input)

(2) Address error (word access with odd number address)

(3) Count error (transfer request with data transfer count register at 0000H)

6.7.2.9 Interface of I/O Devices

A possible I/O device of data transfer is a prepared acknowledge function device in single address transfer mode and an external memory or an external memory mapped I/O device in dual address transfer mode. Built-in DMAC can't transfer data for built-in peripheral device. In shows an external I/O device as follows ;

	Address Mode	Device Type	Request		Operand Size	Device Size	
			Channel	Mode		8 bit	16 bit
DMA Transfer	Single Address	Acknowledge I/O Divice	Ch0, 1, 2	External ($\overline{\text{DREQ}}$)	byte	○	-
					word	-	○
	Dual Address	Memory Mapped I/O Device	Ch0	External ($\overline{\text{DREQ}}$) Auto (DST bit)	byte	○	○
					word	-	○

6.7.2.10 Addressing and Data map of DMA Transferring

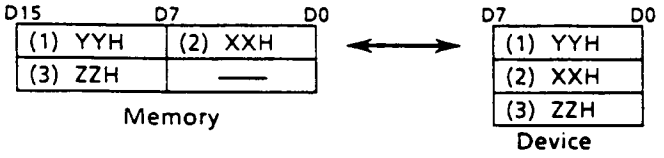
Increment of memory address for DMA data transferring is fixed with an establishment of each modes.

	Operand Size	Device Size	Memory Address increment		Device Address increment	
			Single	Dual	Single	Dual
DMA Transfer	byte	8 bit	+1	+1	-	+2
		16 bit	-	+1	-	+1
	word	16 bit	+2	+2	-	+2

for example an establishment of each modes as follow ;

① Single Address Transfer Mode : operand→byte, device→8-bit

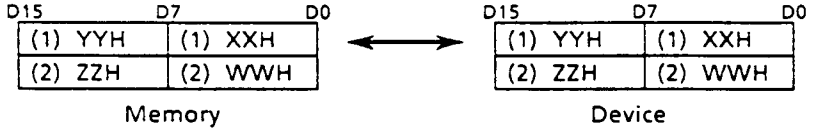
In the condition, I/O device is connected to upper or lower data bus so that when built-in DMAC reads data from memory, it is necessary to gather the data for connected to data bus. Therefor exchange the bus after checks $\overline{DACK}$ signal and put together $\overline{UDS}$ and $\overline{LDS}$ signal.



171189

② Single Address Transfer Mode : operand→word, device→16-bit

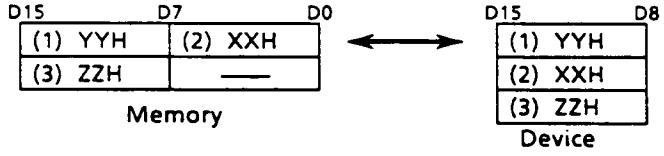
In this condition, I/O device and memory is connected to 16-bit data bus so that it isn't necessary to control for ①. However starting memory address is necessary to set even value.



171189

③ Dual Address Transfer Mode : operand→byte, device→8-bit

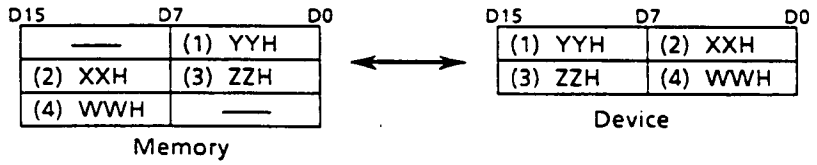
In this condition, I/O device is connected to upper or lower data bus like ①, but in this case built-in DMAC is stored read data to temporary register of the DMAC (read cycle) so that isn't necessary to exchange the data bus by external circuit (that is built-in DMAC exchange the bus data). Starting memory address is good even or odd value but starting I/O device address is need to set even or odd value for upper or lower connected data bus. For example the upper data bus is even value.



171189

④ Dual Address Transfer Mode : operand→byte, device→16-bit

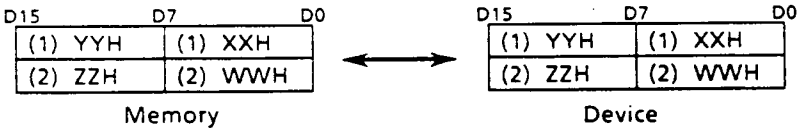
In this condition, I/O device and memory is connected to 16-bit data bus so that it doesn't have a restriction of mode establishment.



171189

⑤ Dual Address Transfer Mode : operand→word, device→16-bit

In this condition, I/O device and memory are connected to 16-bit data bus, but both the starting memory address and I/O device address need to set even value.



6.7.2.11 Reset Operation

After reset, built-in DMAC process as follow ;

① Register initial value

OPCR : 00H
CHCR : 0000H
CHSR : 0000H
SMAR : undefined
DMAR : undefined
MAR1-2 : undefined
DTCR0-2 : undefined

② When $\overline{\text{RESET}}$ signal asserts while built-in DMAC is transferring a data, the DMAC can't transfer a right data to release the bus. And all of the control signals negated after reset, DMA operation stopps.

Notes

1. If you will use channel 0 with single address transfer mode, utiliye only source memory address register (SMAR).
2. If you will write a command data for each register, at first set REW bit of operation control register (OPCR)
3. Don't access the register of built-in DMAC by DMA transfer operation.

6.7.3 Register Configuration

6.7.3.1 Operation Control Register (OPCR)

This register enables and disables the operation of all DMAC channels. After RESET, operation control register (OPCR) becomes 00H.

bit0 INTE : DMA Service Complete Interrupt Enable
bit1 DMAE : DMA Operation Enable
bit2 FC2 : DMA Function Code
bit3 RWE : DMA Register Read and Write Enable

	7	6	5	4	3	2	1	0
OPCR	×	×	×	×	RWE	FC2	DMAE	INTE

INTE : 0 – Interrupt Disable
1 – Interrupt Enable
DMAE : 0 – Operation Disable
1 – Operation Enable
FC2 : 0 – User Data Area
1 – Supervisor Data Area
RWE : 0 – Register Read and Write Disable
1 – Register Read and Write Enable

6.7.3.2 Channel Status Register (CHSR)

This register indicates the operating status of all DMAC channels. After RESET, channel status register (CHSR) becomes 0000H.

bit0~2 OPC0-2 : DMA Channel Service Operation Complete
bit3~5 REQ0-2 : DMA Channel Request
bit6~8 ERR0-2 : DMA Channel Error
bit9~10 ERRCD00-01 : Ch0 Error Code
bit11~12 ERRCD10-11 : Ch1 Error Code
bit13~14 ERRCD20-21 : Ch2 Error Code

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHSR	BERW	ERRCD 21	ERRCD 20	ERRCD 11	ERRCD 10	ERRCD 01	ERRCD 00	ERR2	ERR1	ERR0	REQ2	REQ1	REQ0	OPC2	OPC1	OPC0

OPC0-2 : 0 – DMA Channel Service Operation Not Complete
1 – DMA Channel Service Operation Complete
REQ0-2 : 0 – DMA Channel Request Not Pending
1 – DMA Channel Request Pending
ERR0-2 : 0 – DMA Channel Not Error
1 – DMA Channel Error
ERRCD00-01 : 01 – Ch0 Bus Error
10 – Ch0 Address Error
11 – Ch0 Count Error
ERRCD10-11 : 01 – Ch1 Bus Error
10 – Ch1 Address Error
11 – Ch1 Count Error
ERRCD20-21 : 01 – Ch2 Bus Error
10 – Ch2 Address Error
11 – Ch2 Count Error
BERW : 0 – ch0 Bus Error at write cycle
1 – ch0 Bus Error at read cycle

6.7.3.3 Channel Control Register 0 (CHCR0)

This register controls DMA channel 0 operation and sets the modes. After RESET, channel control register 0 (CHCR0) becomes 0000H.

bit0	INTE0	:	DMA Ch0 Service Complete and Error Interrupt Enable
bit1	DMAE0	:	DMA Ch0 Operation Enable
bit2	RSS0	:	Ch0 Request Signal Sense Select
bit3	TMOD00	:	Ch0 Byte / Demand Transfer Select
bit4	TMOD01	:	Ch0 Cycle Steal / Burst Transfer Select
bit5~6	TDIR00~01	:	Ch0 Data Transfer Direction Select
bit7	DSSH	:	DMA Request Trigger Select
bit8	CONT1	:	Continuous Transfer Enable
bit9	DST	:	DMA Request Start Enable
bit10	OPW0	:	Ch0 Operand Width Select
bit11	DPW0	:	Ch0 Device Width Select
bit12	MAC0	:	Ch0 Memory Address Counter Enable
bit13	ERC0	:	Ch0 Error Flag Clear

CHCR0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	ERC0	MAC 0	DPW 0	OPW 0	DST	CONT 1	DSSH	TDIR 01	TDIR 00	TMOD 01	TMOD 00	RSS0	DMAE 0	INTE0

INTE0	:	0	–	Ch0 Interrupt Disable
		1	–	Ch0 Interrupt Enable
DMAE0	:	0	–	Ch0 Operation Disable
		1	–	Ch0 Operation Enable
RSS0	:	0	–	Ch0 Request Signal Edge
		1	–	Ch0 Request Signal Level
TMOD00	:	0	–	Ch0 Byte Transfer
		1	–	Ch0 Demand Transfer
TMOD01	:	0	–	Ch0 Cycle Steal Transfer
		1	–	Ch0 Burst Transfer
TDIR00~01	:	00	–	Ch0 Memory → I/O Direction
		01	–	Ch0 I/O → Memory Direction
		10	–	Ch0 Memory → Memory Direction
DSSH	:	0	–	DMA Request Start Hard (DREQ0) Select
		1	–	DMA Request Start Soft (Program) Select
CONTI	:	0	–	DMA Service Continuous Disable
		1	–	DMA Service Continuous Enable
DST	:	0	–	DMA Request Start Soft Disable
		1	–	DMA Request Start Soft Enable
OPW0	:	0	–	Ch0 Operand Width 8bit
		1	–	Ch0 Operand Width 16bit
DPW0	:	0	–	Ch0 Device Port Width 8bit
		1	–	Ch0 Device Port Width 16bit
MAC0	:	0	–	Ch0 Memory Address Count Up Disable
		1	–	Ch0 Memory Address Count Up Enable
ERC0	:	0	–	No Operation
		1	–	CHSR ERRCD00-01 and ERR0 Bits Clear

6.7.3.4 Channel Control Registers 1 - 2 (CHCR1-2)

These registers control DMA channel 1 and 2 operations and set the modes. After RESET, channel control register 1 and 2 (CHCR1-2) becomes 0000H.

bit0	INTE1/2	:	DMA Ch1/2 Service Complete and Error Interrupt Enable
bit1	DMAE1/2	:	DMA Ch1/2 Operation Enable
bit2	TDIR1/2	:	Ch1/2 Data Transfer Direction Select
bit3	RSS1/2	:	Ch1/2 Request Signal Sense Select
bit4	TMOD10/20	:	Ch1/2 Byte / Demand Transfer Select
bit5	TMOD11/21	:	Ch1/2 Cycle Steal / Burst Transfer Select
bit6	OPW1/2	:	Ch1/2 Operand Width Select
bit7	DPW1/2	:	Ch1/2 Device Port Width Select
bit8	MAC1/2	:	Ch1/2 Memory Address Counter Enable
bit9	ERC1/2	:	Ch1/2 Error Flag Clear

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHCR1	0	0	0	0	0	0	ERC1	MAC 1	DPW 1	OPW 1	TMOD 11	TMOD 10	RSS1	TDIR1	DMAE 1	INTE1
CHCR2	0	0	0	0	0	0	ERC2	MAC 2	DPW 2	OPW 2	TMOD 21	TMOD 20	RSS2	TDIR2	DMAE 2	INTE2

INTE1/2	: 0	–	Ch1/2 Interrupt Disable
	1	–	Ch1/2 Interrupt Enable
DMAE1/2	: 0	–	Ch1/2 Operation Disable
	1	–	Ch1/2 Operation Enable
TDIR1/2	: 0	–	Ch1/2 Memory → I/O Direction
	1	–	Ch1/2 I/O → Memory Direction
RSS1/2	: 0	–	Ch1/2 Request Signal Edge
	1	–	Ch1/2 Request Signal Level
TMOD10/20	: 0	–	Ch1/2 Byte Transfer
	1	–	Ch1/2 Demand Transfer
TMOD11/21	: 0	–	Ch1/2 Cycle Steal Transfer
	1	–	Ch1/2 Burst Transfer
OPW1/2	: 0	–	Ch1/2 Operand Width 8bit
	1	–	Ch1/2 Operand Width 16bit
DPW1/2	: 0	–	Ch1/2 Device Port Width 8bit
	1	–	Ch1/2 Device Port Width 16bit
MAC1/2	: 0	–	Ch1/2 Memory Address Count Up Disable
	1	–	Ch1/2 Memory Address Count Up Enable
ERC1/2	: 0	–	No Operation
	1	–	CHSR ERRCD10-11/20-21 and ERR1/2 Bits Clear

6.7.3.5 Source Memory Address Register (SMAR)

This is a 24-bit register used to store DMA channel 0 source memory addresses.
After RESET, source memory address register (SMAR) is undefined.

bit0-23 SA0-23 : Source Memory Address

SMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	×	×	×	×	SA23	SA22	SA21	SA20	SA19	SA18	SA17	SA16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SA15	SA14	SA13	SA12	SA11	SA10	SA9	SA8	SA7	SA6	SA5	SA4	SA3	SA2	SA1	SA0

6.7.3.6 Destination Memory Address Register (DMAR)

This is a 24-bit register used to store DMA channel 0 destination memory addresses.
After RESET, destination memory address register (DMAR) is undefined.

bit0-23 DA0-23 : Destination Memory Address

DMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	×	×	×	×	DA23	DA22	DA21	DA20	DA19	DA18	DA17	DA16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DA15	DA14	DA13	DA12	DA11	DA10	DA9	DA8	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0

6.7.3.7 Memory Address Registers 1 - 2 (MAR1-2)

These are 24-bit registers used to store DMA channels 1 and 2 memory addresses (source or destination).
After RESET, memory address registers (MAR1-2) is undefined.

bit0-23 A0-23 : Memory Address

MAR1-2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	×	×	×	×	A23	A22	A21	A20	A19	A18	A17	A16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

6.7.3.8 Data Transfer Count Registers 0 - 2 (DTCR0-2)

These registers specify the number of DMA channels byte data transfers.
After RESET, data transfer count registers (DTCR0-2) is undefined.

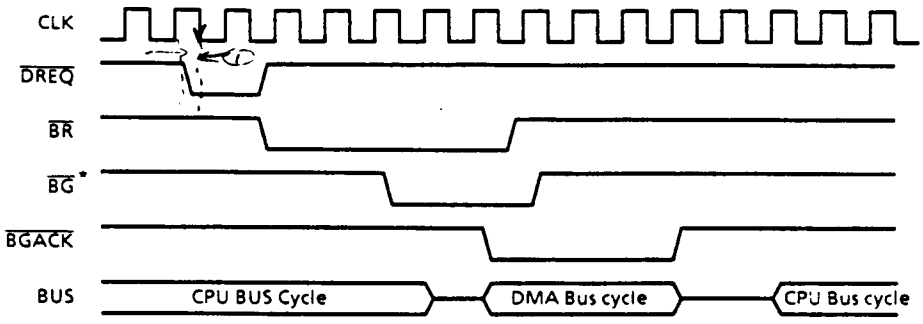
bit0-15 DT0-15 : Data Transfer Count

DTCR0-2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DT15	DT14	DT13	DT12	DT11	DT10	DT9	DT8	DT7	DT6	DT5	DT4	DT3	DT2	DT1	DT0

6.7.4 Bus Cycle Timing Chart

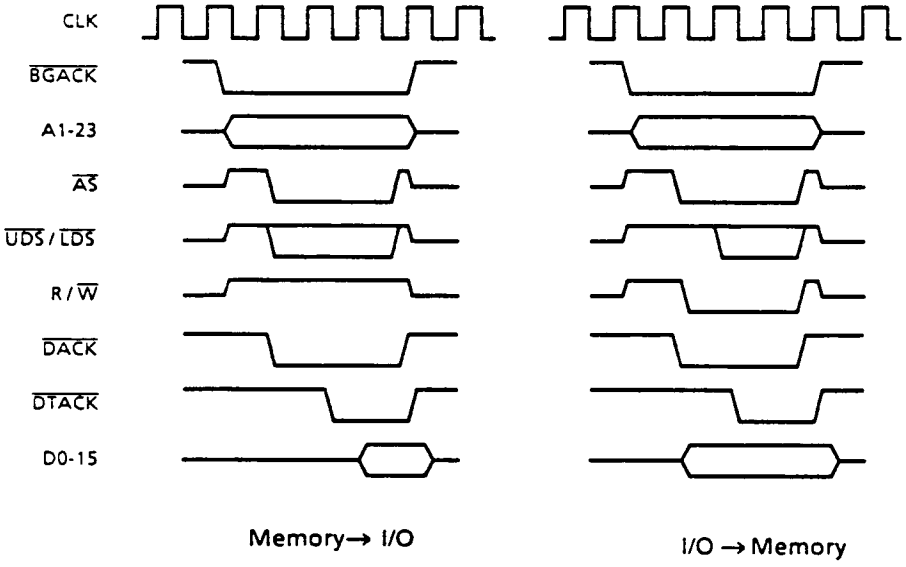
6.7.4.1 DMAC Bus Arbitration



* : This $\overline{BG}$ signal is output of the core processor

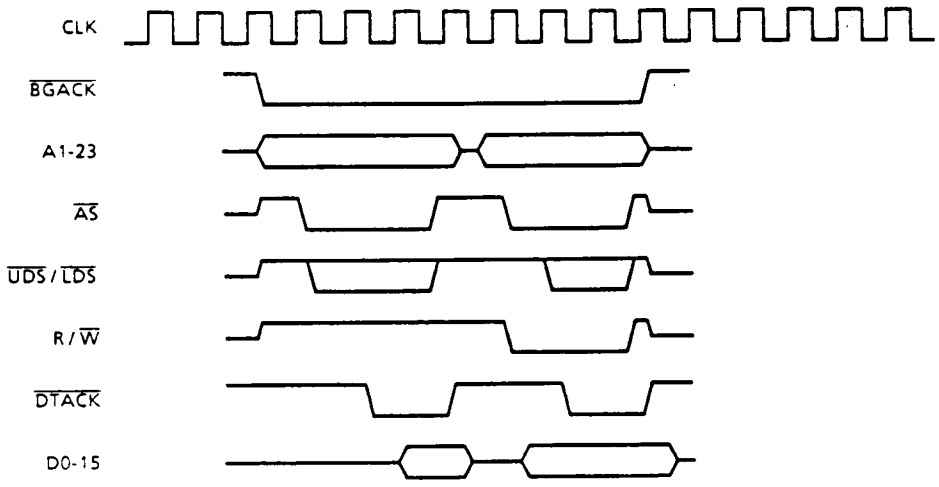
201189

6.7.4.2 Single Address Transfer



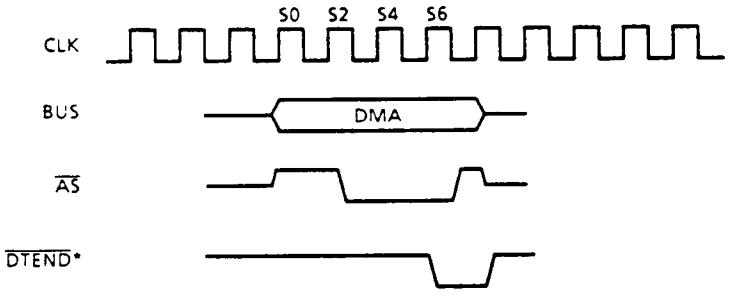
151289

6.7.4.3 Dual Address Transfer (Memory → Memory)



201189

6.7.4.4 Data Transfer Complete Cycle

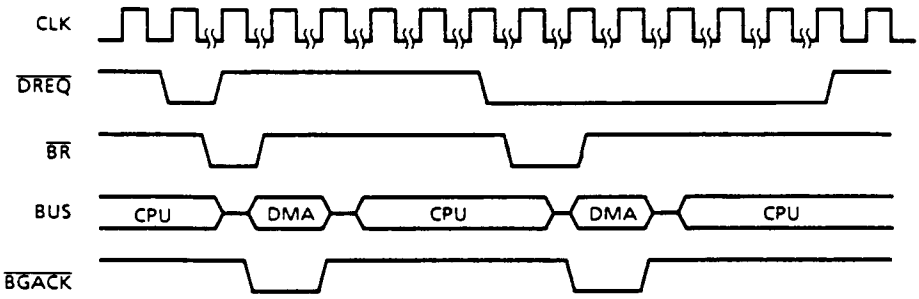


* Open Drain Output

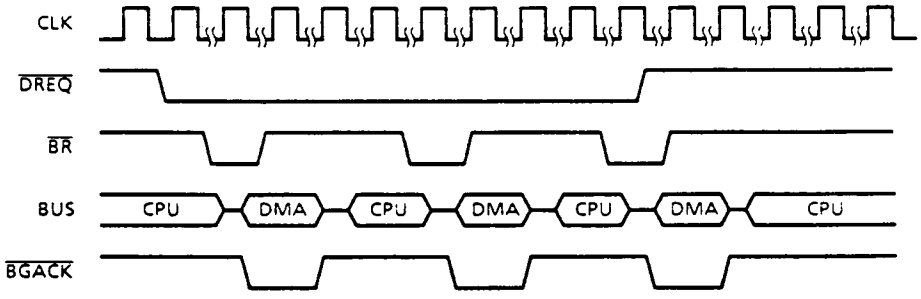
151289

6.7.4.5 Single Address Transfer

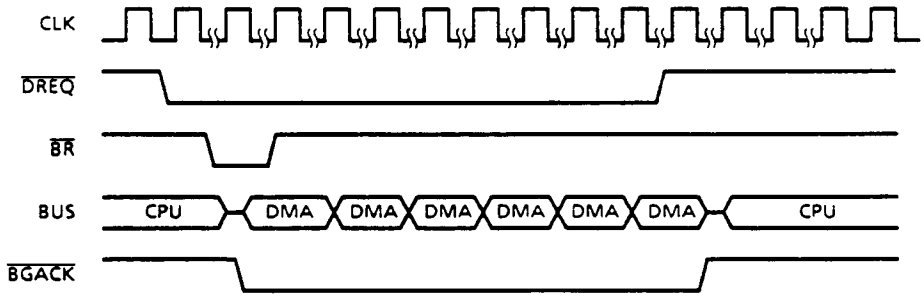
6.7.4.5.1 Byte Transfer



6.7.4.5.2 Cycle Steal Transfer (Demand Mode)

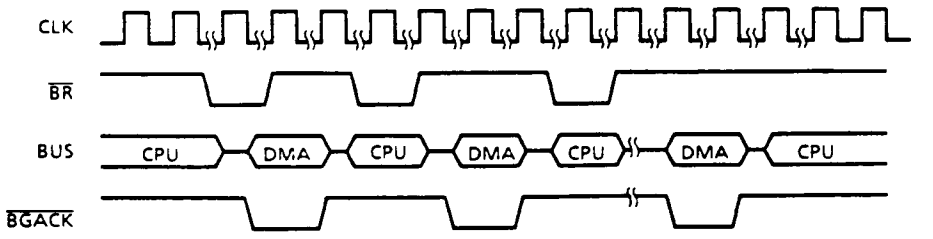


6.7.4.5.3 Burst Transfer (Demand Mode)

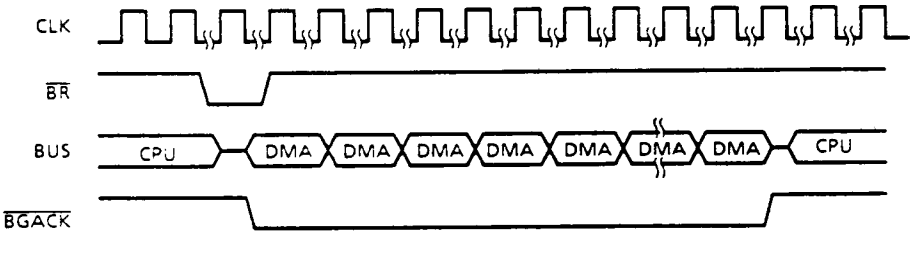


6.7.4.6 Dual Address Transfer

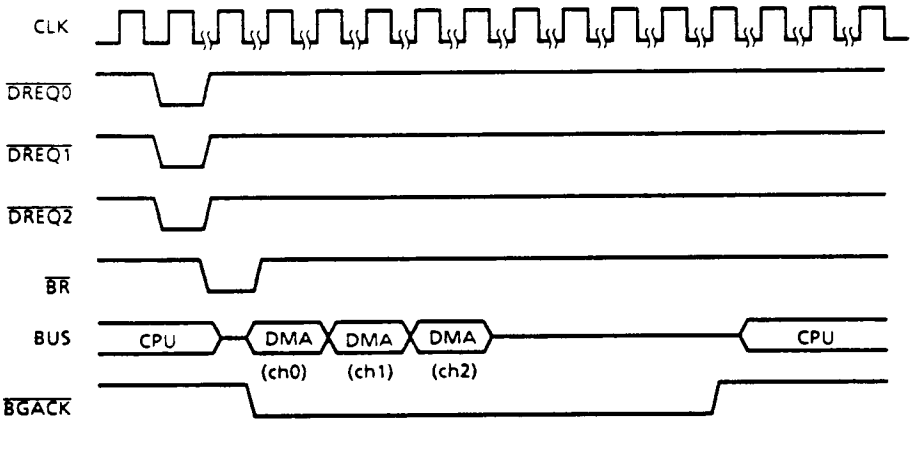
6.7.4.6.1 Cycle Steal Transfer (Continuous Mode)



6.7.4.6.2 Burst Transfer (Continuous Mode)

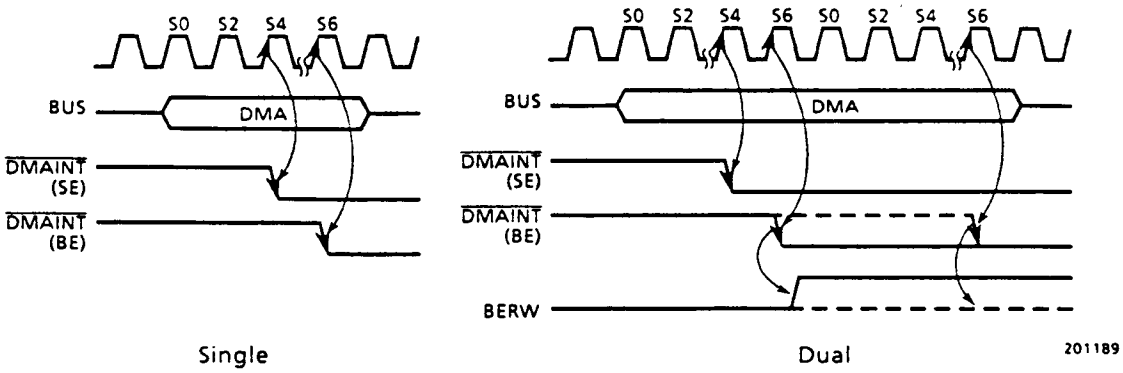


6.7.4.7 Multiple Request

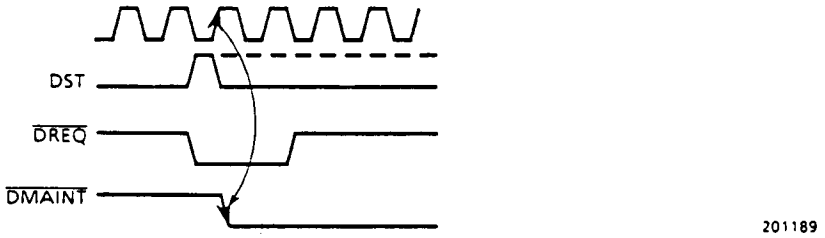


6.7.4.8 Interrupt Request Timing

6.7.4.8.1 Data Transfer Complete / Bus Error

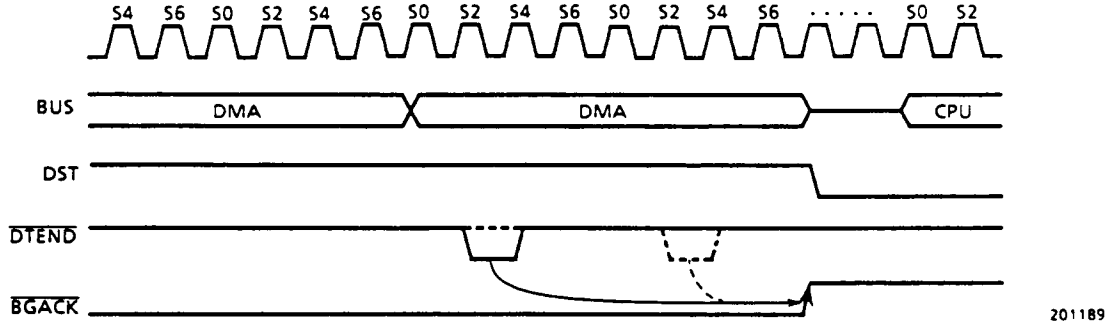


6.7.4.8.2 Address Error / Count Error



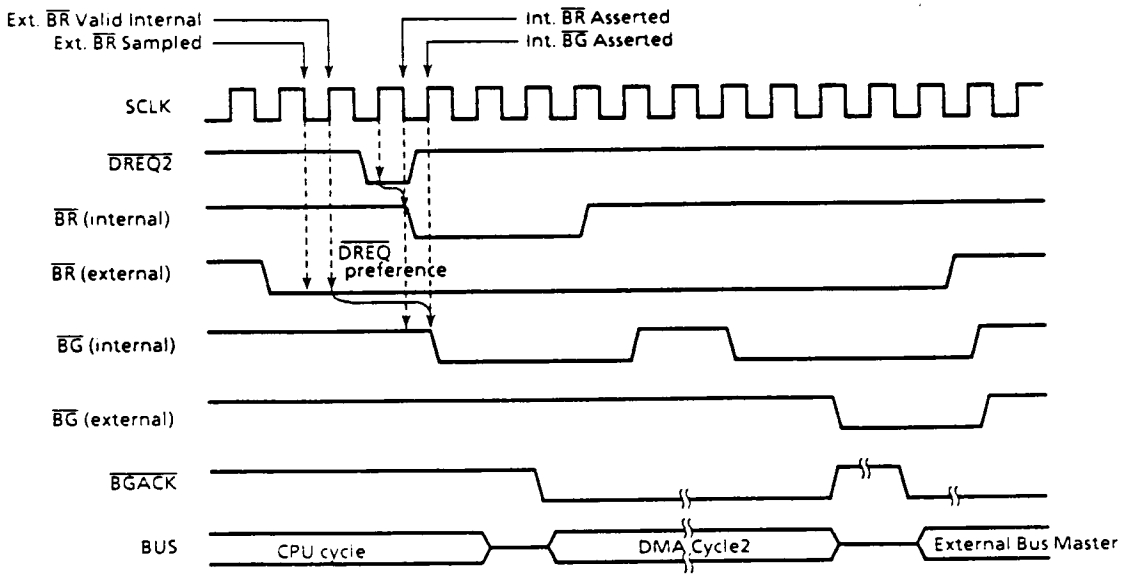
When error of DMAC generated, DST bit is cleared

6.7.4.9 Compulsory Ending



Compulsory ending operation of dual address transfer

6.7.4.10 Puiority sequence for external $\overline{BR}$



※ Priority determination timing for external $\overline{BR}$ judges output timming of internal $\overline{BR}$ and output timming of internal $\overline{BG}$.
But internal $\overline{BR}$ signal output fall edge of clock, and internal $\overline{BG}$ signal output rise edge of clock so that when this $\overline{DREQ2}$ signal is sampled at next clock, external $\overline{BR}$ has preference for $\overline{DREQ2}$.

201189

6.7.5 DMAC Internal Register Map

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Channel Control Register 0	($280)	×	×	ERC0	MAC0	DPW0	OPW0	DST	CONT0	DSSH	TDIR 01	TDIR 00	TMOD 01	TMOD 00	RSSO	DMAE 0	INTE0	($281)
Data Transfer Count Register 0	($282)	DT15	DT14	DT13	DT12	DT11	DT10	DT9	DT8	DT7	DT6	DT5	DT4	DT3	DT2	DT1	DT0	($283)
										SA23	SA22	SA21	SA20	SA19	SA18	SA17	SA16	($285)
Source Memory Address Register 0	($286)	SA15	SA14	SA13	SA12	SA11	SA10	SA9	SA8	SA7	SA6	SA5	SA4	SA3	SA2	SA1	SA0	($287)
										DA23	DA22	DA21	DA20	DA19	DA18	DA17	DA16	($289)
Destination Memory Address Register 0	($28A)	DA15	DA14	DA13	DA12	DA11	DA10	DA9	DA8	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	($288)
										×	×	×	×	RWE	FC2	DMAE	INTE	($28D)
Channel Status Register	($28E)	BERW	ERRCD 21	ERRCD 20	ERRCD 11	ERRCD 10	ERRCD 01	ERRCD 00	ERR2	ERR1	ERR0	REQ2	REQ1	REQ0	OPC2	OPC1	OPC0	($28F)
Channel Control Register 1	($290)	×	×	×	×	×	×	ERC1	MAC 1	DPW1	OPW1	TMOD 11	TMOD 10	RSS1	TDIR1	DMAE 1	INTE1	($291)
Data Transfer Count Register 1	($292)	DT15	DT14	DT13	DT12	DT11	DT10	DT9	DT8	DT7	DT6	DT5	DT4	DT3	DT2	DT1	DT0	($293)
										A23	A22	A21	A20	A19	A18	A17	A16	($295)
Memory Address Register 1	($296)	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	($297)
Channel Control Register 2	($2AQ)	×	×	×	×	×	×	ERC2	MAC 2	DPW2	OPW2	TMOD 21	TMOD 20	RSS2	TDIR2	DMAE 2	INTE2	($2A1)
Data Transfer Count Register 2	($2A2)	DT15	DT14	DT13	DT12	DT11	DT10	DT9	DT8	DT7	DT6	DT5	DT4	DT3	DT2	DT1	DT0	($2A3)
										A23	A22	A21	A20	A19	A18	A17	A16	($2A5)
Memory Address Register 2	($2A6)	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	($2A7)

6.8 DRAM CONTROLLER

6.8.1 Overview

This DRAM controller generates the control signals ($\overline{RAS}$, $\overline{CAS}$) necessary for interfacing with DRAMs.

Main features

Dummy cycle generate

Refresh method: $\overline{CAS}$ before $\overline{RAS}$ refresh

Refresh interval: 80 - 288 states selectable

Refresh cycle length: 2 - 9 states selectable

Memory access cycle length: 2 - 9 states selectable

Memeory access address length : 9, 10 bit length selectable

6.8.2 Block Diagram

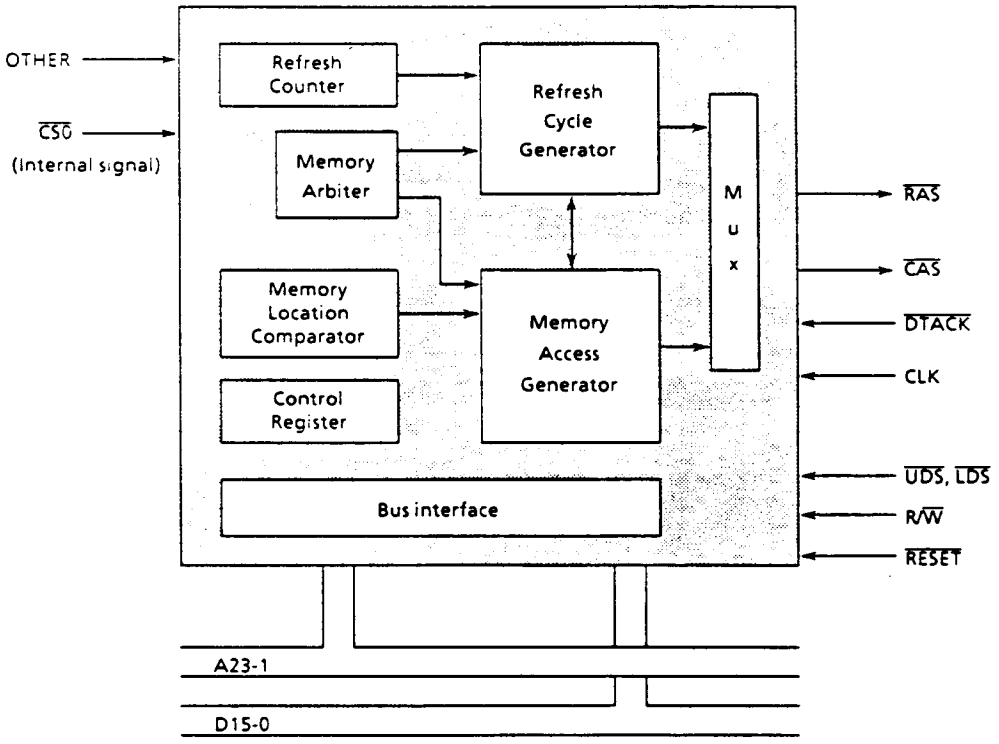


Figure 6.61 Block Diagram

6.8.3 DRAM Initialization

The DRAM controller can issues $\overline{CAS}$ before $\overline{RAS}$ dummy cycles. Dummy cycles are controlled by bit 7 (RINI) of the refrech control register

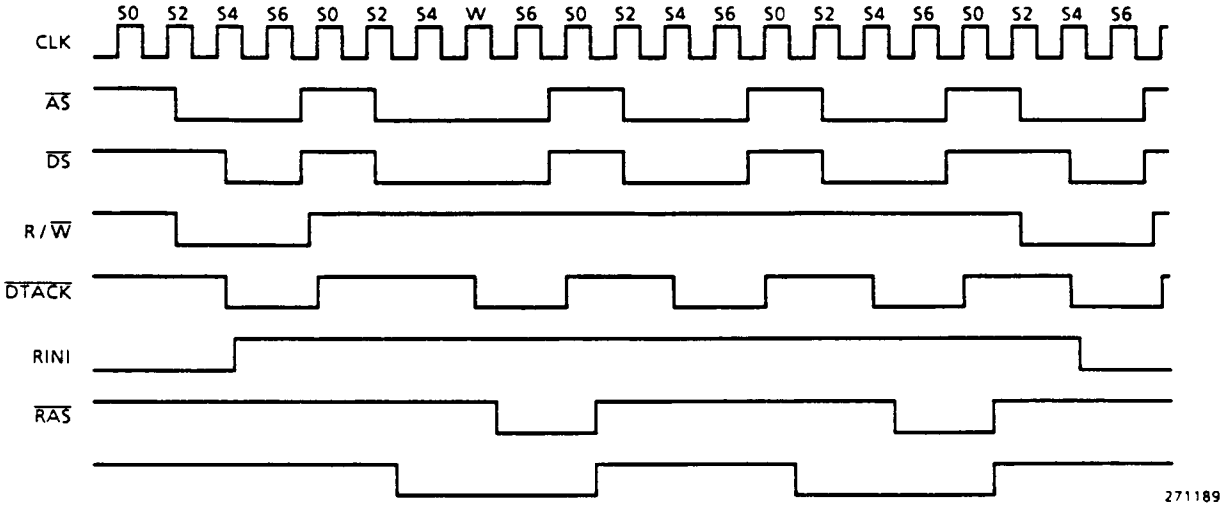


Figure 6.62

6.8.4 DRAM Refresh

The DRAM controller can issue refresh cycles at intervals set in the refresh control register. Refresh cycles of 80, 112, 144, 176, 208, 224, 256 or 288 states can be selected. Refresh cycle lengths of 2 to 9 states can also be selected.

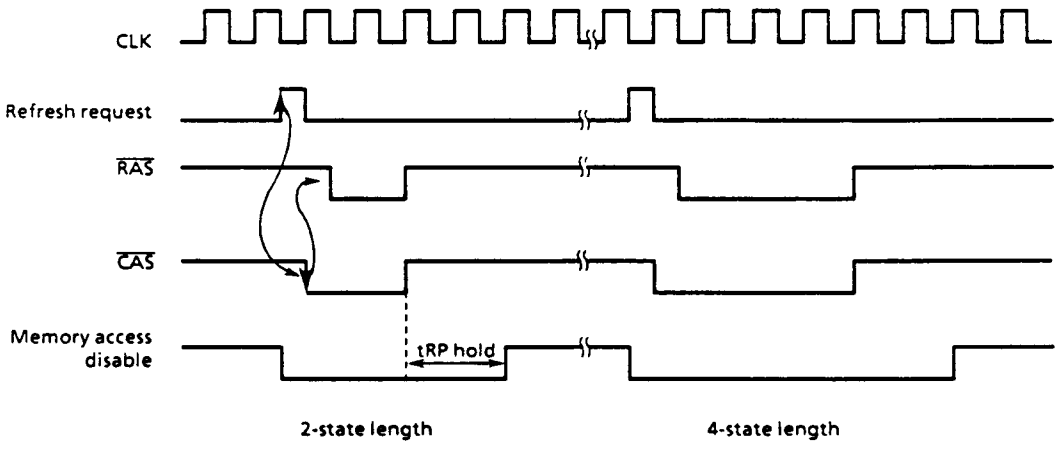


Figure 6.63

6.8.5 DRAM Read/Write

The address decoder asserts the DRAM controller when memory area 0 specified by the address decoder is accessed. If bit 0 (memory access enable/disable bit) of the memory control register is set at this time, the DRAM controller outputs valid signals. 0 to 7 wait states can be inserted to the $\overline{DTACK}$ signal by setting the address decoder. $\overline{DTACK}$ generated by an external circuit can also be used.

The DRAM controller can issue ROW and COLUMN address to A1~A9 (A10) at read/write cycle.

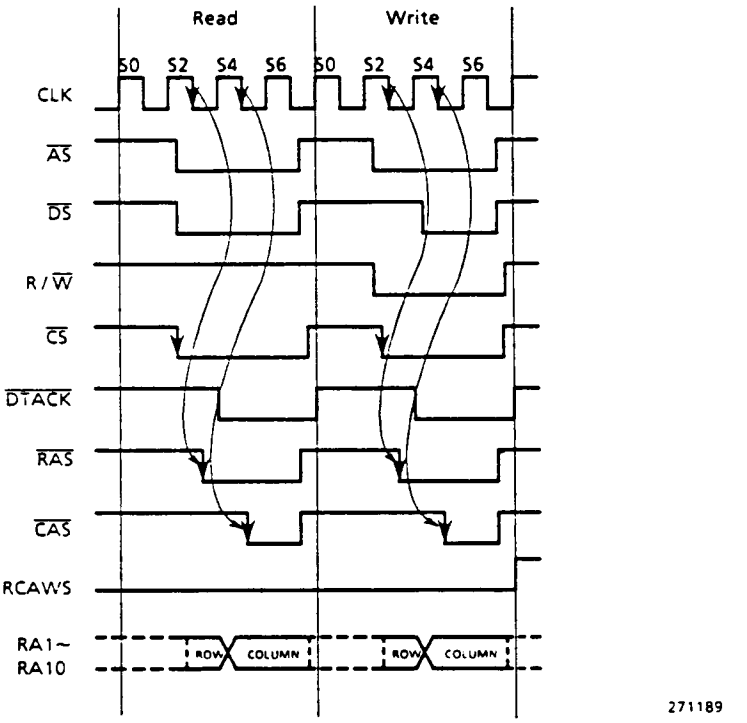


Figure 6.64

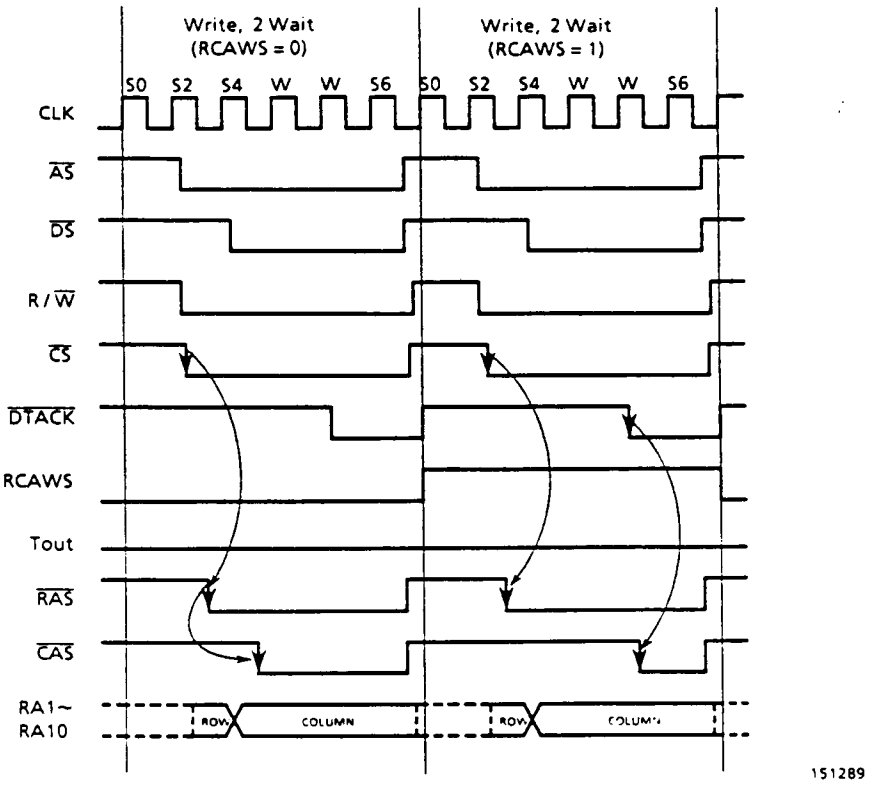


Figure 6.65

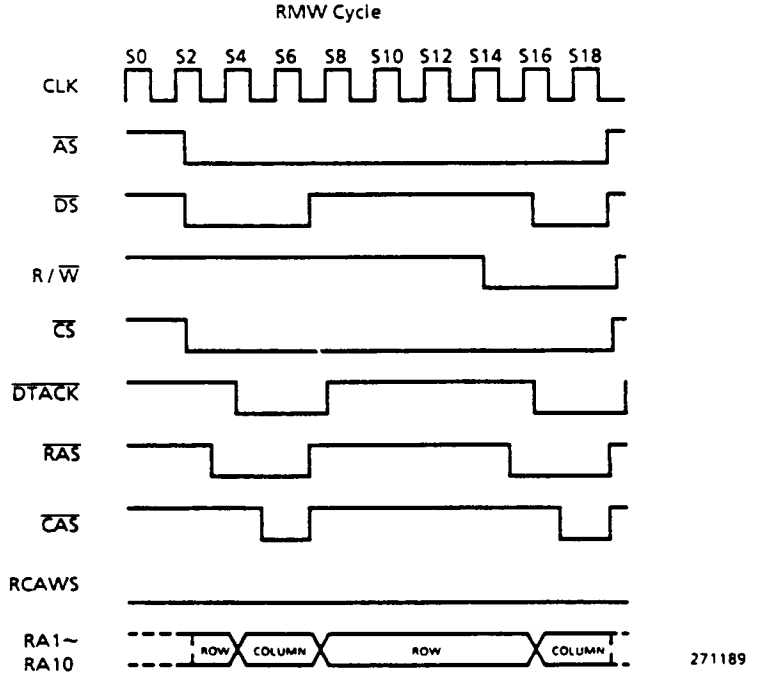


Figure 6.66

6.8.6 Priority Sequence

Since refresh requests to DRAM are not synchronized with the CPU bus cycle, read/write requests from the CPU to DRAM can overlap with the refresh request. In such cases, the DRAM controller assigns priority to the request asserted first. Also, when such requests are issued simultaneously , refreshing has the higher priority.

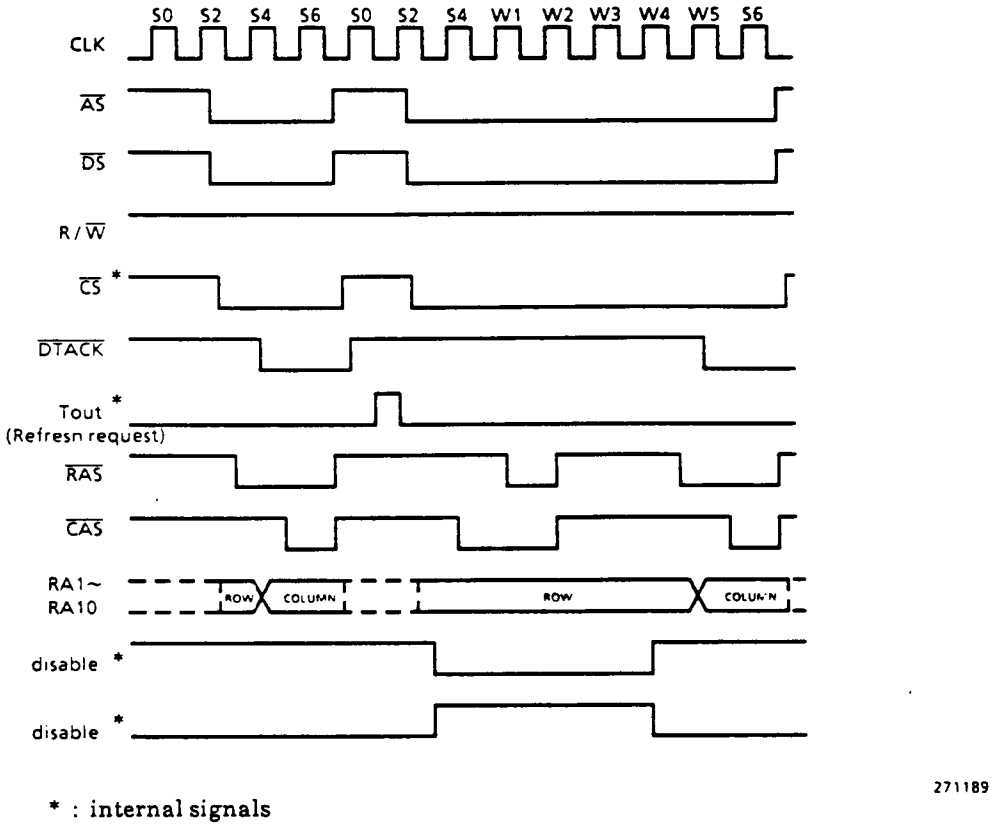


Figure 6.67

6.8.7 Register Configuration

6.8.7.1 Refresh Control Register

This register controls the refresh operation and becomes 00H after a reset.

Refresh control register ($XXX301)

7	6	5	4	3	2	1	0
RINI	RS2	RS1	RS0	RW2	RW1	RW0	RC

bit 0 : refresh enable (1)/disable (0)
bits 1 - 3 : refresh wait cycle (2 - 9 states)
bits 4 - 6 : refresh cycle (80 - 288 states)
bit 7 : dummy cycle enable (1)/disable (0)

6.8.7.2 Memory Control Register

This register controls DRAM read and write operations and becomes 04H after a reset.

Memory control register ($XXX303)

7	6	5	4	3	2	1	0
-	-	-	-	RCAWS	ADR10	ADR9	MC

bit 0 : memory access enable (1)/disable (0)
bit 1 : address length select 9-bit length (1) / disable (0)
bit 2 : address length select 10-bit length (1) / disable (10)
bit 3 : $\overline{CAS}$ assert timming select $\overline{DTACK}$ synchronize (1) / normal (0)

6.8.7.3 Refresh Insertion Interval

Refresh cycle			Insertion interval (state)	Operational frequency				
RS2	RS1	RS0		8MHz	10MHz	12MHz	14MHz	16MHz
0	0	0	80	10	8	6.67	5.71	5
0	0	1	112	(14)	11.2	9.33	8	7
0	1	0	144	18	(14.4)	12	10.29	9
0	1	1	176	22	17.6	(14.67)	12.57	11
1	0	0	208	26	20.8	17.33	(14.86)	13
1	0	1	224	28	22.4	18.67	16	(14)
1	1	0	256	32	25.6	21.33	18.29	16
1	1	1	288	36	28.8	24	20.57	18

(Unit : μsec)

6.9 STEPPING MOTOR CONTROLLER

6.9.1 Over view

There are two 4-bit hardware stepping motor control ports (SMC ports M0 and M1) that synchronize with the 8-bit timers. These SMC ports M0 and M1 are multiplexed with the I/O ports P0 and P1.

The channel 0 (M0) synchronizes with the trigger signal of the Timer Flip-flop TFF3, and the channel 1 (M1) is synchronous with that of the Timer Flip-flop TFF4.

The SMC ports are controlled by three control registers (P01CR, SMMOD, and SMCR), and allow the selection of driving method: 1) 4-phase 1-step/2-step excitation and 2) 4-phase 1-2 step excitation.

6.9.1.1 Feature

2 hardware stepping motor control port
4-phase motor interface (select 1/2/1-2 step excitation)
Normal / Reverse rotation control
Connected internal 8-bit Timers (2ch)

6.9.2 Control Registers

6.9.2.1 Port 0 and Port 1 I/O Selection Register (P01CR)

This register specifies either input or output for each bit of the 4-bit I/O ports 0 and 1.

When all bits of P01CR are initialized to "0" by resetting, Ports 0 and 1 function as input ports.

P01CR is a write-only register (not for readout).

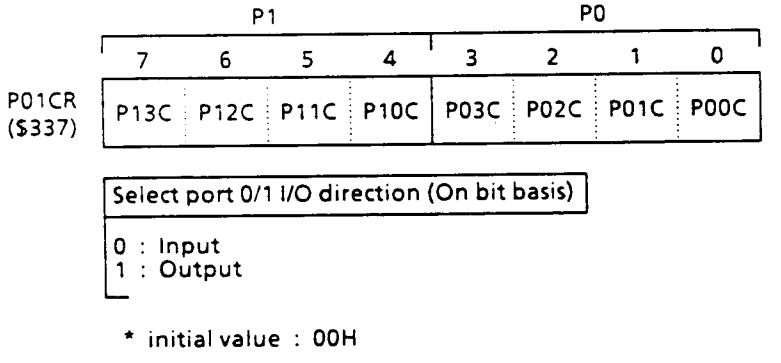


Figure 6.68 Ports 0 and 1 I/O Selection Registers

6.9.2.2 Stepping Motor Control Port Mode Register (SMMOD)

Ports 0 and 1 also function as SMC Ports (M0 and M1) or Timer Output Ports (Tout3 and Tout4), as selected by SMMOD1, 0 or SMMOD5, 4.

When Port 0 is used as SMC Port (M0), SMMOD1, 0 should be set to 10 or 11.

When Port 1 is used as SMC Port (M1), SMMOD5, 4 should be set to 10 or 11.

SMMOD2 and SMMOD6 serve to select the excitation method. With "0" the full-step (1-step/2-step) excitation is selected and with "1" the half-step (1-2 step) excitation is selected. When full-step excitation is selected, the selection of either 1 or 2-step excitation is determined by the initial output value.

SMMOD3 and SMMOD7 should be set to "1" when those ports are used as SMC Ports.

	P1				P0			
	7	6	5	4	3	2	1	0
SMMOD ($331)	SM1M		P1C		SM0M		P0C	

P0C : Select Port 0 function

	P03	P02	P01	P00	SMC Trigger signal
00	IN/OUT	IN/OUT	IN/OUT	IN/OUT	–
01	IN/OUT	IN/OUT	IN/OUT	IN/Tout3	–
10	IN/M03	IN/M02	IN/M01	IN/M00	TFF3
11					

P1C : Select Port 1 function

	P13	P12	P11	P10	SMC Trigger signal
00	IN/OUT	IN/OUT	IN/OUT	IN/OUT	–
01	IN/OUT	IN/OUT	IN/OUT	IN/Tout4	–
10	IN/M13	IN/M12	IN/M11	IN/M10	TFF4
11					

IN : Input port OUT : Output port Tout3/Tout4 : Timer output port

M00-3/10-3 : Stepping motor control port 0/1

SMOM/SM1M : Stepping motor control port 0/1 mode

00	—
01	—
10	4-phase 1or 2 excitation (Full-step)
11	4-phase 1-2 excitation (Half-step)

* initial value : 00H

** bit 3 and 7 always set at "1" when read out

151289

Figure 6.69 Stepping Motor Control Port Mode Register

6.9.2.3 Stepping Motor Control Port Rotating Direction Control Register (SMCR)

This register controls the rotating direction. The direction of the channel 0 (M0) is set by SMCR0 (CCW0) and that of the channel 1 (M1) is set by SMCR4 (CCW1).

"0" denotes normal rotation and "1" denotes reverse rotation.

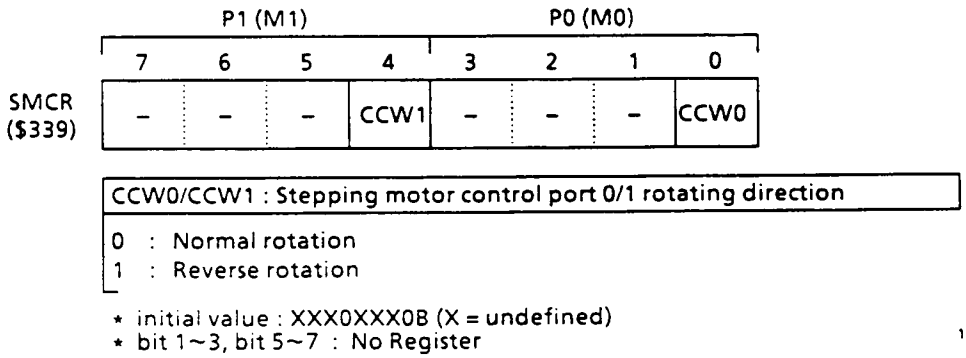
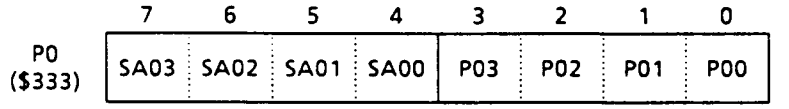


Figure 6.70 Stepping Motor Control Port Rotating Direction Control Register

6.9.1.4 Port 0

This is a 4-bit I/O port allocated to the address $333.

The lower four bits are assigned as Port 0, while the upper four bits function as the shifter alternate register (SA0) for driving the stepping motor with the half-step (1-2) excitation.



P00-3 : Port 0
SA00-3 : Shifter alternate register 0 for stepping motor control

* initial value : X0H (X = undefined)
** SA00-3 always set at "FH" when read out

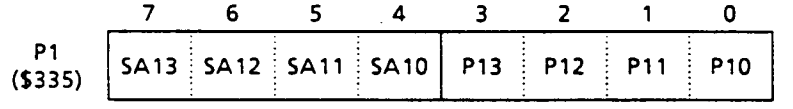
151289

Figure 6.71 Port 0

6.9.1.5 Port 1

This is a 4-bit I/O port allocated to the address $335.

The lower four bits are assigned as Port 1, while the upper four bits function as the shifter alternate register (SA1) for driving the stepping motor with the half-step (1-2) excitation.



P10-3 : Port 1
SA10-3 : Shifter alternate register 1 for stepping motor control

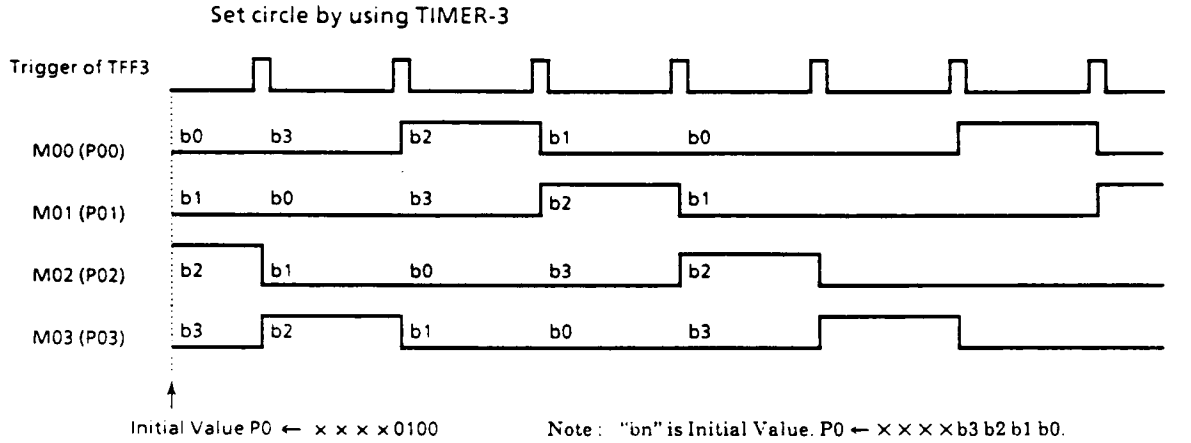
* initial value : X0H (X = undefined)
** SA10-3 aslways set at "FH" when read out

201289

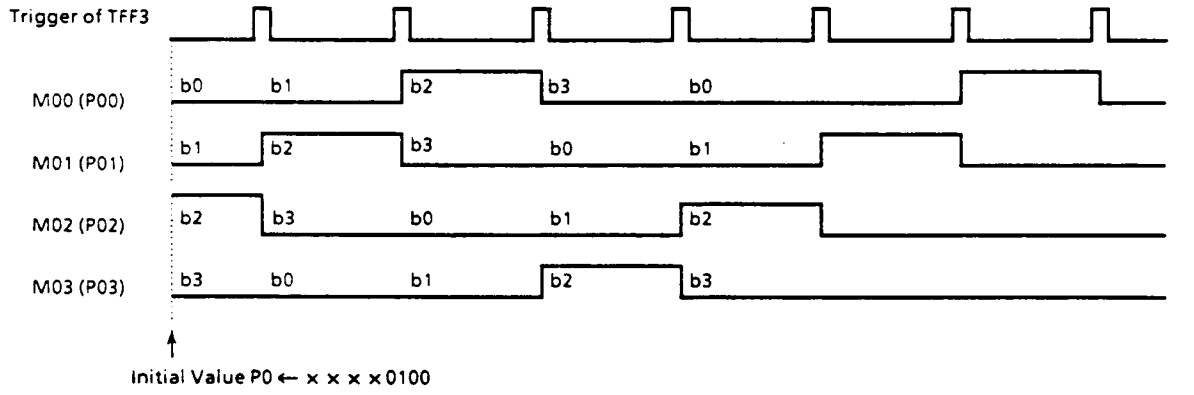
Figure 6.72 Port 1

6.9.3 4-phase 1-step/2-step Excitation

Figure 6.73 and 74 show the output waveforms of 4-phase 1-step excitation and 2-step excitation when Channel 0 is selected.



① Normal Rotation



② Reverse Rotation

Figure 6.73 Output Waveforms of 4-phase 1-step Excitation (Normal Rotation and Reverse Rotation)

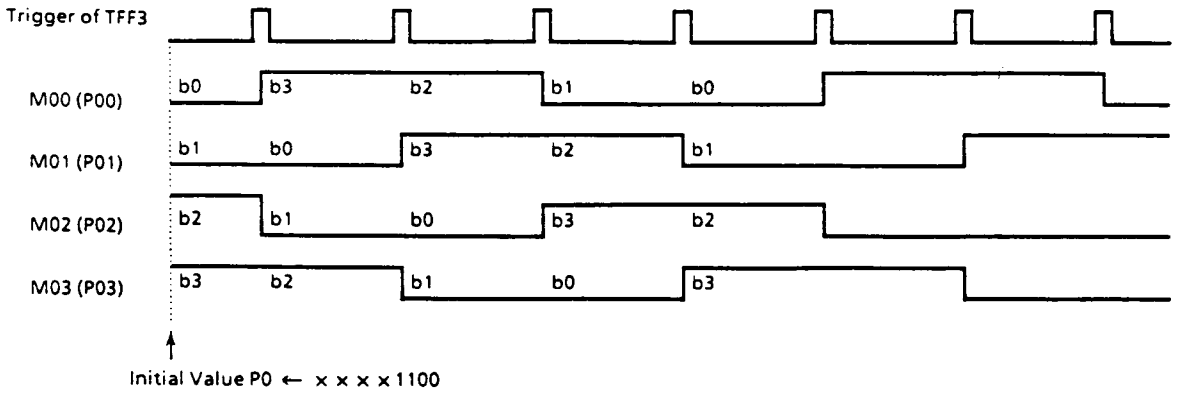


Figure 6.74 Output Waveforms of 4-phase 2-step Excitation (Normal Rotation)

When Channel 0 is selected, for example, the stepping motor control port is controlled as follows:

The output latch of M0 (also used as P0) rotates at the rising edge of the TFF3 trigger pulse (that inverts the value of TFF3) and is output to the port.

The rotating direction is selected by SMCR0 (CCW0). Setting SMCR0 to "0" selects the normal rotation (M00 → M01 → M02 → M03), and setting it to "1" results in the reverse rotation (M00 ← M01 ← M02 ← M03). The 4-phase 1-step excitation can be obtained by initializing any one bit of Port 0 to "1", and setting two succeeding bits to "1" produces the 4-phase 2-step excitation.

Figure 6.75 is a block diagram of the two excitations.

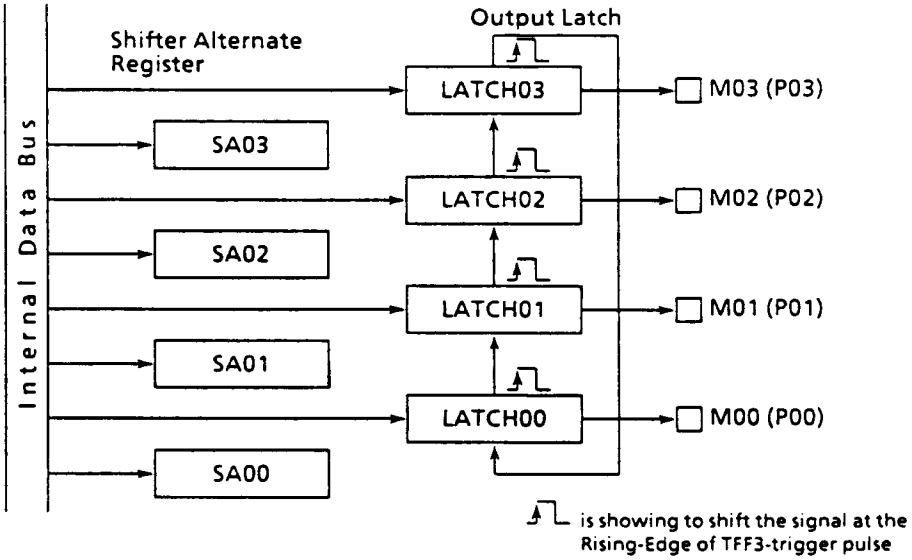
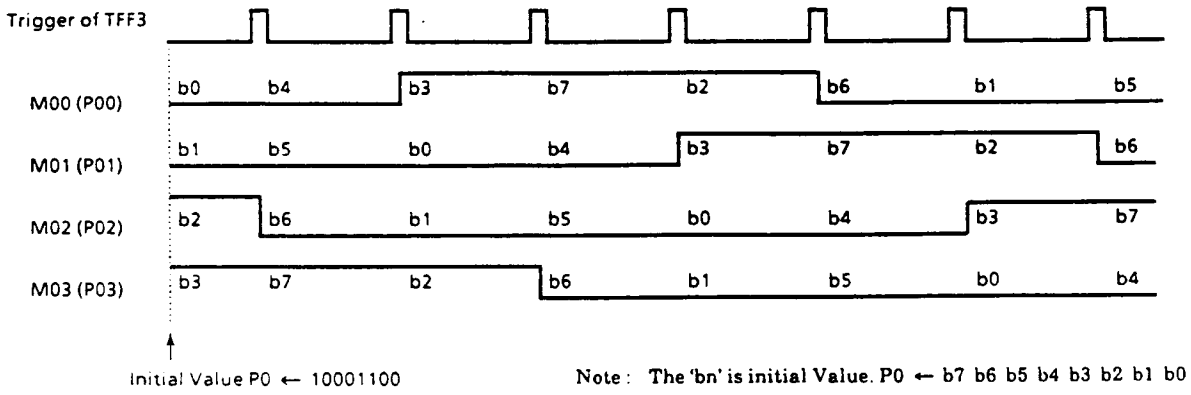


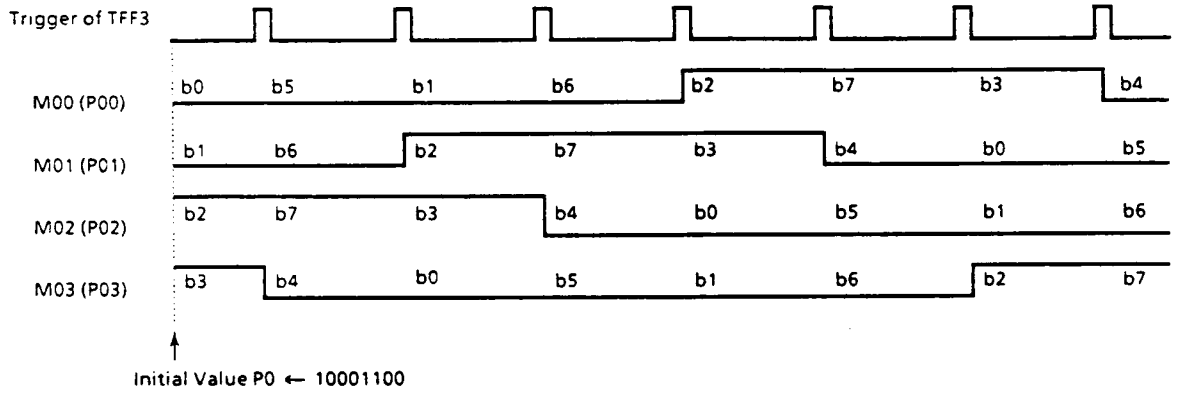
Figure 6.75 Block Diagram of 4-phase 1-step/2-step Excitation (Normal Rotation)

6.9.4 4-phase 1-2 step Excitation

Figure 6.76 shows the output waveforms of 4-phase 1-2 step excitation when Channel 0 is selected.



① Normal Rotation



② Reverse Rotation

Figure 6.76 Output Waveforms of 4-phase 1-2 step Excitation (Normal Rotation and Reverse Rotation)

The 4-phase 1-2 step excitation is obtained as follows:

Given the Port bits | b7 b6 b5 b4 b3 b2 b1 b0 | initialially arranged as, | b3 b7 b2 b6 b1 b5 b0 b4 | set three succeeding bits to "1", and the other bits to "0" (Positive logic).

For example, initializing b3, b7 and b2 to "1" provided the initial value "10001100", which produces the waveforms in Figure 6.76.

To obtain the negative logic output, the above initial value of Port 6 should be inverted.

That is, the waveforms in Figure 6.76 can be converted to the negative logic output by initializing the P0 bits to "01110011".

When Channel 0 is selected, for example, the stepping motor control port is controlled as follows:

The output latch of M0 (also used as P0) and the stepping motor control shifter alternate register (SA0) rotate at the rising edge of the TFF3 trigger pulse and are output to the port. The rotating direction is selected by SMCR0 (CCW0).

Figure 6.77 is a block diagram of this excitation.

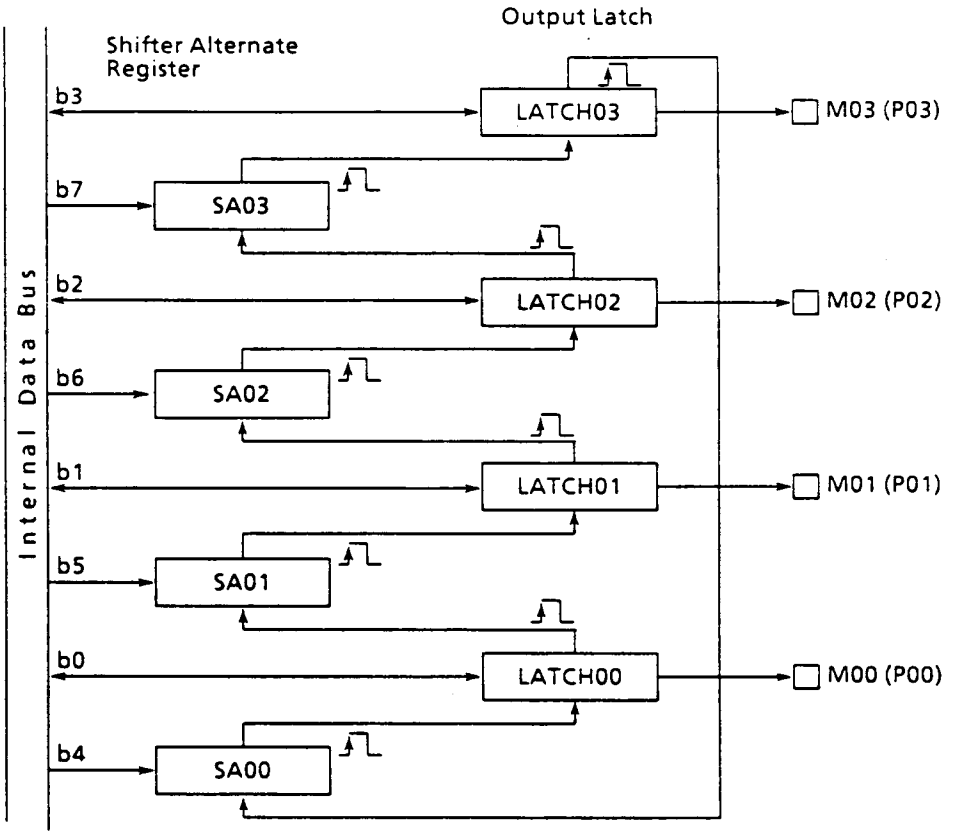


Figure 6.77 Block Diagram of 4-phase 1-2 step Excitation (Normal Rotation)

7. INSTRUCTION SET AND EXECUTION TIMES

7.1 INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the TMP68303.

7.1.1 Addressing Categories

Effective address modes may be categorized by the ways in which they may be used. The following classifications will be used in the instruction definitions.

Data : If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.

Memory : If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.

Alterable : If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.

Control : If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

These categories may be combined, so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

Table 7.1 shows the various categories to which each of the effective address modes belong. Table 7.2 is the instruction set summary.

Table 7.1 Effective Addressing Mode Categories

Effective Address Modes	Mode	Register	Addressing Categories			
			Data	Memory	Control	Alterable
Dn	000	Register Number	×	—	—	×
An	001	Register Number	—	—	—	×
(An)	010	Register Number	×	×	×	×
(An) +	011	Register Number	×	×	—	×
-(An)	100	Register Number	×	×	—	×
d16(An)	101	Register Number	×	×	×	×
d8(An, Xn)	110	Register Number	×	×	×	×
Abs.W	111	000	×	×	×	×
Abs.L	111	001	×	×	×	×
d16(PC)	111	010	×	×	×	—
d8(PC, Xn)	111	011	×	×	×	—
#xxx	111	100	×	×	—	—

Table 7.2 Instruction Set (Sheet 1 of 5)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ABCD	Add Decimal with Extend	(Destination)$_{10}$ + (Source)$_{10}$ + × →Destination	*	U	*	U	*
ADD	Add Binary	(Destination) + (Source) →Destination	*	*	*	*	*
ADDA	Add address	(Destination) + (Source) →Destination	—	—	—	—	—
ADDI	Add Immediate	(Destination) + Immediate Data →Destination	*	*	*	*	*
ADDQ	Add Quick	(Destination) + Immediate Data →Destination	*	*	*	*	*
ADDX	Add Extended	(Destination) + (Source) + × →Destination	*	*	*	*	*
AND	AND Logical	(Destination)∧(Source) →Destination	—	*	*	0	0
ANDI	AND Immediate	(Destination)∧Immediate Data →Destination	—	*	*	0	0
ANDI to CCR	AND Immediate to Condition Codes	(Source)∧CCR→CCR	*	*	*	*	*
ANDI to SR	AND Immediate to Status Register	(Source)∧SR→SR	*	*	*	*	*
ASL, ASR	Arithmetic Shift	(Destination)Shifted by< count> →Destination	*	*	*	*	*
Bcc	Branch Conditionally	If cc then PC + d→PC	—	—	—	—	—
BCHG	Test a Bit and Change	~(<bit number>)OF Destination→Z ~(<bit number>)OF Destination→ <bit number> OF Destination	—	—	*	—	—
BCLR	Test a Bit and Clear	~(<bit number>)OF Destination→Z 0→<bit number> OF Destination	—	—	*	—	—
BRA	Branch Always	PC + d→PC	—	—	—	—	—
BSET	Test a Bit and Set	~(<bit number>)OF Destination→Z 1→<bit number> OF Destination	—	—	*	—	—

Table 7.2 Instruction Set (Sheet 2 of 5)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
BSR	Branch to Subroutine	PC→ – (SP); PC + d→PC	–	–	–	–	–
BTST	Test a Bit	~(<bit number>)OF Destination→Z	–	–	*	–	–
CHK	Check Register Against Bounds	If Dn <0 or DN> (<ea>) then TRAP	–	*	U	U	U
CLR	Clear an Operand	0→Destination	–	0	1	0	0
CMP	Compare	(Destination) – (Source)	–	*	*	*	*
CMPA	Compare Address	(Destination) – (Source)	–	*	*	*	*
CMPI	Compare Immediate	(Destination) – Immediate Data	–	*	*	*	*
CMPM	Compare Memory	(Destination) – (Source)	–	*	*	*	*
DBcc	Test Condition, Decrement and Branch	If ~ cc then Dn – 1→Dn ; if Dn ≠ – 1 then PC + d→PC	–	–	–	–	–
DIVS	Signed Divide	(Destination) / (Source) →Destination	–	*	*	*	0
DIVU	Unsigned Divide	(Destination) / (Source) →Destination	–	*	*	*	0
EOR	Exclusive OR Logical	(Destination)⊕(Source) →Destination	–	*	*	0	0
EORI	Exclusive OR Immediate	(Destination)⊕Immediate Data →Destination	–	*	*	0	0
EORI to CCR	Exclusive OR Immediate to Condition Codes	(Source)⊕CCR→CCR	*	*	*	*	*
EORI to SR	Exclusive OR Immediate to Status Register	(Source)⊕SR→SR	*	*	*	*	*
EXG	Exchange Register	Xx↔Xy	–	–	–	–	–
EXT	Sign Extend	(Destination) Sign-Extended →Destination	–	*	*	0	0
JMP	Jump	Destination→PC	–	–	–	–	–
JSR	Jump to Subroutine	PC→ – (SP); Destination→PC	–	–	–	–	–
LEA	Load Effective Address	<ea>→An	–	–	–	–	–
LINK	Link and Allocate	An→ – (SP); SP→An; SP + Displacement→SP	–	–	–	–	–

Table 7.2 Instruction Set (Sheet 3 of 5)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
LSL, LSR	Logical Shift	(Destination) Shifted by <count> →Destination	*	*	*	0	*
MOVE	Move Data from Source to Destination	(Source)→Destination	—	*	*	0	0
MOVE to CCR	Move to Condition Code	(Source)→CCR	*	*	*	*	*
MOVE to SR	Move to the Status Register	(Source)→SR	*	*	*	*	*
MOVE from SR	Move from the Status Register	SR→Destination	—	—	—	—	—
MOVE USP	Move User Stack Pointer	USP→An; An→USP	—	—	—	—	—
MOVEA	Move Address	(Source)→Destination	—	—	—	—	—
MOVEM	Move Multiple Registers	Registers→Destination (Source)→Registers	—	—	—	—	—
MOVEP	Move Peripheral Data	(Source)→Destination	—	—	—	—	—
MOVEQ	Move Quick	Immediate Data→Destination	—	*	*	0	0
MULS	Signed Multiply	(Destination) × (Source) →Destination	—	*	*	0	0
MULU	Unsigned Multiply	(Destination) × (Source) →Destination	—	*	*	0	0
NBCD	Negate Decimal with Extend	$0 - (\text{Destination})_{10} - X$ →Destination	*	U	*	U	*
NEG	Negate	0 – (Destination) →Destination	*	*	*	*	*
NEGX	Negate with Extend	0 – (Destination) – X →Destination	*	*	*	*	*
NOP	No Operation	—	—	—	—	—	—
NOT	Logical Complement	~(Destination)→Destination	—	*	*	0	0
OR	Inclusive OR Logical	(Destination)∨(Source) →Destination	—	*	*	0	0
ORI	Inclusive OR Immediate	(Destination)∨Immediate Data →Destination	—	*	*	0	0
ORI to CCR	Inclusive OR Immediate to Condition Codes	(Source) ∨ CCR→CCR	*	*	*	*	*

Table 7.2 Instruction Set (Sheet 4 of 5)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ORI to SR	Inclusive OR Immediate to Status Register	(Source) ∨ SR→SR	*	*	*	*	*
PEA	Push Effective Address	<ea>→ – (SP)	—	—	—	—	—
RESET	Reset External Device	—	—	—	—	—	—
ROL, ROR	Rotate (Without Extend)	(Destination) Rotated by <count> →Destination	—	*	*	0	*
ROXL, ROXR	Rotate with Extend	(Destination) Rotated by <count> →Destination	*	*	*	0	*
RTE	Return from Exception	(SP) + →SR; (SP) + →PC	*	*	*	*	*
RTR	Return and Restore Condition Codes	(SP) + →CC; (SP) + →PC	*	*	*	*	*
RTS	Return from Subroutine	(SP) + →PC	—	—	—	—	—
SBCD	Subtract Decimal with Extend	$(Destination)_{10} - (Source)_{10} - X$ →Destination	*	U	*	U	*
Scc	Set According to Condition	If cc then 1's →Destination else 0's→Destination	—	—	—	—	—
STOP	Load Status Register and Stop	Immediate Data→SR;STOP	*	*	*	*	*
SUB	Subtract Binary	(Destination) – (Source) →Destination	*	*	*	*	*
SUBA	Subtract Address	(Destination) – (Source) →Destination	—	—	—	—	—
SUBI	Subtract Immediate	(Destination) – Immediate Data →Destination	*	*	*	*	*
SUBQ	Subtract Quick	(Destination) – Immediate Data →Destination	*	*	*	*	*
SUBX	Subtract with Extend	(Destination) – (source) – X →Destination	*	*	*	*	*
SWAP	Swap Register Halves	Register [31:16]↔Register [15:0]	—	*	*	0	0
TAS	Test and Set an Operand	(Destination) Tested→CC; 1→[7] OF Destination	—	*	*	0	0

Table 7.2 Instruction Set (Sheet 5 of 5)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
TRAP	Trap	PC→ – (SSP); SR→ – (SSP); (Vector)→PC	–	–	–	–	–
TRAPV	Trap on Overflow	If V then TRAP	–	–	–	–	–
TST	Test and Operand	(Destination) Tested→CC	–	*	*	0	0
UNLK	Unlink	An→SP; (SP) + →An	–	–	–	–	–

→ : the left operand is moved to the right operand
↔ : the two operands are exchanged
+ : the operands are added
– : the right operand is subtracted from the left operand
* : the operands are multiplied
/ : the first operand is divided by the second operand
∧ : logical AND
∨ : logical OR
⊕ : logical exclusive OR
< : relational test, true if left operand is less than right operand
> : relational test, true if left operand is greater than right operand
~ : logical complement
[ ] : bit number

* : affected
– : unaffected
0 : cleared
1 : set
U : undefined

7.1.2 Instruction Prefetch

The CP000-T2 uses a two-word tightly-coupled instruction prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved. If the execution of an instruction is defined to begin when the microroutine for that instruction is entered, some features of the prefetch mechanism can be described.

(1) When execution of an instruction begins, the operation word and the word following have already been fetched. The operation word is in the instruction decoder.
(2) In the case of multi-word instructions, as each additional word of the instruction is used internally, a fetch is made to the instruction stream to replace it.
(3) The last fetch for an instruction from the instruction stream is made when the operation word is discarded and decoding is started on the next instruction.
(4) If the instruction is a single-word instruction causing a branch, the second word is not used. But because this word is fetched by the preceding instruction, it is impossible to avoid this superfluous fetch.
(5) In the case of an interrupt or trace exception, both words are not used.
(6) The program counter usually points to the last word fetched from the instruction stream.

7.2 INSTRUCTION EXECUTION TIMES

The following paragraphs contain listings of the instruction execution times in terms of external clock (CLK) periods. In this timing data, it is assumed that both memory raed and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is also included with the timing data. This timing data is enclosed in parenthesis following the execution periods and is shown as (r/w) where r is the number of read cycles and w is the number of write cycles.

Note: The number of periods includes instruction fetch and all applicable operand fetches and stores.

7.2.1 Effective Address Operand Calculation Timing

Table 7.3 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table 7.3 Effective Address Calculation Times

	Addressing Mode	Byte, Word	Long
Dn An	Register Data Register Direct Address Register Direct	 0 (0/0) 0 (0/0)	 0 (0/0) 0 (0/0)
(An) (An) +	Memory Address Register Indirect Address Register Indirect with Postincrement	 4 (1/0) 4 (1/0)	 8 (2/0) 8 (2/0)
– (An) d16(PC)	Address Register Indirect with Predecrement Address Register Indirect with Displacement	6 (1/0) 8 (2/0)	10 (2/0) 12 (3/0)
d8(An, Xn)* Abs.W	Address Register Indirect with Index Absolute Short	10 (2/0) 8 (2/0)	14 (3/0) 12 (3/0)
Abs.L d16(PC)	Absolute Long Program Counter with Displacement	12 (3/0) 8 (2/0)	16 (4/0) 12 (3/0)
d8(PC, Xn)* #xxx	Program Counter with Index Immediate	10 (2/0) 4 (1/0)	14 (3/0) 8 (2/0)

* : The size of the index register (Xn) does not affect execution time.

7.2.2 Move Instruction Execution Times

Table 7.4 and 7.5 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7.4 Move Byte and Word Instruction Excecution Times

Source	Destination								
	Dn	An	(An)	(An) +	– (An)	d16(An)	d8(An,Xn)*	Abs.W	Abs.L
Dn	4 (1/0)	4 (1/0)	8 (1/1)	8 (1/1)	8 (1/1)	12 (2/1)	14 (2/1)	12 (2/1)	16 (3/1)
An	4 (1/0)	4 (1/0)	8 (1/1)	8 (1/1)	8 (1/1)	12 (2/1)	14 (2/1)	12 (2/1)	16 (3/1)
(An)	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)
(An) +	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)
– (An)	10 (2/0)	10 (2/0)	14 (2/1)	14 (2/1)	14 (2/1)	18 (3/1)	20 (3/1)	18 (3/1)	22 (4/1)
d16(An)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
d8(An, Xn)*	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	22 (4/1)	24 (4/1)	22 (4/1)	26 (5/1)
Abs.W	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
Abs.L	16 (4/0)	16 (4/0)	20 (4/1)	20 (4/1)	20 (4/1)	24 (5/1)	26 (5/1)	24 (5/1)	28 (6/1)
d16(PC)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
d8(PC, Xn)*	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	22 (4/1)	24 (4/1)	22 (4/1)	26 (5/1)
#xxx	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)

* : The size of the index register (Xn) does not affect execution time.

Table 7.5 Move Long Instruction Excecution Times

Source	Destination								
	Dn	An	(An)	(An) +	– (An)	d16(An)	d8(An,Xn)*	Abs.W	Abs.L
Dn	4 (1/0)	4 (1/0)	12 (1/2)	12 (1/2)	12 (1/2)	16 (2/2)	18 (2/2)	16 (2/2)	20 (3/2)
An	4 (1/0)	4 (1/0)	12 (1/2)	12 (1/2)	12 (1/2)	16 (2/2)	18 (2/2)	16 (2/2)	20 (3/2)
(An)	12 (3/0)	12 (3/0)	20 (3/2)	20 (3/2)	20 (3/2)	24 (4/2)	26 (4/2)	24 (4/2)	28 (5/2)
(An) +	12 (3/0)	12 (3/0)	20 (3/2)	20 (3/2)	20 (3/2)	24 (4/2)	26 (4/2)	24 (4/2)	28 (5/2)
– (An)	14 (3/0)	14 (3/0)	22 (3/2)	22 (3/2)	22 (3/2)	26 (4/2)	28 (4/2)	26 (4/2)	30 (5/2)
d16(An)	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	28 (5/2)	30 (5/2)	28 (5/2)	32 (6/2)
d8(An, Xn)*	18 (4/0)	18 (4/0)	26 (4/2)	26 (4/2)	26 (4/2)	30 (5/2)	32 (5/2)	30 (5/2)	34 (6/2)
Abs.W	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	28 (5/2)	30 (5/2)	28 (5/2)	32 (6/2)
Abs.L	20 (5/0)	20 (5/0)	28 (5/2)	28 (5/2)	28 (5/2)	32 (6/2)	34 (6/2)	32 (6/2)	36 (7/2)
d16(PC)	16 (4/0)	16 (4/0)	24 (4/1)	24 (4/2)	24 (4/2)	28 (5/2)	30 (5/2)	28 (5/2)	32 (5/2)
d8(PC, Xn)*	18 (4/0)	18 (4/0)	26 (4/2)	26 (4/2)	26 (4/2)	30 (5/2)	32 (5/2)	30 (5/2)	34 (6/2)
#xxx	12 (3/0)	12 (3/0)	20 (3/2)	20 (3/2)	20 (3/2)	24 (4/2)	26 (4/2)	24 (4/2)	28 (5/2)

* : The size of the index register (Xn) does not affect execution time.

7.2.3 Standard Instruction Execution Times

The number of clock periods shown in Table 7.6 indicates the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 7.6 the headings have the following maenings:

An = address register operand
Dn = data register operand
ea = an operand specified by an effective address
M = memory effective address operand

Table 7.6 Standard Instruction Execution Times

Instruction	Size	op<ea>, An^	op<ea>, Dn	opDn, <M>
ADD	Byte, Word	8 (1/0) +	4 (1/0) +	8 (1/1) +
	Long word	6 (1/0) + **	6 (1/0) + **	12 (1/2) +
AND	Byte, Word	—	4 (1/0) +	8 (1/1) +
	Long word	—	6 (1/0) + **	12 (1/2) +
CMP	Byte, Word	6 (1/0) +	4 (1/0) +	—
	Long word	6 (1/0) +	6 (1/0) +	—
DIVS	—	—	158 (1/0) + *	—
DIVU	—	—	140 (1/0) + *	—
EOR	Byte, Word	—	4 (1/0) ***	8 (1/1) +
	Long word	—	8 (1/0) ***	12 (1/2) +
MULS	—	—	70 (1/0) + *	—
MULU	—	—	70 (1/0) + *	—
OR	Byte, Word	—	4 (1/0) +	8 (1/1) +
	Long word	—	6 (1/0) + **	12 (1/2) +
SUB	Byte, Word	8(1/0) +	4 (1/0) +	8 (1/1) +
	Long word	6(1/0) + **	6 (1/0) + **	12 (1/2) +

\+ : add effective address calculation time
^ : word or long word only
* : indicates maximum value
** : The base time of six clock periods is increased to eight if the effective address mode is register direct or immediate (effective address time should also be added).
*** : Only available effective address mode is data register direct.
DIVS, DIVU – The divide algorithm used by the TMP68000 provides less than 10% difference between the best and worst case timings.
MULS, MULU – The multiply algorithm requires 38+2n clocks where n is defined as:

MULU : n = the number of ones in the <ea>

MULS : n = concatanate the <ea> with a zero as the LSB; n is The resultant number of 10 or 01 pattern in the 17-bit source: i.e., worst case happens when the source is $5555.

7.2.4 Immediate Instruction Execution Times

The number of clock periods shown in Table 7.7 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 7.7, the headings have the following meanings:

- # = immediate operand
- Dn = data register operand
- An = address register operand
- M = memory operand
- SR = status register

Table 7.7 Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte Word	8 (2/0)	—	12 (2/1) +
	Long word	16 (3/0)	—	20 (3/2) +
ADDQ	Byte Word	4 (1/0)	8 (1/0) *	8 (1/1) +
	Long word	8 (1/0)	8 (1/0)	12 (1/2) +
ANDI	Byte Word	8 (2/0)	—	12 (2/1) +
	Long word	16 (3/0)	—	20 (3/1) +
CMPI	Byte Word	8 (2/0)	—	8 (2/0) +
	Long word	14 (3/0)	—	12 (3/0) +
EORI	Byte Word	8 (2/0)	—	12 (2/1) +
	Long word	16 (3/0)	—	20 (3/2) +
MOVEQ	Long word	4 (1/0)	—	—
ORI	Byte Word	8 (2/0)	—	12 (2/1) +
	Long word	16 (3/0)	—	20 (3/2) +
SUBI	Byte Word	8 (2/0)	—	12 (2/1) +
	Long word	16 (3/0)	—	20 (3/2) +
SUBQ	Byte Word	4 (1/0)	8 (1/0) *	8 (1/1) +
	Long word	8 (1/0)	8 (1/0)	12 (1/2) +

+ : add effective address calculation time

* : word only

7.2.5 Single Operand Instruction Execution Times

Table 7.8 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.8 Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte, Word	4 (1/0)	8 (1/1) +
	Long word	6 (1/0)	12 (1/2) +
NBCD	Byte	6 (1/0)	8 (1/1) +
NEG	Byte, Word	4 (1/0)	8 (1/1) +
	Long word	6 (1/0)	12 (1/2) +
NEGX	Byte, Word	4 (1/0)	8 (1/1) +
	Long word	6 (1/0)	12 (1/2) +
NOT	Byte, Word	4 (1/0)	8 (1/1) +
	Long word	6 (1/0)	12 (1/2) +
Scc	Byte, False	4 (1/0)	8 (1/1) +
	Byte, True	6 (1/0)	8 (1/1) +
TAS	Byte	4 (1/0)	10 (1/1) +
TST	Byte, Word	4 (1/0)	4 (1/0) +
	Long word	4 (1/0)	4 (1/0) +

+ : add effective address calculation time

7.2.6 Shift/Rotate Instruction Execution Times

Table 7.9 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.9 Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n (1/0)	8 (1/1) +
	Long word	8 + 2n (1/0)	–
LSR, LSL	Byte, Word	6 + 2n (1/0)	8 (1/1) +
	Long word	8 + 2n (1/0)	–
ROR, ROL	Byte, Word	6 + 2n (1/0)	8 (1/1) +
	Long word	8 + 2n (1/0)	–
ROXR, ROXL	Byte, Word	6 + 2n (1/0)	8 (1/1) +
	Long word	8 + 2n (1/0)	–

+ : add effective address calculation time

n : the shift or rotate count

7.2.7 Bit Manipulation Instruction Execution Times

Table 7.10 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.10 Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	—	8 (1/1) +	—	12 (2/1) +
	Long word	8 (1/0) *	—	12 (2/0) *	—
BCLR	Byte	—	8 (1/1) +	—	12 (2/1) +
	Long word	10 (1/0) *	—	14 (2/0) *	—
BSET	Byte	—	8 (1/1) +	—	12 (2/1) +
	Long word	8 (1/0) *	—	12 (2/0) *	—
BTST	Byte	—	4 (1/0) +	—	8 (2/0) +
	Long word	6 (1/0)	—	10 (2/0)	—

+ : add effective address calculation time

* : indicates maximum value

7.2.8 Conditional Instruction Execution Times

Table 7.11 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.11 Conditional Instruction Execution Times

Instruction	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte	10 (2/0)	8 (1/0)
	Word	10 (2/0)	12 (2/0)
BRA	Byte	10 (2/0)	—
	Word	10 (2/0)	—
BSR	Byte	18 (2/2)	—
	Word	18 (2/2)	—
DBcc	CC true	—	12 (2/0)
	CC false	10 (2/0)	14 (3/0)

7.2.9 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Table 7.12 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7.12 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instr	Size	(An)	(An) +	– (An)	d16 (An)	d8 (An, Xn) *	Abs.W	Abs.L	d16 (PC)	d8 (PC, Xn) *
JMP	—	8 (2/0)	—	—	10 (2/0)	14 (3/0)	10 (2/0)	12 (3/0)	10 (2/0)	14 (3/0)
JSR	—	16 (2/2)	—	—	18 (2/2)	22 (2/2)	18 (2/2)	20 (3/2)	18 (2/2)	22 (2/2)
LEA	—	4 (1/0)	—	—	8 (2/0)	12 (2/0)	8 (2/2)	12 (3/0)	8 (2/0)	12 (2/0)
PEA	—	12 (1/2)	—	—	16 (2/2)	20 (2/2)	16 (2/2)	20 (3/2)	16 (2/2)	20 (2/2)
MOVEM M→R	Word	12 + 4n (3 + n/0)	12 + 4n (3 + n/0)	—	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)	16 + 4n (4 + 2n/0)	20 + 4n (5 + n/0)	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)
	Long word	12 + 8n (3 + 2n/0)	12 + 8n (3 + 2n/0)	—	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)	16 + 8n (4 + 2n/0)	20 + 8n (5 + 2n/0)	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)
MOVEM R→M	Word	8 + 4n (2/n)	—	8 + 4n (2/n)	12 + 4n (3/n)	14 + 4n (3/n)	12 + 4n (3/n)	16 + 4n (4/n)	—	—
	Long word	8 + 8n (2/2n)	—	8 + 8n (2/2n)	12 + 8n (3/2n)	14 + 8n (3/2n)	12 + 8n (3/2n)	16 + 8n (4/2n)	—	—

n : the number of registers to move

* : the size of the index register (Xn) does not affect the instruction's execution time

7.2.10 Multi-Precision Instruction Execution Times

Table 7.13 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as (r/w).

In Table 7.13, the headings have the following meaning:

Dn = data register operand

M = memory operand.

Table 7.13 Multi-Precision Instruction Execution Times

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4 (1/0)	18 (3/1)
	Long word	8 (1/0)	30 (5/2)
CMPM	Byte, Word	—	12 (3/0)
	Long word	—	20 (5/0)
SUBX	Byte, Word	4 (1/0)	18 (3/1)
	Long word	8 (1/0)	30 (5/2)
ABCD	Byte	6 (1/0)	18 (3/1)
SBCD	Byte	6 (1/0)	18 (3/1)

7.2.11 Miscellaneous Instruction Execution Times

Table 7.14 and 7.15 indicate the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table 7.14 Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory
ANDI to CCR	Byte	20 (3/0)	–
ANDI to SR	Word	20 (3/0)	–
CHK	–	10 (1/0) +	–
EORI to CCR	Byte	20 (3/0)	–
EORI to SR	Word	20 (3/0)	–
ORI to CCR	Byte	20 (3/0)	–
ORI to SR	Word	20 (3/0)	–
MOVE from SR	–	6 (1/0)	8 (1/1) +
MOVE to CCR	–	12 (1/0)	12 (1/0) +
MOVE to SR	–	12 (1/0)	12 (1/0) +
EXG	–	6 (1/0)	–
EXT	Word	4 (1/0)	–
	Long word	4 (1/0)	–
LINK	–	16 (2/2)	–
MOVE from USP	–	4 (1/0)	–
MOVE to USP	–	4 (1/0)	–
NOP	–	4 (1/0)	–
RESET	–	132 (1/0)	–
RTE	–	20 (5/0)	–
RTR	–	20 (5/0)	–
RTS	–	16 (4/0)	–
STOP	–	4 (0/0)	–
SWAP	–	4 (1/0)	–
TRAPV (No Trap)	–	4 (1/0)	–
UNLK	–	12 (3/0)	–

+ : add effective address calculation time

Table 7.15 Move Peripheral Instruction Execution Times

Instruction	Size	Register→Memory	Memory→Register
MOVEP	Word	16 (2/2)	16 (4/0)
	Long word	24 (2/4)	24 (6/0)

7.2.12 Exception Processing Execution Times

Table 7.16 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7.16 Exception Processing Execution Times

Exception	Periods
Address Error	50 (4/7)
Bus Error	50 (4/7)
CHK Instruction (Trap Taken)	44 (5/3) +
Divide by Zero	42 (5/3)
Illegal Instruction	34 (4/3)
Interrupt	44 (5/3)*
Privilege Violation	34 (4/3)
$\overline{\text{RESET}}$**	40 (6/0)
Trace	34 (4/3)
TRAP Instruction	38 (4/3)
TRAPV Instruction (Trap Taken)	34 (4/3)

+ : add effective address calculation time

* : The interrupt acknowledge cycle is assumed to take four clock periods.

** : Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts.

8. ELECTRICAL SPECIFICATIONS

This section contains electrical specifications and associated timing information for the CP000-T2.

8.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	Vcc	-0.3~ +7.0	V
Input Voltage	Vin	-0.3~ +7.0	V
Operating Temperature Range	Ta	0~ +70	°C
Storage Temperature	Tstg	-55~ +150	°C

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage 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 Vcc).

8.2 DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C)

Item		Symbol	TMP68303 Min	TMP68303 Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND-0.3	0.8	V
Input Leakage Curren (5.25V)	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{HALT}$, $\overline{RESET}$, CLK, NOR / $\overline{EMU}$, TIN, INT0~INT2, RxD0, RxD1, $\overline{DREQ0}$~$\overline{DREQ2}$, $\overline{DTEND}$	I_{IN}	– – –	2.5 2.5 20	μA μA
3-state (off state) Input Current	(2.4V/0.4V) $\overline{AS}$, A1~A23, D0~D15,FC0~FC2, $\overline{LDS}$, $\overline{UDS}$, R / $\overline{W}$	I_{TSI}	– –	20 20	μA μA
Output High Voltage (IOH = -400μA)	$\overline{AS}$, A1~A23, D0~D15, FC0~FC2,$\overline{LDS}$, $\overline{UDS}$, R / $\overline{W}$, $\overline{BG}$, P00~P03, P10~P13, P20, P21 Tout1, Tout2, $\overline{RAS}$, $\overline{CAS}$, INT0~INT2, TxD0, TxD1, $\overline{DACK0}$~$\overline{DACK2}$, IACK0~IACK2	V_{OH}	– V_{CC}-0.75	–	V
Output Low Voltage (IOL = 1.6mA) (IOL = 3.2mA) (IOL = 5.0mA) (IOL = 5.3mA)	$\overline{HALT}$ A1~A23, $\overline{BG}$, FC0~FC2 $\overline{RESET}$ $\overline{AS}$, D0~D15, $\overline{LDS}$, $\overline{UDS}$, R / $\overline{W}$, P00~P03, P10~P13, P20, P21, Tout1, Tout2, $\overline{RAS}$, $\overline{CAS}$, INT0~INT2, TxD0, TxD1, $\overline{DACK0}$~$\overline{DACK2}$, IACK0~IACK2	V_{OL}	– – – –	0.5 0.5 0.5 0.5	V
Current Dissipation	f = 12.5MHz f = 16.67MHz**	I_D	– –	90 100	mA
Power Dissipation	f = 12.5MHz f = 16.67MHz**	P_D	– –	0.473 0.525	W
Capacitance (Vin = 0V, Ta = 25°C : Frequency = 1MHz)*		C_{IN}	–	20.0	pF
Load Capacitance	$\overline{HALT}$ All other	C_L	– –	70 130	pF

281189

* : Capacitance is periodically sampled rather then 100% test

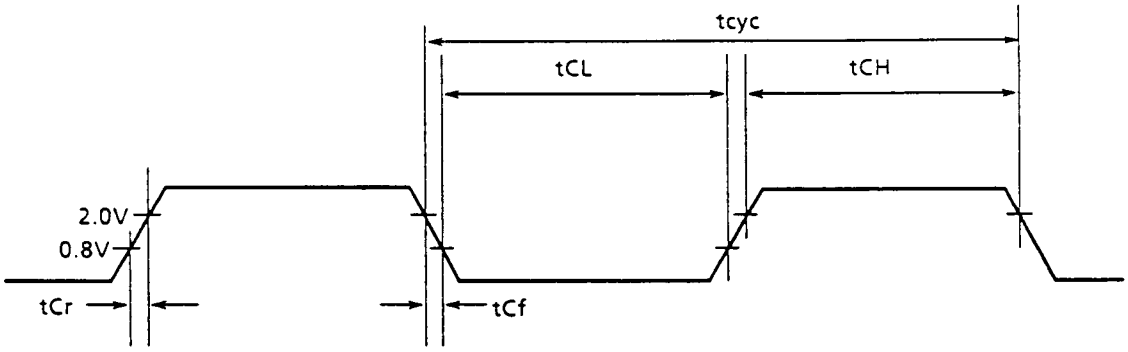
** : Provisinal

8.3 AC Electrical Specifications – Clock Timing

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.1)

Characteristic	Symbol	8MHz		10MHz		12.5MHz		16.67MHz*		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
Frequency of Operation	f	4.0	8.0	4.0	10.0	4.0	12.5	8.0	16.67	MHz
Cycle Time	tcyc	125	250	100	250	80	250	60	125	ns
Clock Pulse Width	tCL tCH	55 55	125 125	45 45	125 125	35 35	125 125	27 27	62.5 62.5	ns
Rise and Fall Times	tCr tCf	– –	10 10	– –	10 10	– –	5 5	– –	5 5	ns

* : Provision



Note : Timing measurements are referenced to and from a low voltage of 0.8 volt and high a voltage of 2.0 volts, 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 volt and 2.0 volts.

Figure 8.1 Clock Input Timing Diagram

8.4 AC Electrical Specifications – Read and Write Cycles (1/4)

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figures 8.2 and 8.3)

Num.	Characteristic	Symbol	8MHz		10MHz		12.5MHz		16.67MHz*		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
1	Clock Period	tCYC	125	250	100	250	80	250	60	125	ns
2	Clock Width Low	tCL	55	125	45	125	35	125	27	62.5	ns
3	Clock Width High	tCH	55	125	45	125	35	125	27	62.5	ns
4	Clock Fall Time	tCf	–	10	–	10	–	5	–	5	ns
5	Clock Rise Time	tCr	–	10	–	10	–	5	–	5	ns
6	Clock Low to Address Valid	tCLAV	–	62	–	50	–	50	–	30	ns
6A	Clock High to FC Valid	tCHFCV	–	62	–	50	–	45	0	30	ns
7	Clock High to Address, Data Bus High Impedance (Maximum)	tCHADZ	–	80	–	70	–	60	–	50	ns
8	Clock High to Address, FC Invalid (Minimum)	tCHAFI	0	–	0	–	0	–	0	–	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low	tCHSL	3	60	3	50	3	40	3	30	ns
11[2]	Address Valid to $\overline{AS}$, $\overline{DS}$ Low (Read) / $\overline{AS}$ Low (Write)	tAVSL	30	–	20	–	15	–	15	–	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ Low (Read) / $\overline{AS}$ Low (Write)	tFCVSL	90	–	70	–	60	–	45	–	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	tCLSH	–	62	–	50	–	40	3	30	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address / FC Invalid	tSHAFI	40	–	30	–	20	–	15	–	ns
14[2]	$\overline{AS}$, $\overline{DS}$ Width Low (Read) / $\overline{AS}$ Low (Write)	tSL	270	–	195	–	160	–	120	–	ns
14A[2]	$\overline{DS}$ Width Low (Write)	tDSL	140	–	95	–	80	–	60	–	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	tSH	150	–	105	–	65	–	60	–	ns

*: Provision

8.4 AC Electrical Specifications – Read and Write Cycles (2/4)

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figures 8.2 and 8.3)

Num.	Characteristic	Symbol	8MHz		10MHz		12.5MHz		16.67MHz*		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
16	Clock High to Control Bus High Impedance	tCHCZ	–	80	–	70	–	60	–	50	ns
17[2]	$\overline{AS}$, $\overline{DS}$, High to R / $\overline{W}$ High (Read)	tSHRH	40	–	30	–	20	–	15	–	ns
18[1]	Clock High to R / $\overline{W}$ High	tCHRH	0	55	0	45	0	40	0	30	ns
20[1]	Clock High to R / $\overline{W}$ Low (Write)	tCHRL	0	55	0	45	0	40	0	30	ns
20A[2,8]	$\overline{AS}$ Low to R / $\overline{W}$ Valid (Write)	tASRV	–	10	–	10	–	10	–	10	ns
21[2]	Address Valid to R / $\overline{W}$ Low (Write)	tAVRL	20	–	0	–	0	–	0	–	ns
21A[2]	FC Valid to R / $\overline{W}$ Low (Write)	tFCVRL	60	–	50	–	30	–	30	–	ns
22[2]	R / $\overline{W}$ Low to $\overline{DS}$ Low (Write)	tRLSL	80	–	50	–	30	–	30	–	ns
23	CLock Low to Data Out Valid (Write)	tCLDO	–	62	–	50	–	50	–	30	ns
25[2]	$\overline{AS}$, $\overline{DS}$ High to Data Out Invalid (Write)	tSHDOI	40	–	30	–	20	–	15	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	tDOSL	40	–	30	–	20	–	15	–	ns
27[6]	Data in to Clock Low (Setup Time on Read)	tDICL	10	–	10	–	10	–	5	–	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to Data in Invalid (Hot Time on Read)	tSHDAH	0	240	0	190	0	150	0	110	ns
29	($\overline{AS}$, $\overline{DS}$ High to BERR High	tSHDII	0	–	0	–	0	–	0	–	ns

*: Provision

8.4 AC Electrical Specifications – Read and Write Cycles (3/4)

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figures 8.2 and 8.3)

Num.	Characteristic	Symbol	8MHz		10MHz		12.5MHz		16.67MHz*		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	tSHBEH	0	–	0	–	0	–	0	–	ns
31[2,6]	$\overline{DTACK}$ Low to Data In (Setup Time)	tDALDI	–	90	–	65	–	50	–	50	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition	tRHr, f	0	200	0	200	0	200	–	150	ns
33	Clock High to $\overline{BG}$ Low	tCHGL	–	62	–	50	–	40	0	30	ns
34	CLock High to $\overline{BG}$ High	tCHGH	–	62	–	50	–	40	0	30	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	tBRLGL	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
36[9]	$\overline{BR}$ High to $\overline{BG}$ High	tBRHGH	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	tGALGH	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
37A[10]	$\overline{BGACK}$ Low to $\overline{BR}$ High	tGALBRH	20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	10	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	tGLZ	–	80	–	70	–	60	–	50	ns
39	$\overline{BG}$ Width High	tGH	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.

*: Provision

8.4 AC Electrical Specifications – Read and Write Cycles (4/4)

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figures 8.2 and 8.3)

Num.	Characteristic	Symbol	8MHz		10MHz		12.5MHz		16.67MHz*		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
46	$\overline{BGACK}$ Width Low	tGAL	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
47[6]	Asynchronous Input Setup Time	tASI	10	–	10	–	10	–	5	–	ns
48[2,3]	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	tBELDAL	20	–	20	–	20	–	10	–	ns
53	Clock High to Data Out Invalid	tCHDOI	0	–	0	–	0	–	0	–	ns
55	R / $\overline{W}$ to Data Bus Driven	tRLDBD	30	–	20	–	10	–	0	–	ns
56[4]	$\overline{HALT}$ / $\overline{RESET}$ Pulse Width	tHRPW	12	–	12	–	12	–	12	–	Clk. Per.
57	$\overline{BGACK}$ High to Control Bus Driven	tGASD	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
58[9]	$\overline{BG}$ High to Control Bus Driven	tRHSD	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.

*: Provision

Note : 1. For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the value given in the maximum columns.

2. Actual value depends on clock period.

3. If #47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, #48 may be ignored. In the absence of $\overline{DTACK}$, $\overline{BERR}$ is an asynchronous input using the asynchronous input setup time (#47).

4. For power-up, the MPU must be held in the reset state for 100 milliseconds to allow stabilization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the processor.

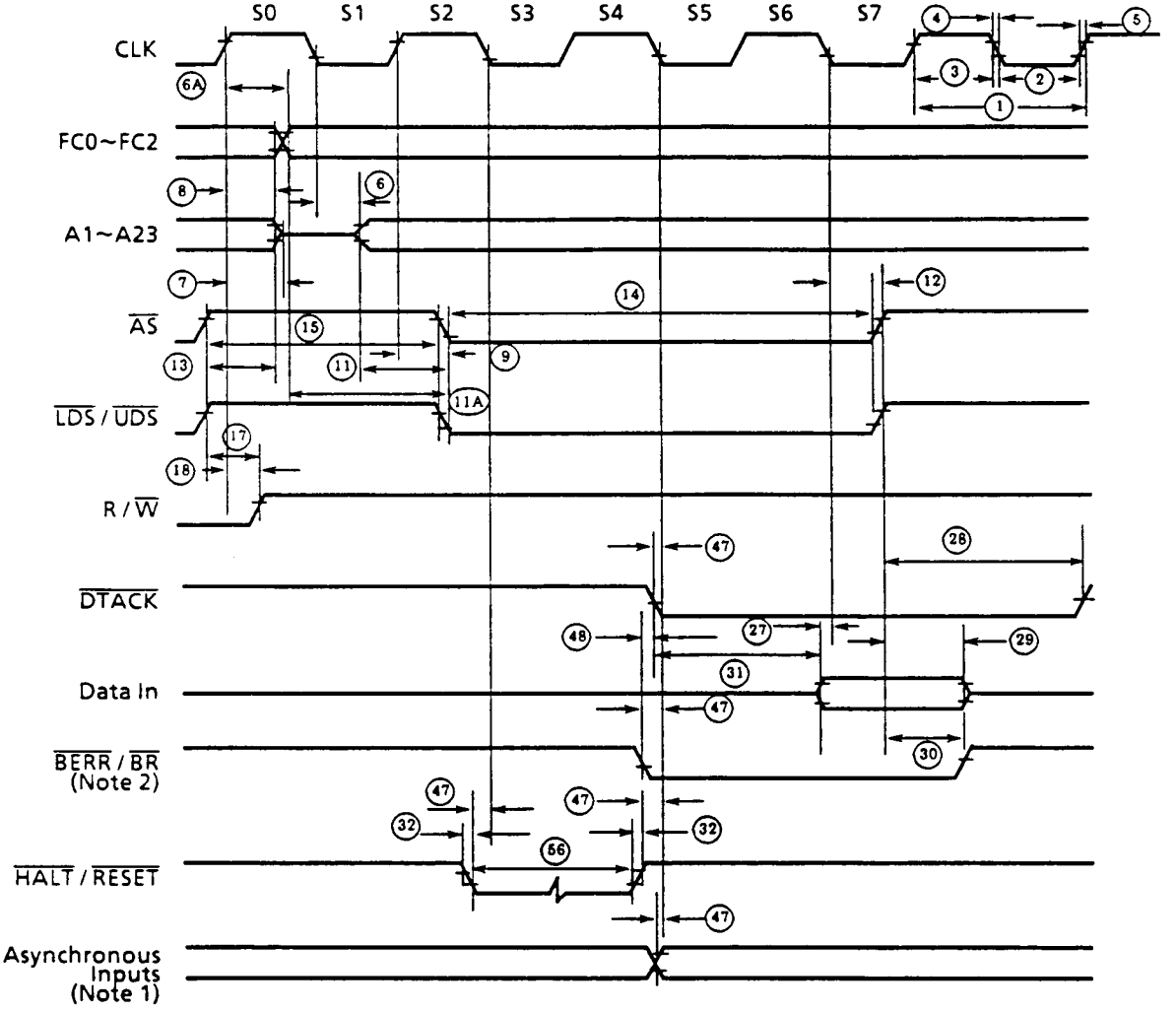
6. If the asynchronous input setup time (#47) requirement is satisfied for $\overline{DTACK}$, the $\overline{DTACK}$-asserted to data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle.

8. When $\overline{AS}$ and R/$\overline{W}$ are equally loaded (±20%), subtract 5 nanoseconds from the values given is these columns.

9. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.

10. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be re-asserted.

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional descriptions of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



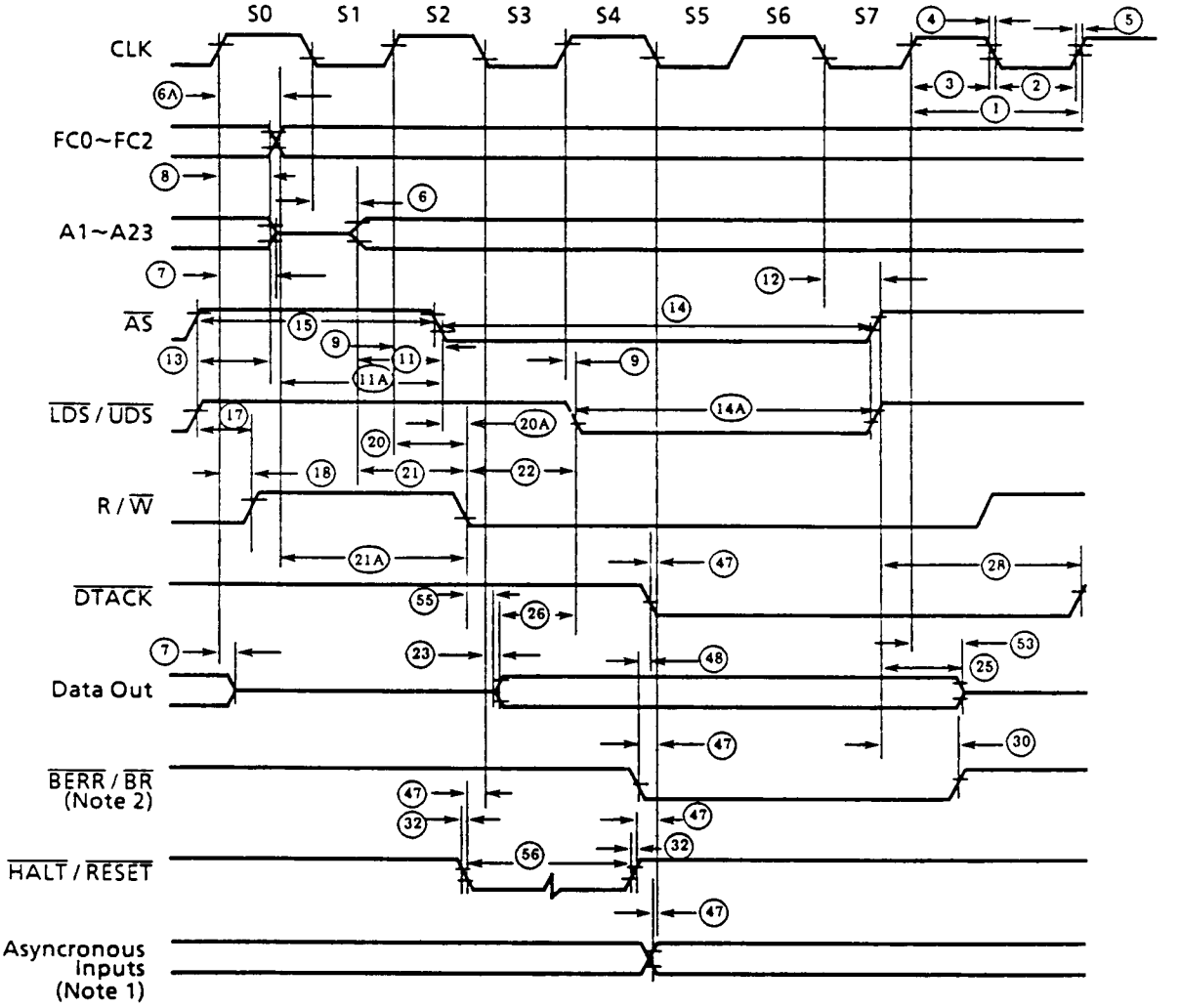
Note : 1. Setup time for the asynchronous inputs $\overline{BGACK}$, $\overline{IPL0}$~$\overline{IPL2}$ guarantees their recognitions at the next falling edge of the clock.

2. $\overline{BR}$ need fall at this time only in order to insure being recognized at the end of this bus cycle.

3. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage 2.0 volts, 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 volt and 2.0 volts.

Figure 8.2 Read Cycle Timing Diagram

These waveforms should onluy be referenced in regard to the edge-to-edge reasurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions andf their related diagrams for device operation.



Note : 1. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted.
The voltage swing through this range should start outside and pass through the range such that the rise of fall will be linear between 0.8 volt and 2.0 volt.

2. Because of loading variation, R/$\overline{W}$ may be valid after $\overline{AS}$ even through both are initiated by the rising edge of S2 (Specification 20A).

Figure 8.3 Write Cycle Timing Diagram

8.5 AC Electrical Specifications – Bus Arbitration

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figures 8.4, 8.5 and 8.6)

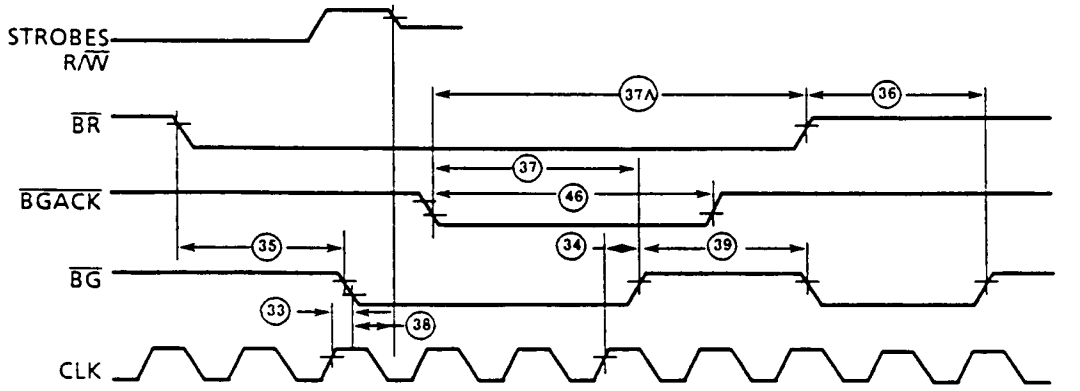
Num.	Characteristic	Symbol	8MHz		10MHz		12.5MHz		16.67MHz*		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
7	Clock High to Address, Data Bus High Impedance	tCHADZ	–	80	–	70	–	60	–	50	ns
16	Clock High to Control Bus High Impedance	tCHCZ	–	80	–	70	–	60	–	50	ns
33	Clock High to $\overline{BG}$ Low	tCHGL	–	62	–	50	–	40	0	30	ns
34	Clock High to $\overline{BG}$ High	tCHGH	–	62	–	50	–	40	0	30	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	tBRLGL	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
36[1]	$\overline{BR}$ High to $\overline{BG}$ High	tBKHGH	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	tGALGH	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
37A[2]	$\overline{BGACK}$ Low to $\overline{BR}$ High	tGALBRH	20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	10	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	tGLZ	–	80	–	70	–	60	–	50	ns
39	$\overline{BG}$ Width High	tGH	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
46	$\overline{BGACK}$ Width Low	tGAL	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
47	Asynchronous Input Setup Time	tASI	10	–	10	–	10	–	5	–	ns
57	$\overline{BGACK}$ High to Control Bus Driven	tGABD	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
58[1]	$\overline{BG}$ High to Control Bus Driven	tGHBD	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.

Note: 1. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.

2. The minimum value must to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reasserted.

* : Provision

These waveforms (Figures 8.4, 8.5 and 8.6) should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



Note : Setup time to the clock (#47) for the asynchronous inputs $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL0}$-$\overline{IPL2}$ guarantees their recognition at the next falling edge of the clock.

Figure 8.4 Bus Arbitration Timing Diagram

8.6 AC Electrical Specifications – Peripheral

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.5~8.13)

Num.	Item	Symbol	12.5MHz		16.67MHz*		Unit
			MIN	MAX	MIN	MAX	
47	Asynchronous Input setup Time	tASI	10	–	10	–	ns
101	Delay Time from $\overline{AS}$	tDAS	–	10	–	10	ns
102	Clock Low to TOUT High	tCLTO	–	70	–	70	ns
103	BCLK Peviod	tBCYC	125	–	125	–	ns
104	BClk Width (Low)	tBCL	55	–	55	–	ns
105	BCLK Width (High)	tBCH	55	–	55	–	ns
106	BCLK Rise Time	tBCr	–	10	–	10	ns
107	BCLK Fall Time	tBCf	–	10	–	10	ns
108	$\overline{LDS}$ (High) to $\overline{RTS0}$ (Low)	tDSMC	–	140	–	140	ns
109	$\overline{CTS0}$ (Low) to $\overline{LDS}$ (Low)	tMCDS	50	–	50	–	ns
113	Clock (Low) to $\overline{RAS}$, $\overline{CAS}$ Assert	tCLNRC	40	–	30	ns	ns
114	$\overline{AS}$, $\overline{DS}$ (High) to $\overline{RAS}$, $\overline{CAS}$ Negated	tADSRC	40	–	30	ns	ns
115	$\overline{RAS}$ Precharge Time	tRP	1.5	–	1.5	–	CLK Per
116	Clock (Low) to Row Address valid	tCLNRO					ns
117	Clock (High) to Column Address Valid	tCLCO					ns
118	$\overline{RAS}$-$\overline{CAS}$ Delay Time	tRCD	1.0	–	1.0	–	CLK Per
119	$\overline{RAS}$ Pulse Width (Read / Write)	tRAS	2.0	–	2.0	–	CLK Per
120	$\overline{DTACK}$ (Low) to $\overline{CAS}$ Assert	tDTCCA					ns
121	Clock (High) to $\overline{RAS}$, $\overline{CAS}$ Assert	tCLRCA					ns
122	Clock (High) to $\overline{RAS}$, $\overline{CAS}$ Negate	tCLRCN					ns
123	$\overline{CAS}$ Setup Time	tCSR	1.0	–	1.0	–	CLK Per

201289

*: provision

(V_{CC} = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.14~8.18)

Num.	Item	Symbol	12.5MHz		16.67MHz		Unit
			MIN	MAX	MIN	MAX	
124	LDS (High) to P0, P1 Output	tDPW1	40	—	40	—	ns
125	P0, P1 Input Setup Time	tPRD1	1.5	—	1.5	—	Clk. Per.
126	P0, P1 Input Hold Time	tPHD1	0	—	0	—	ns
127	$\overline{LDS}$ (Low) to P2 Output	tDPW2	40	—	30	—	ns
128	P2 Input Setup Time	tPRD2	10	—	10	—	ns
129	P2 Input Hold Time	tPHD2	40	—	30	—	ns
130	$\overline{DREQ}$ Input Setup Time	tDASI	40	—	30	—	ns
131	$\overline{DREQ}$ Input Hold Time	tDCH	40	—	30	—	ns
132	$\overline{BGACK}$ (Low) to $\overline{DREQ}$ (High)	tGALRQH	0	—	0	—	ns
133	Clock to $\overline{BR}$ (Low)	tCHBRL	—	10	—	10	ns
134	Clock (Low) to Bus Drive	tGALDBD	—	40	—	30	ns
135	$\overline{BGACK}$ (High) to Bus Hz	tGAHZ	—	40	—	30	ns
136	CPU Bus Hz to $\overline{BGACK}$ (Low)	tZGAL	—	1.0	—	1.0	Clk. Per.
137	Clock (High) to $\overline{BGACK}$ (Low)	tCHGAL	—	40	—	30	ns
138	Clock (High) to $\overline{BGACK}$ (High)	tCHGAH	—	40	—	30	ns
139	$\overline{AS}$ (Low) to $\overline{DTEND}$ Input Setup Time	tASLEDL	0	—	0	—	ns
140	$\overline{DTEND}$ (Low) Width	tEDW	1.0	—	1.0	—	Clk. Per.
141	Clock (High) to $\overline{DTEND}$ (Low)	tCHEDL	—	10	—	10	ns
142	Clock (High) to $\overline{DTEND}$ (High)	tCHEDH	—	10	—	10	ns
143	Clock (Low) to FC0-2 Valid	tGALFC	—	40	—	30	ns
144	$\overline{BGACK}$ (High) to FC 0-2 Hz	tDFCZ	—	20	—	20	ns
145	Clock (Low) to Control Bus Valid	tGALCSV	—	40	—	30	ns
146	$\overline{BGACK}$ (High) to Control Bus Hz	tGAHCSZ	—	20	—	20	ns

051289

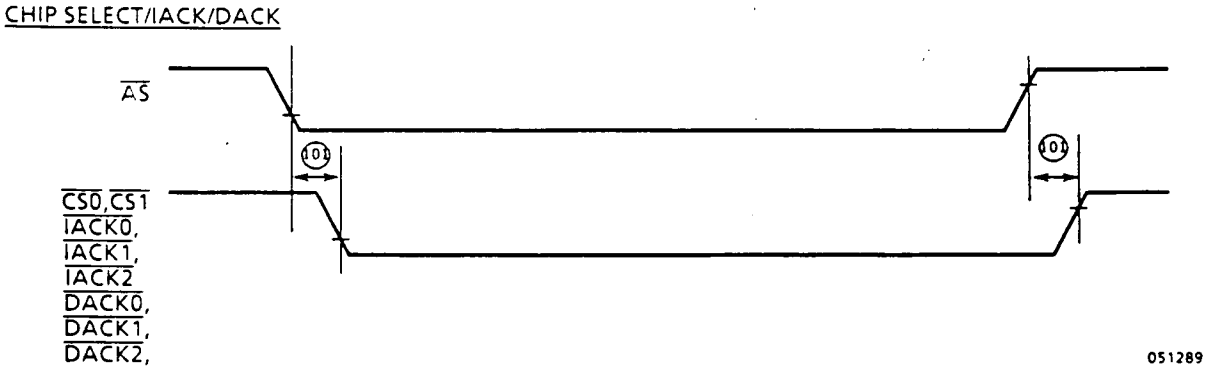


Figure 8.5 $\overline{CS}$, $\overline{IACK}$ Timming

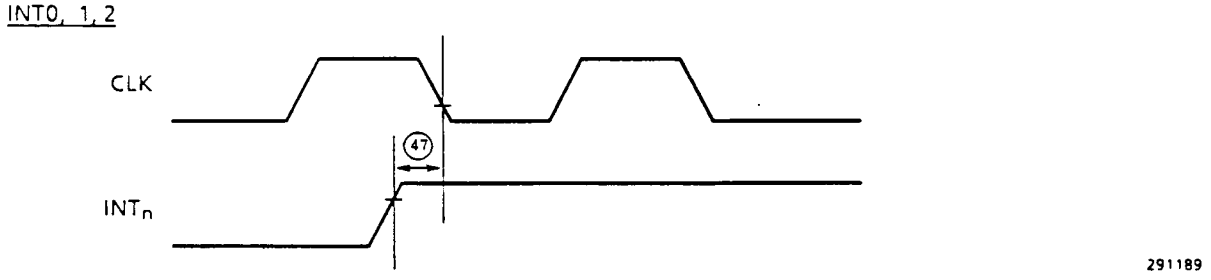


Figure 8.6 Interrupt Request Timming

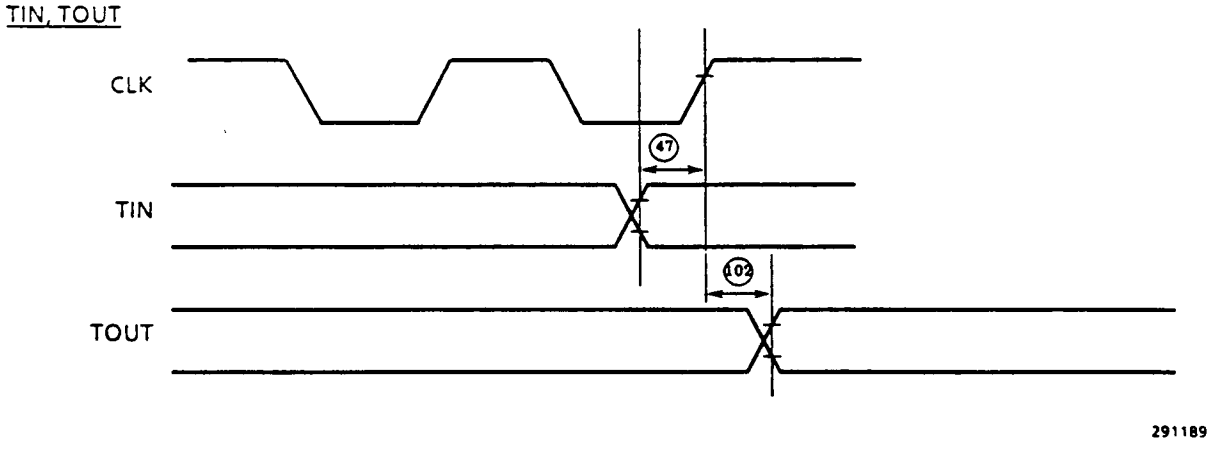


Figure 8.7 Timer Input / Output Timming

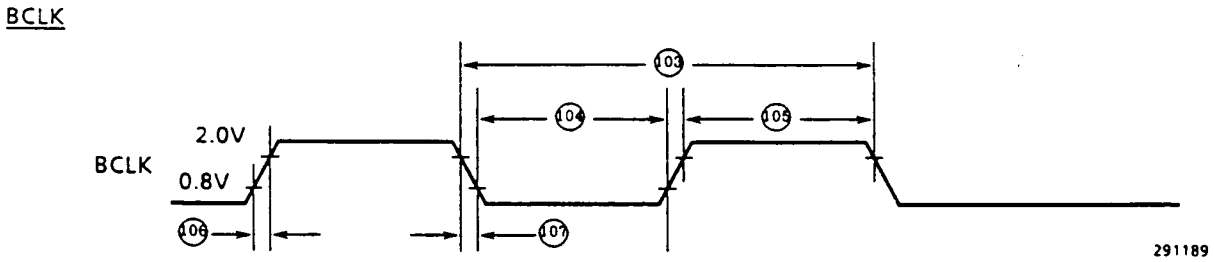


Figure 8.8 Baud Rate Clock Timming

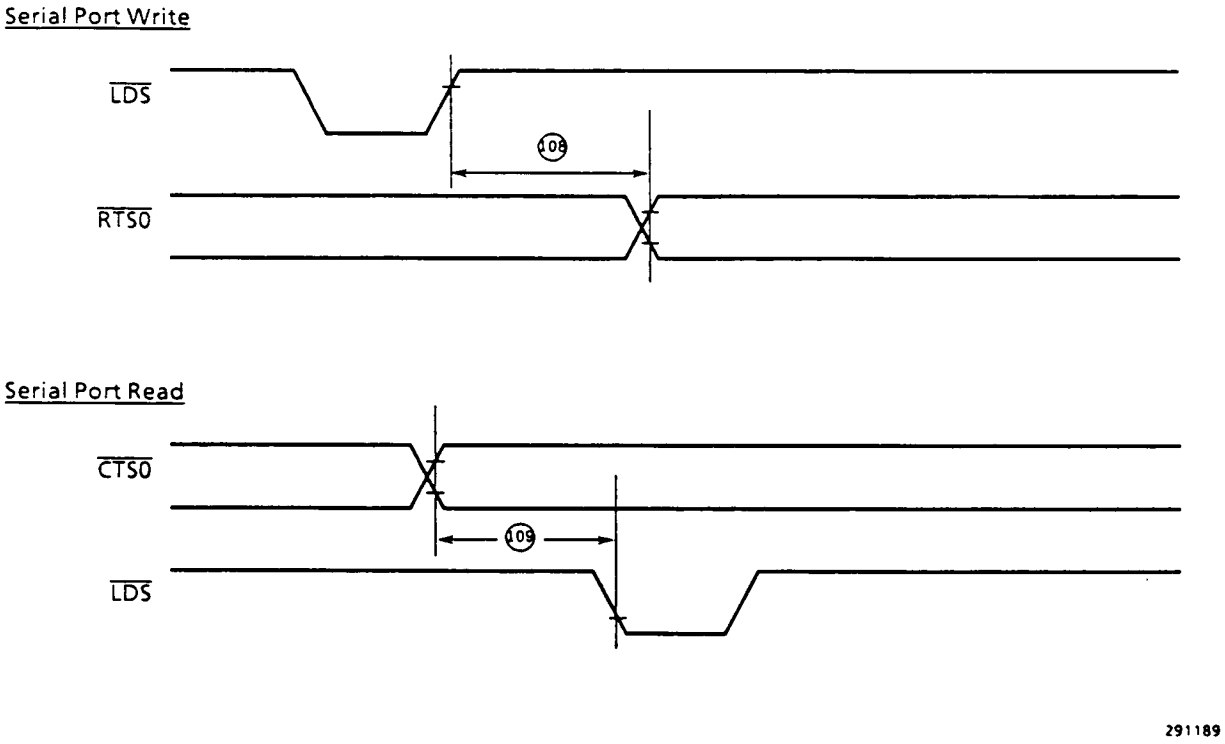


Figure 8.9 Serial Port Timming

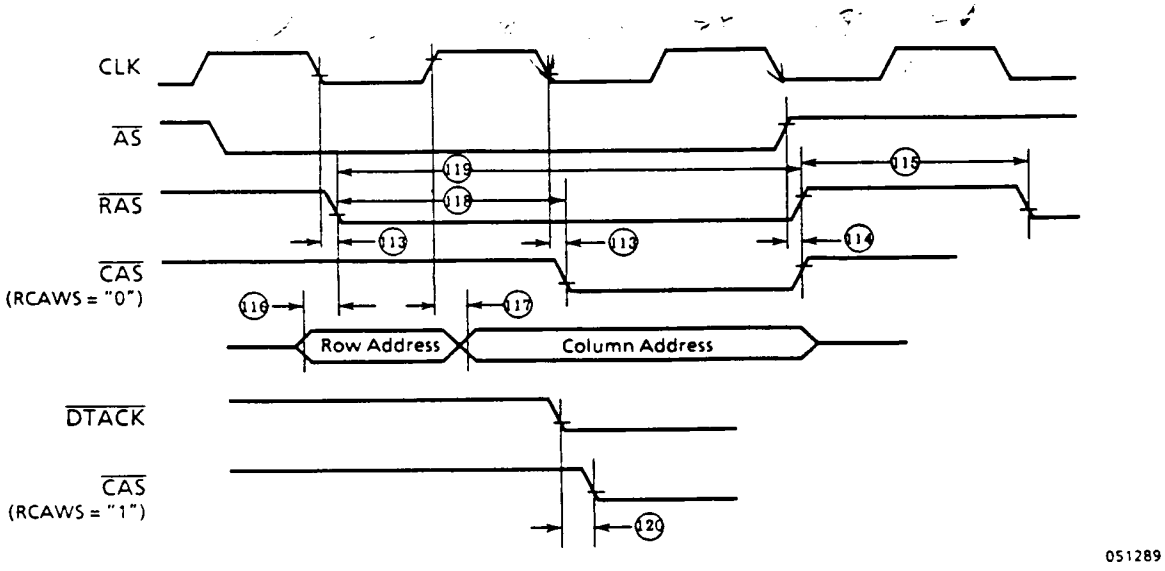


Figure 8.10 DRAM Read / Write

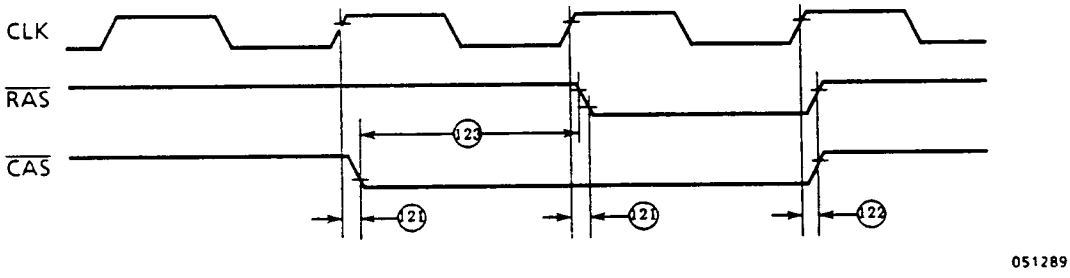


Figure 8.11 DRAM Refresh

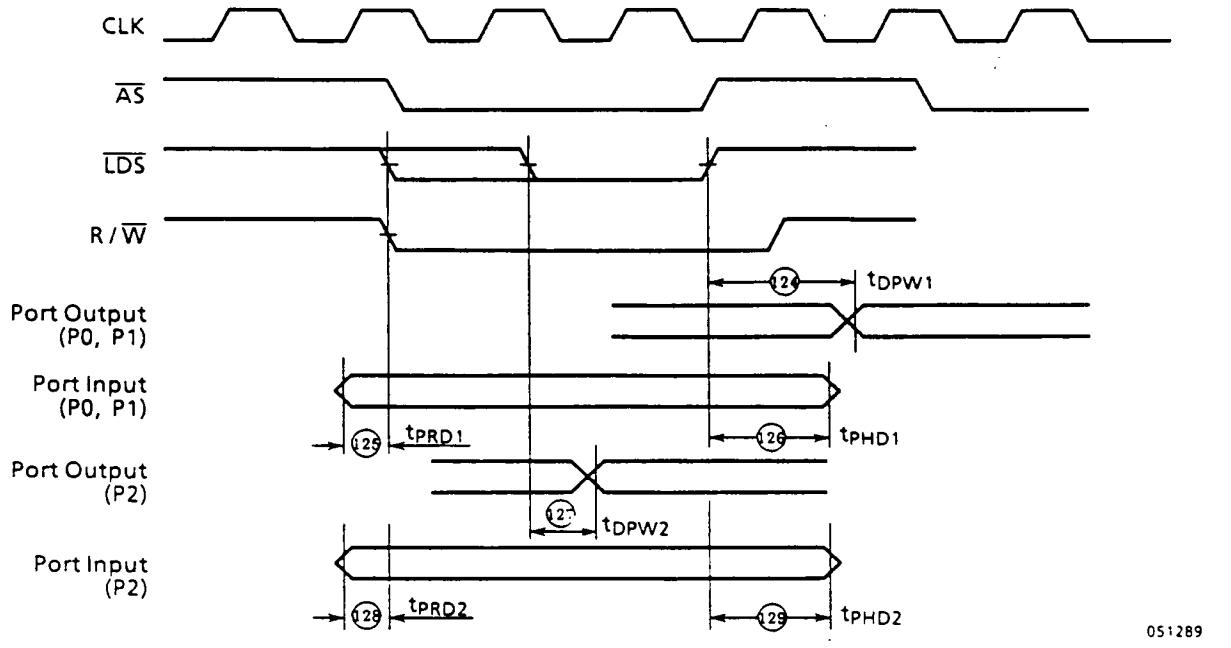


Figure 8.12

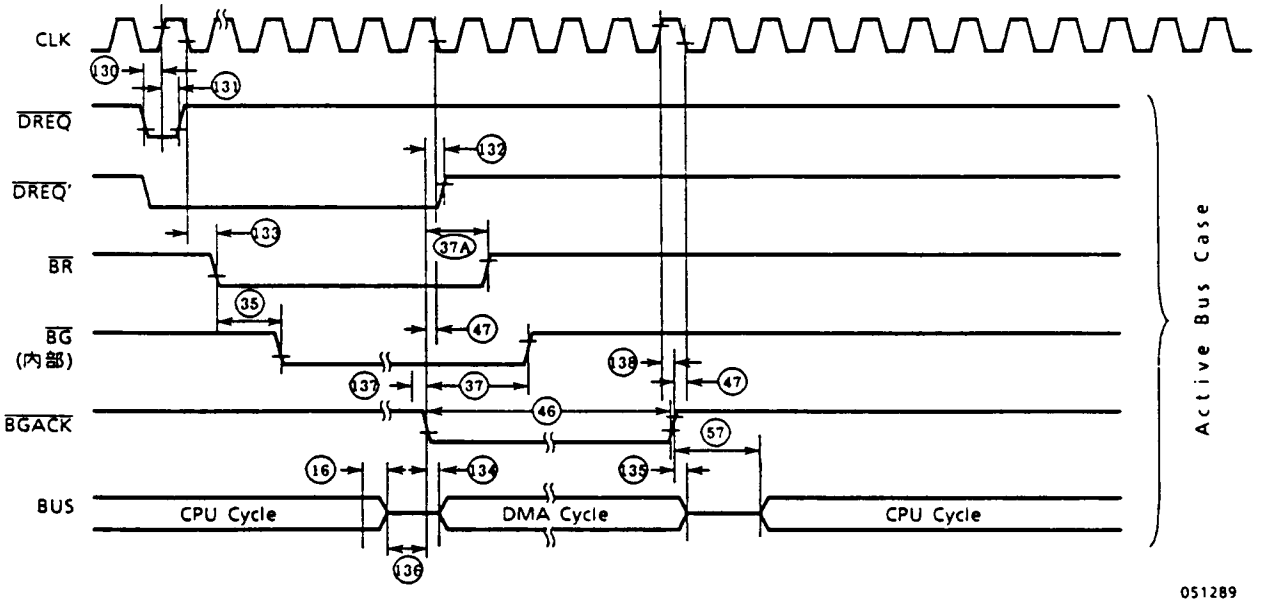


Figure 8.13

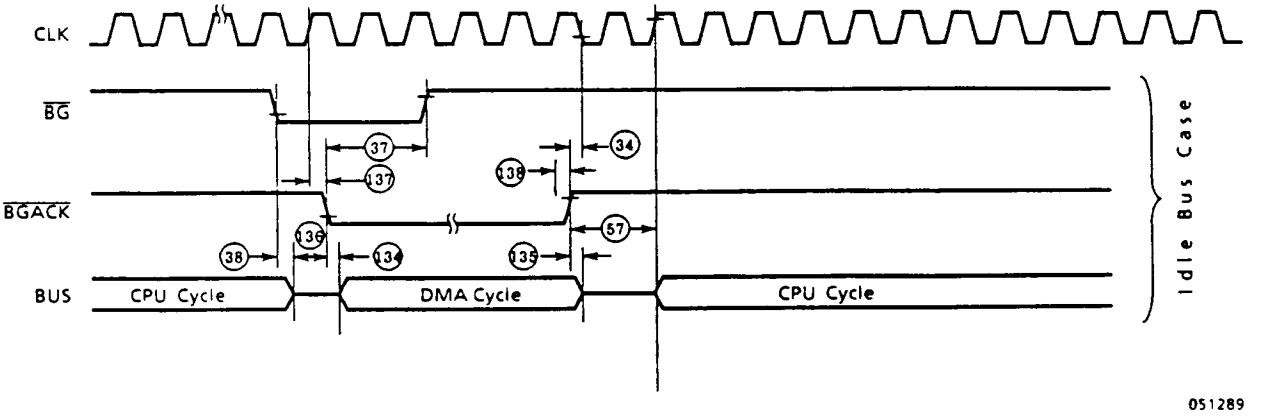


Figure 8.14

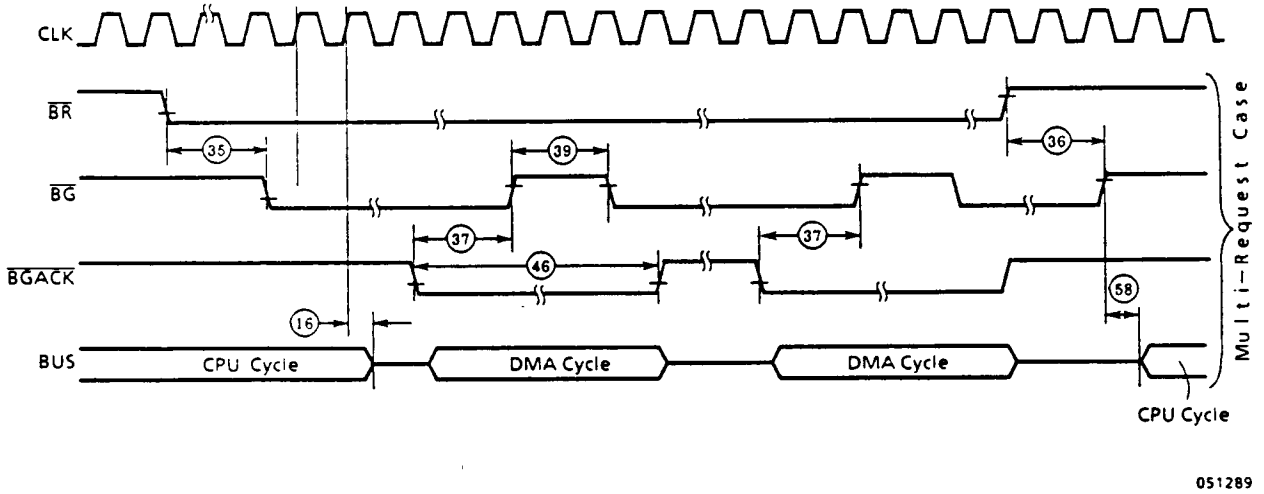


Figure 8.15

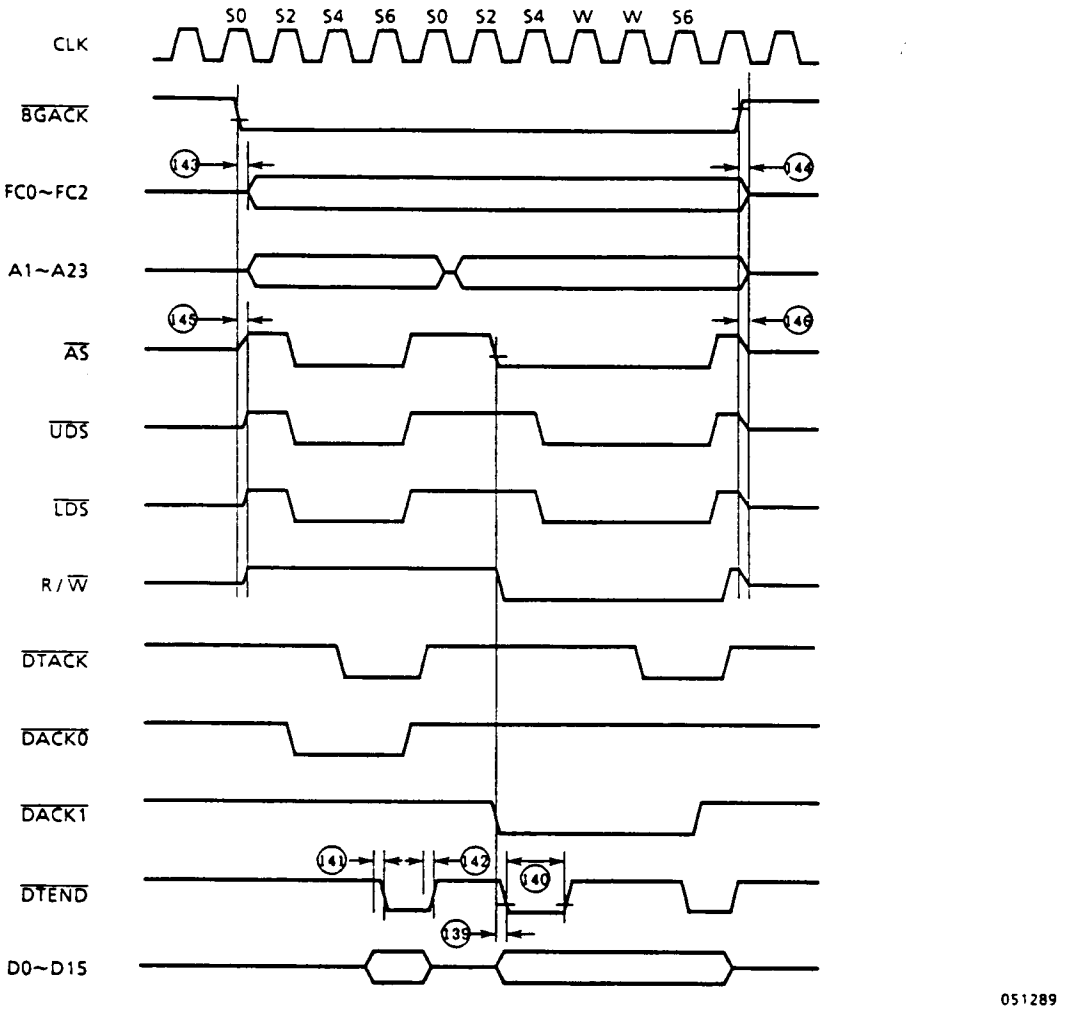


Figure 8.16

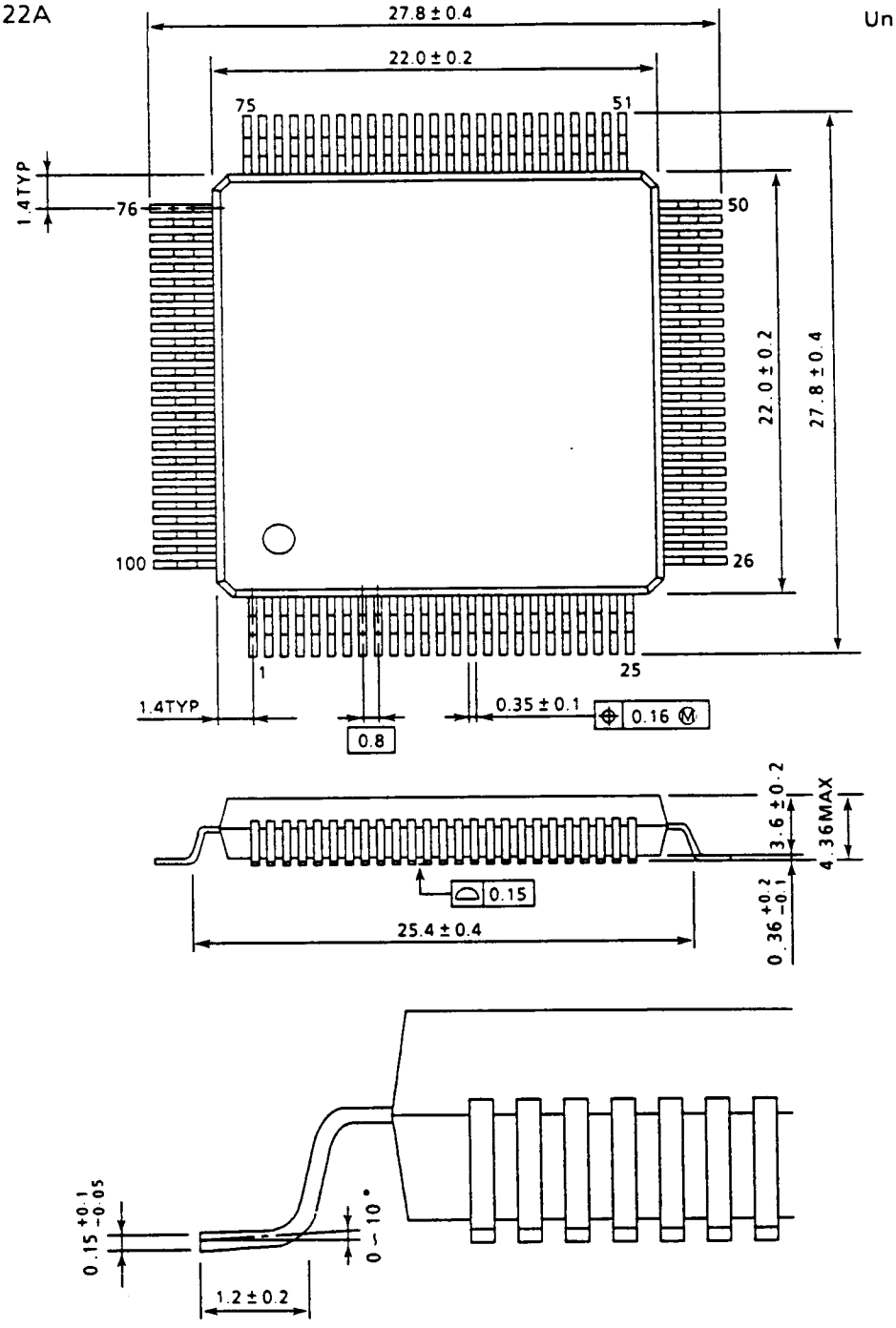
9. MECHANICAL DATA

This section contains the package dimensions for the TMP68303.

9.1 PACKAGE DIMENSIONS

QFP100-P-2222A

Unit : mm



The products described in this document are strategic products subject to COCOM regulations.
They should not be exported without authorization from the appropriate governmental authorities.

MPU-202