MC68040

DESIGNER'S HANDBOOK

MOTOROLA INC., 1990

PREFACE

The complete documentation package for the MC68040 consist of the MC68040UM/AD *MC68040 User's Manual,* the M68000PM/AD *M68000 Programmer's Reference Manual*, the MC68040/D *MC68040 Technical Summary*, and the MC68040DH/AD *MC68040 Designer's Handbook.*

The *MC68040 User's Manual* describes the capabilities, operation, and programming of the MC68040 32-bit third-generation microprocessor. The *M68000 Programmer's Reference Manual* contains the complete instruction set for all the M68000 family. The *MC68040 Technical Summary.* contains an overview of the MC68040 and includes the electrical specifications. The *MC68040 Designer's Handbook* contains applications and system design information.

This Designer's Handbook is organized as follows:

Section 1	Introduction
Section 2	Instruction Timing
Section 3	Floating-Point Emulation
Section 4	MC68040 Thermal Considerations
Section 5	Transmission Lines
Section 6	SPICE
Section 7	MC68040 To MC68030 Bus Adapter Application
Section 8	IEEE P1149.1 Test Access Port
Section 9	Bus Interface Considerations and Examples
Appendix A	ASCII Conversion Table
Appendix B	Product Support

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

SECTION 1
INTRODUCTION

This manual provides application examples and other support information for the system designer using the MC68040 32-bit microprocessor and is designed as a dynamic document. This means that new applications and information are provided on a continuing basis.

1.1 INSTRUCTION SET OVERVIEW

The instructions provided by the MC68040 are listed in Table 1-1 and 1-2. The instruction set has been tailored to support high-level languages and is optimized for those instructions most commonly executed; however, all instructions listed are fully supported. Many instructions operate on bytes, words, and long words, and most instructions can use any of the addressing modes of Table 1-3.

The floating-point instructions for the MC68040 are a commonly used subset of the MC68881/MC68882 instruction set with new arithmetic instructions to explicitly select single- or double-precision rounding. The remaining unimplemented instructions are less frequently used and are efficiently emulated in software, maintaining compatibility with the MC68881/MC68882 floating-point coprocessors.

The MC68040 instruction set includes MOVE16, a new user instruction allowing high-speed transfers of 16-byte blocks between external devices such as memory to memory or coprocessor to memory. For detailed information on the MC68040 instruction set, refer to M68000PM/AD, *M68000 Programmer's Reference Manual.*

Table 1-1. MC68040 Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ASL,ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BEFXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch
DIVS,DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EXG	Exchange Registers
EXT, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right

Mnemonic	Description
MOVE	Move
MOVEA	Move Address
MOVE CCR	Move Condition Code Register
MOVE SR	Move Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply -
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
PACK	Pack BCD
PEA	Push Effective Address
PFLUSH	No Effect
PLOAD	No Effect
PMOVE	Move to/from ACx Registers
PTEST	Test Address in ACX Registers
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore Codes
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test Operand and Set
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

Table 1-2. MC68040 Coprocessor Instructions

cpBCC	Branch Conditionally
cpDBcc	Test Coprocessor Condition, Decrement and Branch
cpGEN	Coprocessor General Instruction
cpRESTORE	Restore Internal State of Coprocessor
cpSAVE	Save Internal State of Coprocessor
cpScc	Set Conditionally
cpTRAPcc	Trap Conditionally

1.2 PROGRAMMING MODEL

Figure 1-1 shows the registers depicted in the programming model. The registers are partitioned into user and supervisor levels of privilege. User programs operate in the lower privilege mode and can only use the resources of the user model. The supervisor mode has unrestricted access to all processor resources.

The integer portion of the user programming model, consisting of 16 general-purpose, 32-bit registers and two control registers, is the same as the user programming model of the MC68030. The MC68040 user programming model also incorporates the MC68882 programming model consisting of eight, 80-bit floating-point data registers, a floating-point control register, a floating-point status register, and a floating-point instruction address register.

Registers D7–D0 are data registers containing operands for bit and bit field (1 to 32 bits), byte (8 bit), word (16 bit), long-word (32 bit), and quad-word (64 bit) operations. Registers A6–A0 and the stack pointer registers (user, interrupt, and master) are address registers that may be used as software stack pointers or base address registers. Register A7 is the user stack pointer in user mode and is either the interrupt or master stack pointer (A7' or A7") in supervisor mode. In supervisor mode, the active stack pointer (interrupt or master) is selected based on the setting of the master (M) bit in the status register (SR). In addition, the address registers may be used for word and long-word operations. All of the 16 general-purpose registers (D7–D0, A7–A0 in Figure 1-2) may be used as index registers.

The eight, 80-bit, floating-point data registers (FP7–FP0) are comparable to the integer data registers (D7–D0) of all M68000 Family processors. Floating-point data registers always contain extended-precision numbers. All external operands, regardless of the data format, are converted to extended-precision values before being used in any floating-point calculation or stored in a floating-point data register.

The program counter (PC) usually contains the address of the instruction being executed by the MC68040. The SR contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program. The lower byte of the SR is accessible in user mode as the condition code register (CCR). Access to the upper byte of the SR, which contains operation control information, is restricted to the supervisor mode.

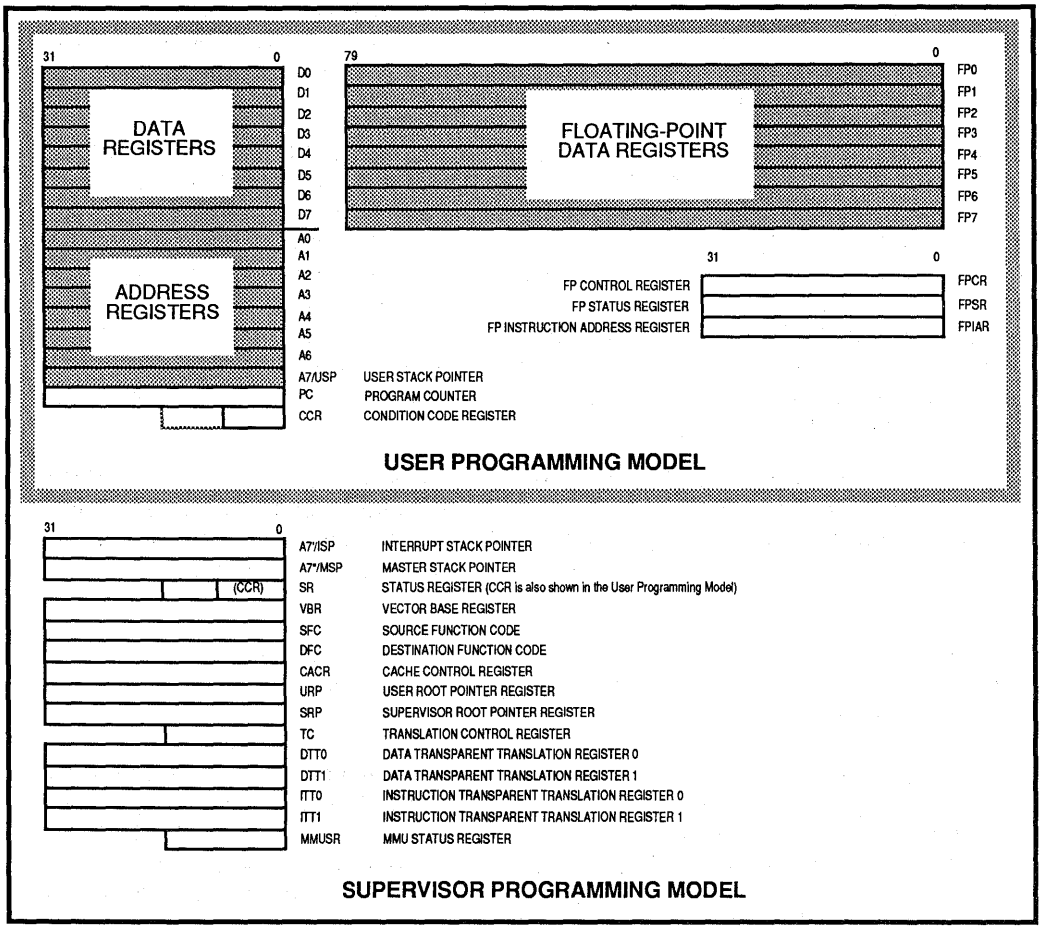


Figure 1-1. Programming Model

The base address of the exception vector table is stored in the vector base register (VBR).

Alternate function code registers, SFC and DFC (source and destination), contain 3-bit function codes, which can be considered extensions of the 32-bit logical address. Function codes are automatically generated by the processor to select address spaces for data and program accesses in the user and supervisor modes. The alternate function code registers are used by certain instructions to explicitly specify the function codes for various operations.

The cache control register (CACR) controls enabling of the on-chip instruction and data caches of the MC68040.

The supervisor root pointer (SRP) and user root pointer (URP) registers point to the root of the address translation table tree to be used for supervisor mode and user mode accesses. The URP is used if FC2 of the logical address is zero, and the SRP is used if FC2 is one.

The translation control register (TC) enables logical-to-physical address translation and selects either 4K or 8K page sizes. There are four transparent translation registers: ITT0 and ITT1 are for instruction accesses; DTT0 and DTT1 for data accesses. These registers allow portions of the logical address space to be transparently mapped and accessed without the use of resident descriptors in an address translation cache (ATC). The MMU status register (MMUSR) contains status information derived from the execution of a PTEST instruction. The PTEST instruction searches the translation tables for the logical address as specified by this instruction's effective address field and the DFC, and returns status information corresponding to the translation.

The 32-bit floating-point control register (FPCR) contains an exception enable byte that enables/disables traps for each class of floating-point exceptions and a mode byte that sets the user-selectable modes. The floating-point status register (FPSR) contains a condition code byte, quotient bits, an exception status byte, and an accrued exception byte.

For the subset of the FPU instructions that generate exception traps, the 32-bit floating-point instruction address register (FPIAR) is loaded with the logical address of an instruction before the instruction is executed. This address can then be used by a floating-point exception handler to locate a floating-point instruction that has caused an exception. The floating-point instructions FMOVE (to/from the FPCR, FPSR, or FPIAR) and FMOVEM cannot generate floating-point exceptions; therefore, these instructions do not modify the FPIAR. Thus, the FMOVE and FMOVEM instructions can be used to read the FPIAR in the trap handler without changing the previous value. For detailed information on the MC68040 registers refer to MC68040UM/AD, *MC68040 User's Manual.*

1.3 ADDRESSING MODES

Table 1-3 lists the MC68040 addressing modes. The register indirect addressing modes support postincrement, predecrement, offset, and indexing, which are particularly useful for handling data structures common to sophisticated applications and high-level languages. The program counter indirect mode also has indexing and offset capabilities; this addressing mode is typically required to support position-independent software. In addition to these addressing modes, the MC68040 provides index sizing and scaling features that enhance software performance. Data formats are supported orthogonally by all arithmetic operations and by all appropriate addressing modes.

Table 1-3. Addressing Modes

Addressing Modes	Syntax
Register Direct	
Data Register Direct	Dn
Address Register Direct	An
Register Indirect	
Address Register Indirect	(An)
Address Register Indirect with Postincrement	(An)+
Address Register Indirect with Predecrement	-(An)
Address Register Indirect with Displacement	(d_{16},An)
Register Indirect with Index	
Address Register Indirect with Index (8-Bit Displacement)	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	(bd,An,Xn)
Memory Indirect	
Memory Indirect Post-Indexed	([bd,An],Xn,od)
Memory Indirect Pre-Indexed	([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d_{16},PC)
Program Counter Indirect with Index	
PC Indirect with Index (8-Bit Displacement)	(d_8,PC,Xn)
PC Indirect with Index (Base Displacement)	(bd,PC,Xn)
Program Counter Memory Indirect	
PC Memory Indirect Post-Indexed	([bd,PC],Xn,od)
PC Memory Indirect Pre-Indexed	([bd,PC,Xn],od)
Absolute	
Absolute Short	xxx.W
Absolute Long	xxx.L
Immediate	#<data>

NOTES:

Dn = Data Register, D7-D0

An = Address Register, A7-A0

d_8, d_{16} = A twos-complement or sign-extended displacement; added as part of the effective address calculation; size is 8 (d_8) or 16 (d_{16}) bits; when omitted, assemblers use a value of zero.

Xn = Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.

bd = A twos-complement base displacement; when present, size can be 16 or 32 bits.

od = Outer displacement, added as part of effective address calculation after any memory indirection; use is optional with size of 16 or 32 bits.

PC = Program Counter

<data> = Immediate value of 8, 16, or 32 bits

() = Effective Address

[] = Used as indirect access to long-word address.

SECTION 2
INTEGER INSTRUCTION TIMING

This section provides information on the integer instruction timing. Some of the timining specifications are incomplete due to pending completion of timing evaluation. These will be provide in the near future after completion of the timing evaluation.

2.1 FETCH EFFECTIVE ADDRESS

Table 2-1 provides the single effective address (EA) calculation time for the different address modes. Table 2-2 lists the EA timing for using a breif format extension word. Table 2-3 list the EA timing using a full extension word. All timing assumes data cache hits with no stalls.

Table 2-1. Single Effective Address Instruction Format

Address Mode	EA CALCULATE TIME (CLOCKS)					
	A	B	C	D	E	F
Dn	1	1	1	1	1	1
An	1	1	1	—	1	—
(An)	1	1	1	1	1	1
(An)+	1	2	2	1	2	2
-(An)	1	2	2	1	2	2
(d_{16},An)	1	2	2	2	2	2
(d_{16},PC)	3	4	—	3	3	3
(xxx).W or (xxx).L	1	2	1	2	1	2
#<data>.B/W/L	1	—	—	—	—	—

Table 2-2. Brief Format Extension Word

Address Mode	EA CALCULATE TIME (CLOCKS)	
	A	B
(d_8,An,Xn)	3 (#3)	4 (#3)
(d_8,PC,Xn)	5 (#4)	—

Note: (#n) indicates the effective address calculation stalls at this clock if the previous instruction has not completed execution in the EU pipeline stage.

Table 2-3. Full Format Extension Word

Address Mode	Base Reg. = An		Base Reg. = PC	
	fea	flea	fea	flea
Register Indirect with Index (Base):				
(BR,I)	6 (#4)	7 (#5)	7 (#5)	8 (#6)
(BD,BR,I)	7 (#4)	8 (#5)	8 (#5)	9 (#6)
Memory Indirect Preindexed:				
([BR,I]) or ([BD,BR,I])	9 (#4)	10 (#5)	10 (#5)	11 (#6)
([BR,I],OD) or ([BD,BR,I],OD)	10 (#4)	11 (#5)	11 (#5)	12 (#6)
Memory Indirect Postindexed:				
([BR]) or ([BD,BR])	9 (#4)	10 (#5)	10 (#5)	11 (#6)
([BR],OD) or ([BD,BR],OD)	10 (#4)	11 (#5)	11 (#5)	12 (#6)
([BR],Xn) or ([BD,BR],Xn)	10 (#6)	11 (#7)	11 (#7)	12 (#8)
([BR],Xn,OD) or ([BD,BR],Xn,OD)	11 (#6)	12 (#7)	12 (#7)	13 (#8)

Notes:

BR = Base Register; suppressed or An/PC. If base register is suppressed, timing is still dependent on whether An or PC is specified as the base register
I = Index register; suppressed or Xn.
OD = Outer Displacement; d_{16} or d_{32}.
BD = Base Displacement; d_{16} or d_{32}.

2.2 INSTRUCTION EXECUTION TIMES

Table 2-4 lists the execution time for the MC68040.

Table 2-4. Execution Times

INSTRUCTION		EA CALC.	EAFETCH	EU	WRITEBACK
ABCD	Dy,Dx	1	null	3	—
	-(Ay),-(Ax)	2+1	1+1	3	Y
ADD	ea,Dx	TL: 1A,2A,3	1	1	—
	Dx,ea	TL: 1A,2A,3	1	1	Y
ADDA	Dy,Ax	1	null	1	—
	Ay,Ax	1	null	null	—
	(Ay),Ax	1	1	2	—
	(Ay)+,Ax	2	1	2	—
	-(Ay),Ax	2	1	2	—
	(d16,Ay),Ax	2	1	2	—
	ind,Ax	TL: 2A,3	TL	2	—
	xxx,Ax	1	1	2	—
	(d16,PC)	3	1	2	—
	ind,PC	TL: 2A,3	TL	2	—
	#xxx,Ax	1	null	null	—
ADDI	#xxx,Dn	1	null	1	—
	#xxx,mem	TL: 2A,3	TL	1	Y
ADDQ	#data,Dn	1	null	1	—
	#data,An	1	null	null	—
	#data,indexed	TL: 3	TL	1	Y
	#data,other	TL: 1C	1	1	Y
ADDX	Dy,Dx	1	null	1	—
	-(Ay),-(Ax)	2+1	1+1	2	Y
AND	same as ADD				
ANDI	same as ADDI				
ANDI	#xxx,CCR	1	null	4	—
ANDI	#xxx,SR				
ASL	Dx,Dy	1	null	3/4/4/4	—
	#count,Dy	1	null	2/3/3/3	—
	<EA>	TL: 1A,2A,3	TL	3	Y
ASR	Dx,Dy	1	null	3/3/4/4	—
	#count,Dy	1	null	2/2/3/3	—
	<EA>	TL: 1A,2A,3	TL	2	Y
Bcc	taken	2	null	null	—
	not taken	3	null	null	—
BCHG	Dx,Dy	1	null	4	—
	Dx,mem	TL: 1A,2A,3	TL	4	Y
	#data,Dy	1	null	3	—
	#data,mem	TL: 1D,2A,3	TL	3	Y
BCLR	same as BCHG				
BFxxx					
BKPT					
BRA		2	null	null	—
BSR					
BSET	same as BCHG				

Table 2-4. Execution Times — continued

INSTRUCTION		EA CALC.	EA FETCH	EU	WRITE BACK
BTST	Dx,Dy	1	null	2	—
	Dx,mem	TL: 1A,2A,3	TL	2	—
	#data,Dy	1	null	1	—
	#data,mem	TL: 1D,2A,3	TL	1	—
CAS					
CAS2					
CHK					
CHK2					
CLR	Dy	1	null	1	—
	<ea>	TL: 1A,2A,3	TL	1	Y
CMP	Dy,Dx	1	null	1	—
	Ay,Dx	1	null	1	—
	mem,Dx	TL: 1A,2A,3	TL	1	—
CMPA.W	Dy,Ax	1	null	2	—
	Ay,Ax	1	null	2	—
	mem,Ax	TL: 1E,2A,3	TL	2	—
CMPA.L	Dy,Ax	1	null	1	—
	Ay,Ax	1	null	1	—
	mem,Ax	TL: 1E,2A,3	TL	1	—
CMPI	#data,Dy	1	null	1	—
	#data,mem	TL: 1F,2A,3	TL	1	—
CMPM	(Ay)+,(Ax)+	2+1	1+1	2	—
CMP2					
DBcc					
DIVS					
DIVSL					
DIVU					
DIVUL					
EOR	same as ADD Dx,ea				
EORI	same as ADDI				
EORI	#xxx,CCR - same as ADDI #xxx,CCR				
EORI	#xxx,SR				
EXG	Dy,Dx	1	null	1	—
	Ay,Ax	2	null	null	—
	Ay,Dx	1	null	1	—
EXT.W		1	null	2	—
EXT.L, EXTB.L		1	null	1	—
ILLEGAL					
Fxxx					
JMP					
JSR					
LEA	(Ay),An	1	null	null	—
	(d16,Ay)	2	null	null	—
	xxx.W/L	1	null	null	—
	(d16,PC)	4	null	null	—
	indexed	TL: 2A,3	TL-1 clock	null	—

Table 2-4. Execution Times - continued

INSTRUCTION		EA CALC.	EA FETCH	EU	WRITEBACK
LINK	Ay	3	null	1	Y
LSL, LSR	Dx,Dy	1	null	3	—
	#count,Dy	1	null	2	—
	<EA>	TL:1A,2A,3	TL	2	Y
MOVEA					
MOVE from CCR					
MOVE to CCR					
MOVE from SR					
MOVE to SR					
MOVE USP					
MOVE16					
MOVEC					
MOVEM					
MOVEP					
MOVEQ	#data,Dn	1	null	1	—
MOVES					
MULS.W	<ea>,Dx	TL:1A,2A,3	TL	15/16	—
MULS.L	<ea>,Dl <ea>,Dh:Dl	TL:1D,2A,3	TL	20	—
MULU.W	<ea>,Dx	TL:1A,2A,3	TL	14	—
MULU.L	<ea>,Dl <ea>,Dh:Dl	TL:1D,2A,3	TL	20	—
NBCD	Dy	1	null	3	—
	mem	TL:1A,2A,3	TL	2	Y
NEG	Dy	1	null	1	—
	mem	TL:1A,2A,3	TL	1	Y
NEGX	same as NEG				
NOP					
NOT	same as NEG				
OR	same as ADD				
ORI	same as ADDI				
ORI	#xxx,CCR same as ADDI #xxx,CCR				
ORI	#xxx,SR				
PACK	Dx,Dy,#adj	1	null	3	—
	-(Ay),-(Ax),#adj	3	1	3	Y
PEA	ea	TL: 1?,2B,3 +1 clock	null/TL	1	Y
PFLUSH					
PTEST					
RESET					
ROL, ROR	Dx,Dy	1	null	4	—
	#count,Dy	1	null	3	—
	<EA>	TL:1A,2A,3	TL	3	Y
ROXL, ROXR	Dx,Dy	1	null	3/6/3/ TBD	—
	#count,Dy	1	null	2/5/2/ TBD	—
	<EA>	TL:1A,2A,3	TL	2	Y

Table 2-4. Execution Times - continued

INSTRUCTION		EA CALC.	EA FETCH	EU	WRITEBACK
RTD					
RTE					
RTR					
RTS					
SBCD	Dy,Dx	1	null	3	—
	-(Ay),-(Ax)	2+1	1+1	3	Y
Scc					
STOP					
SUB	same as ADD				
SUBA	same as ADDA except:				
	Ay,Ax	1	null	2	—
	#xxx,Ax	1	null	2	—
SUBI	same as ADDI				
SUBQ	same as ADDQ except:				
	#data,An	1	null	2	—
SUBX	same as ADDX				
SWAP	Dy	1	null	2	—
TAS					
TRAP					
TRAPcc					
TRAPV					
TST	Dy	1	null	1	—
	Ay	1	null	1	—
	mem	TL:1A,2A,3	TL	1	—
UNLK	Ay	2	1	1	—
UNPK	Dx,Dy,#adj	1	null	4	—
	-(Ay),-(Ax),#adj	3	1	4	Y

EA CALCULATE:

Instruction execution time in the effective address calculate (EAC) pipe stage is shown as either an absolute clock count or a reference to a lookup table for EA calculation. The notation TL: 1x,2y,3 indicates that column x of Table 2-1, y of Table 2-2, and Table 2-3 list the execution time in the EA calculate stage for the different addressing modes. The "fea" columns in Table 2-3 are for use with instructions which specify a single effective address (for example ADD Dx,<EA>). The "fiea" columns are for instructions which also specify an immediate operand (for example ADDI #xxx,<EA>).

Dual EA instructions such as ABCD -(Ay),-(Ax) require two calculations in the EAC stage and two memory fetches. Due to pipelining, the fetch of the first operand occurs in the same clock as the EA calculation for the second operand.

EA FETCH:

One clock is required in the EA fetch stage for each memory access to obtain an operand. An EA fetch value of "Null" indicates no operands are fetched, but the instruction still requires one clock to pass through the EA fetch pipe stage. A notation of "TL" indicates the the number of fetches is determined by the addressing mode.

EXECUTE UNIT (EU):

This lists the number of clocks required in the EU to execute the instruction. Rotate instructions have times listed as T1/T2/T3/T4 where:

T1 - clocks for shift count = 0
T2 - clocks for 0 < shift count < operation size
T3 - clocks for shift count = operation size
T4 - clocks for shift count > operation size

WRITEBACK:

"Y" in this column indicates the instruction generates a pending writeback to memory.

2.3 MOVE INSTRUCTION TIMING

The MOVE instruction timing tables indicate the number of clocks required in the effective address calculation (EAC) pipeline stage, and when the execute unit (EU) stage must be available to provide register information if required for the effective address calculation. Table 2-5 list total EA calculation times for MOVE instructions specifying both effective addresses with any combination of single effective address format or brief format extension word.

Table 2-6 lists source and destination effective address calculation times for MOVE instructions which specify one or both of the EAs using full format extension words. The total time in the EAC is determined by adding the EA calculation time for the source EA to the time for the destination EA. The EU must complete execution of any prior instructions by the clock time indicated by (#T) for the brief and full formats. After fetching the source operand from memory (if required) in the effective address fetch (EAF) stage, the MOVE completes in one clock in the EU stage. Examples 1–3 illustrate the pipeline operation.

Table 2-5. MOVE with Single or Brief Format Source and Destination EAs

	DESTINATION						
SOURCE	Dx	(Ax)	(Ax)+	−(Ax)	(d_{16},Ax)	xxx.W/L	(d_8,An,Xn)
Dy	1	1	1	1	1	1	3 (#3)
Ay	1	1	2	2	2	1	3 (#3)
(Ay)	1	1	2	2	2	1	4 (#3)
(Ay)+	1	2	2	2	2	2	4 (#3)
−(Ay)	1	2	2	2	2	2	4 (#3)
(d_{16},Ay)	1	2	2	2	2	2	4 (#3)
xxx.W/L	1	1	2	2	2	2	4 (#3)
(d_{16},PC)	3	3	3	3	4	4	8 (#7)
#xxx.B/W/L	1	1	2	2	2	2	3 (#3)
(d_8,An,Xn)	4 (#3)	4 (#3)	5 (#3)	5 (#3)	5 (#3)	5 (#3)	8 (#3)
(d_8,PC,Xn)	5 (#4)	5 (#4)	6 (#4)	6 (#4)	6 (#4)	6 (#4)	9 (#4)

Table 2-6. MOVE with Full Format Source or Destination EA

Address Mode	Source	Destination
Address Register Indirect with Index (Base):		
(An,I)	5 (#4)	5 (#3)
(BD,An,I)	6 (#4)	6 (#3)
Memory Indirect Preindexed:		
([An,I]) or ([BD,An,I])	8 (#4)	8 (#3)
([An,I],OD) or ([BD,An,I],OD)	9 (#4)	9 (#3)
Memory Indirect Postindexed:		
([An]) or ([BD,An])	8 (#4)	8 (#3)
([An],OD) or ([BD,An],OD)	9 (#4)	9 (#3)
([An],Xn) or ([BD,An],Xn)	9 (#6)	9 (#5)
([An],Xn,OD) or ([BD,An],Xn,OD)	10 (#6)	10 (#5)
PC Register Indirect with Index (Base):		
(PC,I)	6 (#5)	—
(BD,PC,I)	7 (#5)	—
PC Memory Indirect Preindexed:		
([PC,I]) or ([BD,PC,I])	9 (#5)	—
([PC,I],OD) or ([BD,PC,I],OD)	10 (#5)	—
PC Memory Indirect Postindexed:		
([PC]) or ([BD,PC])	9 (#5)	—
([PC],OD) or ([BD,PC],OD)	10 (#5)	—
([PC],Xn) or ([BD,PC],Xn)	10 (#7)	—
([PC],Xn,OD) or ([BD,PC],Xn,OD)	11 (#7)	—
Single or Brief Format EAs:		
Dn	1	1
An	1	—
(An)	1	1
(An)+	1	2
-(An)	1	2
(d_{16},An)	1	2
xxx.W/L	1	2
(d_{16},PC)	4	—
#xxx.B/W/L	1	—
(d_8,An,Xn)	3 (#3)	5 (#4)
(d_8,PC,Xn)	4 (#4)	—

Notes:

BR = Base Register; suppressed or An/PC. If base register is suppressed, timing is still dependent on whether An or PC is specified as the base register

I = Index register; suppressed or Xn.

OD = Outer Displacement; d_{16} or d_{32}.

BD = Base Displacement; d_{16} or d_{32}.

Example 1: MOVE Dy,Dx

Table 2-5 indicates the EAC stage takes one clock, and the EU is not required for the EA calculation. In Figure 2-1, the MOVE instruction executes in one clock in the EAC, flows through the EAF stage in the next clock without fetching a memory operand, and completes in the EU in one clock.

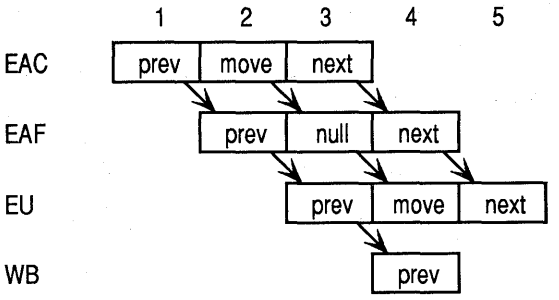


Figure 2-1. MOVE Dy,Dx Pipeline Flow

Example 2: MOVE (d_8,An,Dn),-(Ax)

Table 2-5 indicates the EAC stage takes five clocks, and the EU is required by the third clock to supply information for the EA calculation. Figure 2-2 shows the flow of the instruction through the pipeline. At clock 3, the EU must have completed the prior instruction, or the EAC operation at clock three is stalled. In clock 6 the source operand is fetched from memory. Since a memory destination is specified, a writeback is generated in clock 8.

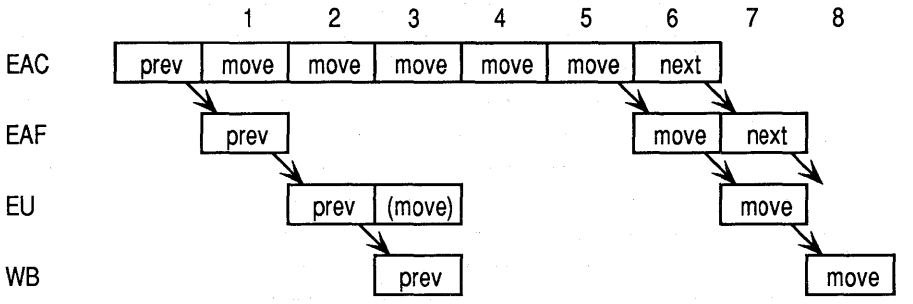


Figure 2-2. MOVE (d_8,An,Dn),-(Ax) Pipeline Flow

Example 3: MOVE An,([Ax]) - memory indirect preindexed.

For this case, Table 2-6 is used. The EA calculation requires one clock for An, and 8 clocks for ([Ax]), with the EU required during the third clock. In Figure 2-3, the move1 box in the EAC flow performs the An access, and the move2 boxes

perform the memory indirect preindexed address calculation. An EAF is required in clock 7 to load the indirect pointer from memory, which is used as the source effective address in clock 10.

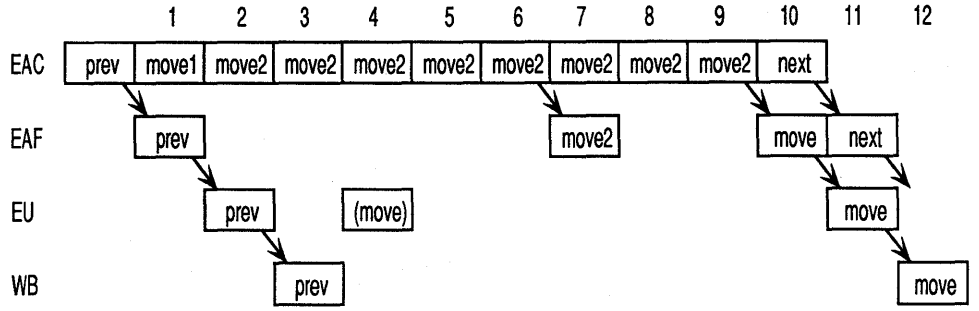


Figure 2-3. MOVE (An),([Ax]) Pipeline Flow

2.4 CACHE MAINTENANCE

This following paragraphs detail the execution time for the CINV and CPUSH instructions used to perform maintenance of the instruction and data caches.

2.4.1 Cache Invalidate Instruction Timing

Approximate execution times for the CINV instruction variants are listed in Table 2-7. During performing the actual cache invalidate operation, the EU stalls to allow previous writebacks to complete, as well as any pending instruction prefetches. As a result, the total time time required to execute a cache invalidate instruction is dependent on the prior instruction stream. Execution time of this instruction is independent of which combination of caches is selected.

Table 2-7. CINV Timing

Instruction	EU Execution Time
CINVL - Line Invalidate	9 + machine_idle
CINVP - Page Invalidate	266 + machine_idle
CINVA - Cache Invalidate	9 + machine_idle

machine_idle = number of clocks required for all pending writes and instruction prefetches to complete.

The CINV instructions also interlock operation of the EAC and EU pipeline stages to prevent a following instruction from accessing the caches until the invalidate operation is complete. The resulting integer unit pipeline flow is shown in Figure 2-4. The completion of the CINV instruction at clock 6 in the EU also allows the instruction to complete in the EAC stage, and the next instruction begins in clock 7.

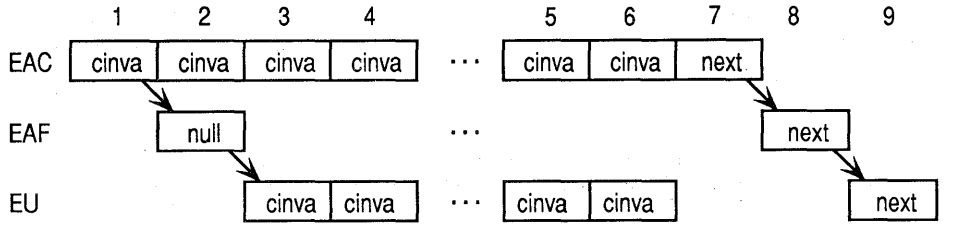


Figure 2-4. CINV Integer Instruction Flow

2.4.2 Cache Push Instruction Timing

Execution time for the CPUSH instruction is heavily dependent on several factors. Within the data cache, the number of dirty lines and size of the resulting push (either longword or line) are a major factor. Due to overlap of operations within the data cache and the bus controller, the distribution of dirty lines within the cache can also affect total execution time. Finally, the number of wait states in the push access on the bus also have an effect.

The interaction of all of the factors noted above determines the total time required to execute a CPUSH instruction. Since the distribution of dirty data within the cache is entirely dependent on the nature of the user's code, it is impossible to provide an equation for execution time that works for all code sequences. Instead, this section provides baseline information indicating best and worst case execution times for the three CPUSH instruction variants, followed by the algorithm used for performing the push operation.

Table 2-8 provides a range of execution time from best case to worst case. Best case corresponds to the cache containing no dirty entries, while worst case corresponds to all lines dirty and requiring line pushes.

Table 2-8. CPUSH Best/Worst Case Timing

Instruction	EU Execution Time	
	Best Case	Worst Case
CPUSHL	6	6 + N_{LINE} + machine_idle
CPUSHP CPUSHA	267	11 + 256*N_{LINE} + machine_idle

N_{LINE} = number clocks required in the user's system for a line transfer.
machine_idle = number of clocks required for all pending writes and instruction prefetches to complete.

A more detailed understanding of the internal operation of the MC68040 is needed to determine actual execution time for a CPUSHP or CPUSHA instruction. The interaction of three functional units, the EU pipeline stage, data cache controller, and the bus controller, is required to analyze a sequence of pushes.

The EU generates an index each clock which references one of the 256 cache lines, and sends this index to the data memory unit to push the corresponding line. Individual cache lines within each set are referenced in the order 0-1-2-3, and each set is pushed beginning with set 0 and ending with set 63. After passing the last index to the data cache, the EU stalls until any pending pushes complete before ending the CPUSH instruction. Like the CINV instructions, CPUSH also prevents any following instructions from beginning in the EAC stage until the CPUSH completes in the EU.

The data memory unit takes a single clock to check a cache line for a dirty state. If the line is dirty, the first half of the line (read when the tag is checked) is loaded into a push buffer, and the second half of the cache line is loaded into the buffer on the next clock. Note that the push buffer is not available until after the first longword transfer of a previous push has been completed. The check of the dirty cache line repeats until the push buffer is available. After checking a clean line, or completing the push buffer load for a dirty line, the data unit checks the line pointed to by the next index.

The bus controller performs the actual push transfers. For a line containing a single dirty entry, only the dirty longword is written. Lines containing two or more dirty entries are written as a line transfer, and the bus controller releases the push buffer back to the data memory unit after the first longword transfer completes.

Figures 2-5 and 2-6 illustrates the pipelining of the above operations in the data cache and bus controller which can occur when pushing several cache lines in succession.

In Figure 2-5, lines X, Y, and Z are successive lines in the data cache. Line X contains a single dirty entry, line Y is not dirty, and line Z is dirty. The status of line X is checked for a dirty state in clock 1. Since the line is dirty and the push buffer is available, the first half of the cache line is loaded into the buffer at this time. The second half is loaded in clock 2, and the external longword push bus transfer begins in clock 3. The check of line Y in clock 3 is completely overlapped with the external bus access, since line Y in not dirty and the push buffer is not required. However, the check of line Z (which is dirty) must be repeated since the push buffer has not yet been released. The load of line Z completes in clock 6 and the bus transfer begins in clock 7, two clocks after the transfer of the dirty entry for line X.

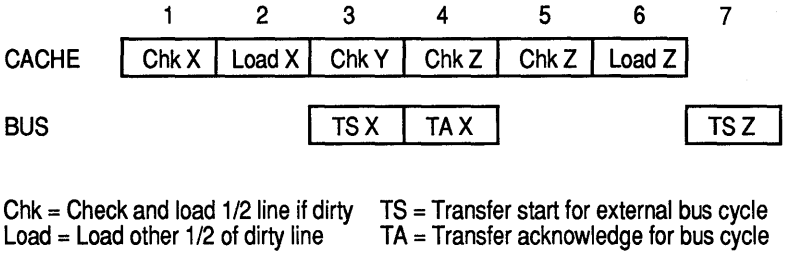


Figure 2-5. CPUSH Example 1

Figure 2-6 shows the results for a similar situation where line X contains more than one dirty entry, and must be pushed as a line. The load of the second half of line Z still occurs in clock 6, but the push transfer for line Z does not begin until clock 8. The check of line N is stalled until the external bus transfer for cache line Z begins in clock 8.

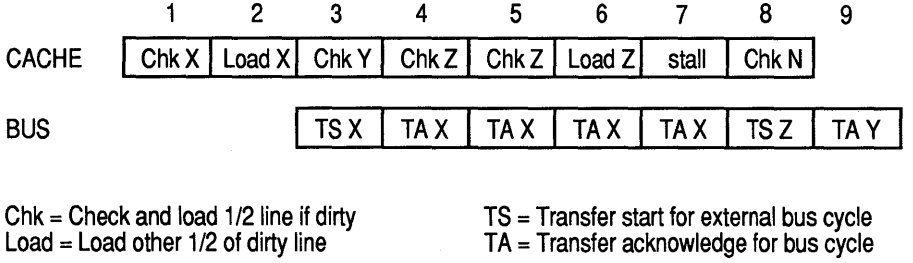


Figure 2-6. CPUSH Example 2

SECTION 3
Software Specification for an MC68040 Floating-Point Software Package

The purpose of this section is to provide an overview of the floating-point software package (FPSP) for the MC68040. The FPSP emulates the floating-point instructions of the MC68881/MC68882 which are not provided by the MC68040.

3.1 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

FPn - Floating-Point Data Register Source
FPSP - Floating-Point Software Package
FPU - Floating-Point Unit
FPx - Floating-Point Data Register

See the Glossary of References 1 for additional definitions.

3.2 PRODUCT OVERVIEW

The FPSP adds additional floating-point capabilities to the MC68040. A subset of the MC6888x instruction set is executed by the MC68040 on-chip FPU. The remaining floating-point instructions are emulated in software by the FPSP (see Reference 2). There are two types of FPSP: one for applications compiled for the MC68881/MC68882 and another for applications compiled for the MC68040 (see **3.8.2 Packaging**).

The FPSP provides:

- Arithmetic and Transcendental Instructions
- Decimal Conversions
- Exception Handlers
- MC68040 Unimplemented Data Type and Data Format Handlers
- System Call To Allow Users To Read the Exceptional Operand

There are two types of users: 1) end users who are running applications and 2) system integrators who will install the package (see **3.8.3 Site Adaptations**).

3.3 GENERAL CONSTRAINTS

The FPSP satisfies the requirements of the ANSI IEEE *Standard for Binary Floating-Point Arithmetic* 754. The FPSP runs old user code unchanged and is transparent for old code. The FPSP is easy to modify and install. The performance of the transcendental function routines is equivalent or superior to that of a 33-MHz MC68881/MC68882. The error bound is equivalent or superior to the MC68881/MC68882 (see **3.7.2 Accuracy**).

3.4 ASSUMPTIONS AND DEPENDENCIES

The FPSP can be installed into any operating system. The MC68040 FPU shall be implemented as described in Reference 2. Table 3-1 Lists the functions provided by the MC68040.

Table 3-1. Functions Provided by MC68040

Name	Description	Name	Description
FMOVE	Move to FPU	FMOVEM	Move Multiple Registers
FSMOVE	Single-Precision Move	FDMOVE	Double-Precision Move
FCMP	Compare	FABS	Absolute Value
FSABS	Single-Precision Absolute Value	FDABS	Double-Precision Absolute Value
FTST	Test	FNEG	Negate
FSNEG	Single-Precision Negate	FDNEG	Double-Precision Negate
FADD	Add	FSUB	Subtract
FDIV	Divide	FMUL	Multiply
FBcc	Branch Conditionally	FScc	Set According to Condition
FDBcc	Test Cond, Dec and Branch	FTRAPcc	Trap Conditionally
FSADD	Single-Precision Add	FSSUB	Single-Precision Subtract
FSMUL	Single-Precision Multiply	FSDIV	Single-Precision Divide
FDADD	Double-Precision Add	FDSUB	Double-Precision Subtract
FDMUL	Double-Precision Multiply	FDDIV	Double-Precision Divide
FSQRT	Square Root	FSSQRT	Single-Precision Square Root
FDSQRT	Double-Precision Square Root	FNOP	No Operation
FSAVE	Save Internal State	FRESTORE	Restore Internal State

Note : FSGLDIV and FSGLMUL are mapped as FMUL and FDIV for performance reasons

3.5 FUNCTIONAL REQUIREMENTS

The following paragraphs describe the data types and data formats and exceptions for the FPSP.

3.5.1 Data Formats and Data Types

All data formats and data types not supported by the MC68040 FPU are supported by the FPSP. Table 3-2 shows how the data formats and data types are supported.

Table 3-2. Support for Data Types and Data Formats

Data Types	Data Formats						
	SGL	DBL	EXT	Decimal	Byte	Word	Long Word
Norm	*	*	*	@	*	*	*
Zero	*	*	*	@	*	*	*
Infinity	*	*	*	@			
NaN	*	*	*	@			
Denorm	#	#	@	@			
Unnorm			@	@			

@ = supported by FPSP
* = supported by the MC68040 FPU
# = supported by FPSP after being converted to extended precision by MC68040

3.5.2 Exceptions

The main goal of the FPSP exception handlers is to provide the user with an easy path to port over existing MC68882 exception handlers for use with the MC68040. The end result is that the FPSP provides an entry point so that once this point is reached, there is an indication that an IEEE-defined trap condition exist.

3.5.2.1 BSUN — Branch/Set on Unordered. On a trap-enabled condition, the FPSP updates the floating-point instruction address register (FPIAR) by copying the PC value in the pre-instruction stack frame to the FPIAR. Once this is done, the exceptional frame is restored without clearing the exception, and the program flow goes to the FPSP provided entry point. At the entry point the MC68040 is in an exceptional state, ready to execute the user-supplied exception handler.

3.5.2.2 SNAN — SIGNALING NOT-A-NUMBER. On a trap-disabled condition, and if the destination format is B,W, or L, then the FPSP stores the most significant 8, 16, or 32 bits, respectively, of the SNAN mantissa, with the SNAN bit set, to the destination. The FPSP discards the exceptional frame, then returns to the main program flow without entering the FPSP provided entry point, hence the user-provided exception handler is not executed.

On a trap-enabled condition, the FPSP checks if the destination format is B,W, or L. Then, the FPSP stores the most significant 8, 16, or 32 bits, respectively, of the SNAN mantissa, with the SNAN bit set, to the destination. The FPSP then restores the exceptional frame without clearing the exception, and branches to the FPSP provided entry point. At the entry point, the MC68040 is in an exceptional state, ready to execute the user-supplied exception handler.

Trap Disabled Results

If the destination format is B, W, or, L, then the FPSP stores the 8 or 16 most significant bits of the SNAN significand, with the SNAN bit set, to the destination.

3.5.2.3 OPERR — OPERAND ERROR. This exception traps through vector number 52. Table 3-3 shows the operand errors generated by the MC68040. Table 3-4 shows the operand errors generated by the FPSP.

Note that the FPSP Unimplemented Instruction Handler detects and adds to the cases in which OPERR exceptions occur. Refer to Table 3-4 for these specific exception-causing conditions.

On a trap-disabled condition, the FPSP checks if the operand error is caused by an FMOVE to a B,W, or L memory or integer data register destination. If it is caused by an integer overflow or if the floating-point data register to be stored contains infinity, the FPSP stores the largest positive or negative integer that can fit in the specified destination format size. If the destination is integer and the floating-point number to be stored is a NAN, then the 8, 16, or 32 most significant bits of the NAN significand is stored as a result.

Next the FPSP checks for a false OPERR condition for an FMOVE to memory or integer data register. This condition occurs if the operand is equal to the largest negative integer representable in its format. The FPSP then stores the proper result, discards the exceptional frame, and returns to the main program flow without executing the user-supplied exception handler.

On a trap-enabled condition, the FPSP does the same functions as the above trap-disabled condition, with the exception that in the end, the FPSP restores the exceptional frame without clearing the exception and branches to the FPSP supplied entry point instead of returning to the main program flow. At the FPSP supplied entry point, the MC68040 is in an exceptional state, ready to execute the user-supplied exception handler. A system call is provided by the FPSP to calculate the exceptional operand.

Table 3-3. Operand Errors Handled by the MC68040

Instruction	Condition Causing Operand Error
FADD	(+ infinity)+(- infinity) or (- infinity)+(+ infinity)
FSUB	(+ infinity)-(+ infinity) or (- infinity)-(- infinity)
FMUL	(0) x (infinity) or (infinity) x (0)
FDIV	0 / 0 or infinity / infinity
FMOVE.BWL	Integer overflow, Source is NaN, or Source is infinity
FSQRT	Source < 0, Source = - infinity

Note : FSGLDIV and FSGLMUL are mapped as FMUL and FDIV for performance reasons

Table 3-4. Operand Errors Generated by the FPSP

Instruction	Condition Causing Operand Error
FSADD	(+ infinity)+(- infinity) or (- infinity)+(+ infinity)
FDADD	(+ infinity)+(- infinity) or (- infinity)+(+ infinity)
FSSUB	(+ infinity)-(+ infinity) or (- infinity)-(- infinity)
FDSUB	(+ infinity)-(+ infinity) or (- infinity)-(- infinity)
FSMUL	(0) x (infinity) or (infinity) x (0)
FDMUL	(0) x (infinity) or (infinity) x (0)
FSDIV	0 / 0 or infinity / infinity
FDDIV	0 / 0 or infinity / infinity
FCOS	Source is ± infinity
FSIN	Source is ± infinity
FTAN	Source is ± infinity
FACOS	Source is ± infinity, > +1, or < -1
FASIN	Source is ± infinity, > +1, or < -1
FATANH	Source is > +1, or < -1, Source = ± infinity
FSINCOS	Source is ± infinity
FGETEXP	Source is ± infinity
FGETMAN	Source is ± infinity
FLOG10	Source is < 0, Source = - infinity
FLOG2	Source is ± infinity, > +1, or < -1
FLOGN	Source is ± infinity, > +1, or < -1
FLOGNP1	Source is < -1, Source is = - infinity
FMOD	FPx is ± infinity or Source is 0, Other Operand is not a NaN
FMOVE to P	Result Exponent > 999 (Decimal) or k-Factor > +17
FREM	FPx is ± infinity or Source, Other Operand is not a NaN
FSCALE	Source is ± infinity, Other Operand is not a NaN

Trap Disabled Results

If the destination format is B, W, or L and the source operand is a NaN, then the result stored is the 8, 16, or 32 most significant bits of the significand. If the source is infinity or if the source is too large to fit in the destination format, then the result stored is the largest integer that will fit in the destination format.

There are three cases in which OPERR is wrongly signaled on the MC68040, for B, W, and L (for move_out only). These cases are the largest negative integer representable in its format (-2^7 for B, -2^{15} for W, -2^{31} for L). These cases are signaled as OPERRs even though no exception is generated. The FPSP detects these cases, stores the proper result to memory, clears the exception, and continues as if nothing occurred.

Trap Enabled Results

The FPSP provides the same results as the trap disabled case and then provides an entry point for the trap-enabled case.

3.5.2.4 OVFL — OVERFLOW. This exception traps through vector number 53.

Trap Disabled Results

On a trap-disabled case, the FPSP stores the result in the destination as determined by the rounding mode at the destination as follows:

Rounding Mode	Result
RN	Infinity, with the sign of the intermediate result
RZ	Largest magnitued number, with the sign of the intermediate result.
RM	For positive overflow, largest positive number For negative overflow, infinity
RP	For positive overflow, infinity For negative overflow, largest negative number

The FPSP then clears the appropriate exception bit in the frame and restores the non-exceptional frame into the MC68040, and then returns to the main program flow.

Trap Enabled Results

On a trap-enabled case, the FPSP actions are identical to those found in the trap-disabled case, with the exception that instead of restoring a non-exceptional frame, the original exceptional frame is restored to the MC68040 and the FPSP branches to the FPSP supplied entry point. At this entry point, the MC68040 is in an exceptional state, ready to execute the user-supplied exception handler. A system call is provided by the FPSP to calculate the exceptional operand.

3.5.2.5 UNFL — UNDERFLOW. This exception traps through vector number 51.

Trap Disabled Results

On a trap-disabled case, the FPSP stores the result in the destination as determined by the rounding mode at the destination as follows:

RN - Zero with the sign of the intermediate result.
RZ - Zero with the sign of the intermediate result.
RM - For positive underflow, +zero. For negative underflow, the smallest denormalized negative number.
RP - For positive underflow, the smallest denormalized positive number. For negative underflow, -zero.

The FPSP then clears the appropriate exception bit in the frame and restores the non-exceptional frame into the MC68040, and then returns to the main program flow.

Trap enabled results

On a trap-enabled case, the FPSP actions are identical to those found in the trap-disabled case, with the exception that instead of restoring a non-exceptional frame, the original exceptional frame is restored to the MC68040 and the FPSP branches to the FPSP supplied entry point. At this entry point, the MC68040 is in an exceptional state, ready to execute the user-supplied exception handler. A system call is provided by the FPSP to calculate the exceptional operand.

3.5.2.6 DZ — DIVIDE BY ZERO. Note that the FPSP Unimplemented Instruction Handler detects and adds to the cases in which DZ exceptions occur. Refer to Table 3-5 for these specific exception-causing conditions.

The FPSP is not needed for this exception. The user-supplied exception handler is always entered. A system call is provided by the FPSP to calculate the exceptional operand.

3.5.2.7 INEX1/INEX2 — INEXACT RESULT 1/2. Note that the FPSP Unimplemented Instruction Handler detects and allows INEX1 exceptions to occur. Furthermore, many new cases of INEX2 exceptions may be generated by the FPSP Unimplemented Instruction Handler as well. The INEX1 exception traps into this handler as well as INEX2 exceptions.

The FPSP is not needed for this exception. The user-supplied exception handler is always entered.

3.5.3 Instructions

The following paragraphs describe the arithmetic and transcendental instructions supported by the FPSP.

3.5.3.1 ARITHMETIC. Table 3-7 shows the arithmetic instructions supported by the FPSP.

Table 3-7. Arithmetic Instructions

Name	Description	Name	Description
FADD*	Add	FSUB*	Subtract
FSADD*+	Single-Precision Add	FSSUB*+	Single-Precision Subtract
FDADD*+	Double-Precision Add	FDSUB*+	Double-Precision Subtract
FMUL*	Multiply	FDIV*	Divide
FSMUL*+	Single-Precision Multiply	FSDIV*+	Single-Precision Divide
FDMUL*+	Double-Precision Multiply	FDDIV*+	Double-Precision Divide
FSGLMUL	Single-Precision Multiply	FSGLDIV	Single-Precision Divide
FINT	Integer Part	FINTRZ	Integer Part (Truncated)
FABS*	Absolute Value	FNEG*	Negate
FGETEXP	Get Exponent	FGETMAN	Get Mantissa
FTST*	Test Operand	FCMP*	Compare
FREM	IEEE Remainder	FSCALE	Scale Exponent
FMOVE*	Move FP data register	FSMOVE*	Single-Precision Move
FDMOVE*	Double-Precision Move	FSQRT*	Square Root
FSSQRT*	Single-Precision Square Root	FTWOTOX	2 to the X Power
FMOD	Modulo Remainder	FDSQRT*	Double-Precision Square Root
FDMOD	Double-Precision Modulo Remainder	FSMOD	Single-Precision Modulo Remainder

* The FPSP provides these functions for all decimal data formats, single, double, and extended denormalized data types, and extended unnormalized data types. The MC68040 provides these functions for the remaining formats and types (See page 11 of Reference 2).

\+ Additional functions which are not provided by the MC68881/MC68882.

3.5.3.2 TRANSCENDENTAL. Table 3-8 shows the transcendental instructions supported by the FPSP.

Table 3-8. Transcendental Instructions

Name	Description	Name	Description
FCOS	Cosine	FSIN	Sine
FACOS	Arc Cosine	FASIN	Arc Sine
FCOSH	Hyperbolic Cosine	FSINH	Hyperbolic Sine
FSINCOS	Simultaneous Sine & Cosine	FXTOY*	x to the y Power
FTAN	Tangent	FATAN	Arc Tangent
FTANH	Hyperbolic Tangent	FATANH	Hyperbolic Arc Tan
FLOG2	Log Base 2	FLOG10	Log Base 10
FLOGN	Log Base e	FLOGNP1	Log Base e of (x+1)
FETOX	e to the x Power	FETOXM1	(e to the x Power) -1
FTENTOX	10 to the x Power	FTWOTOX	2 to the x Power

* Additional function not provided by the MC68881/MC68882.

3.6 EXTERNAL INTERFACE REQUIREMENTS

For end users the FPSP is transparent; system Integrators will integrate the FPSP into their system. (**See 3.8.3. Site Adaptations**)

For applications compiled for the MC68881/MC68882 the FPSP provides kernel routines to support the MC68040 unimplemented instructions. The MC68040 uses vector number 11 for the unimplemented instructions. The MC68040 stack frames are different for unimplemented MC68881/MC68882 instructions and other F-line traps. For applications compiled for the MC68040 the unimplemented instructions are contained in a library (to avoid the F_line trap overhead at runtime).

For both applications the FPSP provides kernel routines to support exceptions (vectors 48—54) and unsupported data types (vector 55). The FPSP also provides a system call to make the exceptional operand available to the user.

3.7 PERFORMANCE REQUIREMENTS

The following paragraphs describe the speed, accuracy, and compatibility requirements for the FPSP.

3.7.1 Speed

The performance of the transcendental function routines is equivalent or superior to that of a 33-MHz MC68881/MC68882.

3.7.2 Accuracy

The following paragraphs describe the arithmetic instructions, transcendental instructions, and decimal conversions for the FPSP.

3.7.2.1 ARITHMETIC INSTRUCTIONS. The error bound is one-half unit in the last place of the destination format in the round-to-nearest mode, and one unit in the last place in the other rounding modes.

3.7.2.2 TRANSCENDENTAL INSTRUCTIONS. The error bound is less than 0.6 ulp of double precision.

3.7.2.3 DECIMAL CONVERSIONS. The error bound is 0.97 unit in the last digit of the destination precision for the round-to-nearest mode; and 1.47 units in the last digit of the destination precision for the other rounding modes.

3.7.3 Compatibility

The FPSP transcendental calculation results are not the same as for the MC68881/MC68882. This is because the algorithms used by the MC68881/MC68882 (CORDIC) cannot be effectively implemented in software. All other calculations are identical. The error bound is equivalent or superior to the MC68881/MC68882.

3.8 OTHER REQUIREMENTS

The following paragraphs describe other requirements for the FPSP, such as maintainability, packaging, and site adaptations.

3.8.1 Maintainability

The speed requirements have forced writing most of the package in assembly language.

3.8.2 Packaging

There are two versions of the FPSP. The FPSP Kernel Version is used to execute pre-existing user object code written for the MC68882. This is installed as part of the operating system. User applications need not be recompiled or modified in any way once the FPSP Kernel Version is installed.

The FPSP Library Version is used to compile code that uses only the MC68040-implemented floating point instructions. The library version provides less overhead than the FPSP Kernel Version. Other features of this library includes ABI compliance as well as IEEE exception-reporting compliant. It is not however, UNIX exception-reporting compliant. The FPSP is not available in library format.

3.8.3 Site Adaptations

Some of the entries in the vector table needs to point to entry points within the FPSP Kernel Version. For those vectors the FPSP displaces, an entry point is provided to replace that which it takes. Note that former MC68882 floating-point exception handlers need to go through minor modifications to account for the differences between the MC68040 and MC68882 floating point exceptional state frames. The FPSP provides skeleton code for each floating-point exception handler to aid in porting the MC68882 floating-point exception handlers. The FPSP also provides a system call which extracts and calculates the exceptional operand similar to that found in the MC68882 exceptional state frame.

For systems and applications that never set any of the exception bits in the FPCR, or if the former MC68882 floating-point exception handlers only contain minimum code needed to clear the exception and return, no work is needed and the FPSP is a drop-in package.

The FPSP Library Version needs to "intercept" the appropriate math library calls which use MC68882 transcendental instructions. Since each site has different naming conventions, the FPSP subroutines need to be renamed accordingly and recompiled. The resident compiler also needs to provide a library path search pattern such that the FPSP is given a chance to resolve those trancendentals instructions.

3.8.4 Stack Area Usage

To achieve code re-eentrace, the FPSP allocates context-sensitive variables on the stack. The FPSP does not require more than 512 bytes on the stack per context. This may be an installation concern for UNIX applications in which there is a limited UBLOCK area, and that the system stack resides there.

3.8.5 ROM-based applications

One of the goals of the FPSP Kernel Version is to be able to fit in a read-only space of no more than 64 KBytes. There are two main sections that need to reside in ROM. The text section, and the initialized data section. The text section accounts for 65% while the initialized data section accounts for 35%.

3.9 FPSP KERNEL VERSION INSTALLATION NOTES

Thee following paragraphs provide the MC68882 users with an understanding of the issues involved in porting over the FPSP into existing MC68030/MC68882 systems. Once these issues are understood, then the actual installation is explained.

3.9.1 Differences between the MC68040 and MC68882 Floating-point Exception Handling

The main reason for providing the FPSP is to provide MC68882 compatibility. If the installer understands the main differences between the MC68882 and MC68040 in the area of floating-point exception handlers, skip this section and go to the next section.

There are three areas that differ between the MC68040 and MC68882.

The first difference is that of unimplemented instructions. The FPSP handles this by means of the F-line exception handling. This means that if there is an existing F-line handler, the FPSP replaces the existing F-line exception handler, but provides an alternate entry point for the existing F-line handler.

The second difference is unsupported data types. The MC68040 provides a new entry point in the vector table, therefore no existing handler tare replaced by the FPSP. There are no installation issues here.

The third difference is that of floating point exception differences. This issue is more involved and requires further explanations.

The IEEE standard allows the user to enable or disable each floating point exception individually. If an exceptional condition occurs, the IEEE defines a specific action for the trap-disabled condition, and it also defines certain specific actions for a trap-enabled condition. The IEEE standard however, does not constrain the implementation of exception handling; both software and hardware can be used.

The MC68882 supports the IEEE exception handling compliance totally in hardware. For example, a user-disabled (trap disabled) exception will cause the specified IEEE defined actions for user-disabled exception handling to occur. Similarly, user-enabled exceptions will cause the MC68882 to take the exception as defined by the IEEE trap enabled case.

The MC68040 provides full IEEE trap-disabled exception handling compliance for the INEX and DZ exceptions. Just as the MC68882, the MC68040 takes these exceptions only for an IEEE trap-enabled condition. Existing MC68882 handlers have a minimum code requirement as defined by the MC68882 User's Manual. As the MC68882 handlers, the MC68040 handlers have a minimum code requirement as well. The FPSP provides this minimum code requirement.

The MC68040 does not provide full IEEE exception compliance on IEEE defined trap-disabled conditions for the following exceptions: OVFL, UNFL, OPERR, SNAN. For these exceptions, the MC68040 may take an exception even on an IEEE-defined trap-disabled condition. The FPSP provided exception handlers decide if its job is to implement IEEE trap-disabled exception compliance, (and therefore not execute the user supplied exception handler) or to implement IEEE trap-enabled exception compliance, (hence executing the user supplied exception handler). The FPSP provides a user entry point so that when this entry point is taken an IEEE-defined trap-enabled condition has definitely occurred. At this specified entry point, an exception handler written for the MC68882 needs to be modified to account for MC68040 stack differences, and then placed at the user entry point.

Note that the MC68882 explicitly provides the exceptional operand on its exception state frame, but the MC68040 however, does not. The information needed to calculate the exceptional operand is on the MC68040 exception state frame, and the FPSP provides a system call that performs this calculation.

As with the MC68882, there is a minimum code requirement for the MC68040 handler, but this minimum code is provided by the FPSP.

From an installation perspective, the OVFL, UNFL, OPERR, SNAN exception handlers are replaced by the FPSP handlers, but the FPSP provides an entry point so that MC68882-like exception handlers may be written. Furthermore, minimum code is provided by the FPSP and can be used as a template.

The BSUN exception is different in that unlike the previous exception handlers, the difference between the MC68882 and MC68040 resides in the IEEE-defined trap enabled case. The FPSP handles this by performing the patch needed for MC68882 compatibility, and then restoring the exception to the MC68040 without performing the necessary steps to clear the BSUN exception. The exceptional frame is restored into the MC68040 and the FPSP branches to the user entry point provided. At this entry point, an MC68882-like

exception handler written for the MC68040 is executed without having to worry about the built-in incompatibility. Although this method incurs a performance hit, it frees the user-defined exception handler from having to write the code needed to implement MC68882 code compatibility. As with the other exception handlers, the FPSP provides the minimum code needed.

In summary, the FPSP replaces the following exception handlers and provides an entry-point for MC68882-like exception handlers for these exceptions: OVFL, UNFL, OPERR, SNAN, BSUN, F-line.

The FPSP is not needed for the INEX and DZ exception handlers, and these exception handlers just need to be MC68882-like.

3.9.2 Vector Table

The entry point into the FPSP is achieved by having the appropriate vector table offset point to a specified entry point within the FPSP. For simplicity, all of the FPSP main entry points are found in the file *skeleton.sa* . Table 1 shows the vector table offset and the appropriate labels within the file *skeleton.sa* that it needs to point to. Figure 3-1 shows a flowchart of the entry points.

Table 3-9. FPSP Provided Entry Points

Exception Type	Vector Table offset	FPSP Entry Point	User Entry Point
F-line (unimplemented FP Instruction	vector 11 ($2C)	fline	real_fline
Branch/Set On Unordered	vector 48 ($20)	bsun	real_bsun
Inexact	vector 49 ($C4)	inex	real_inex
Divide-by-zero	vector 50 ($C8)	dz	real_dz
Underflow	vector 51 ($CC)	unfl	real_unfl
Operand Error	vector 52 ($D0)	operr	real_operr
Overflow	vector 53 ($D4)	ovfl	real_ovfl
Signalling Not-A-Number	vector 54 ($D8)	snan	real_snan
Unsupported Data Type	vector 55 ($DC)	unsupp	

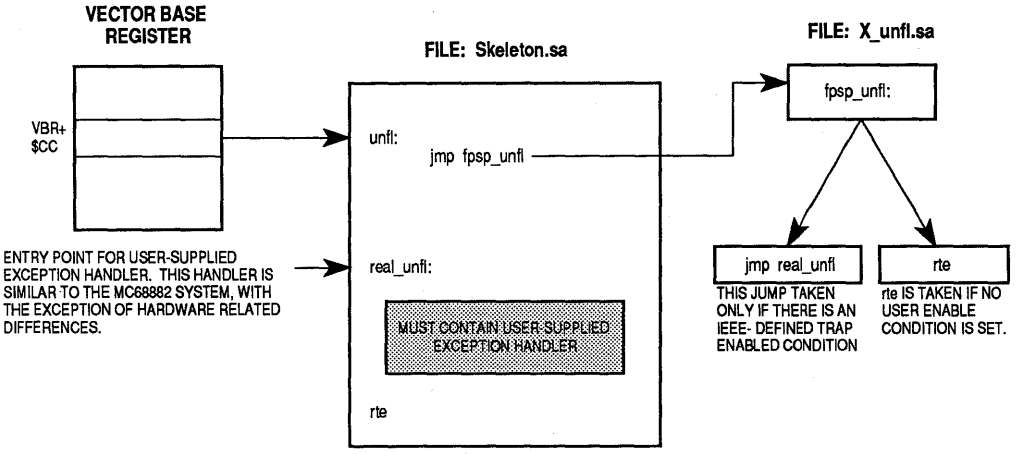


Figure 3-1. FPSP Entry Points

Once the entry point is reached, the user may add some user-specific code prior to jumping to the FPSP routines (FPSP routines are prefixed by "fpsp_"). After the jump to the FPSP routines, the FPSP performs its function and then jumps to the FPSP supplied entry points (if needed) found in the file *skeleton.sa.*

3.9.3 FPSP Supplied Entry Points

To replace the vector table entries it displaces, the FPSP provides an alternate entry point. For simplicity, all of the FPSP supplied entry points are found in the file *skeleton.sa.* The FPSP supplied F-line exception entry point is straight-forward. An F-line exception handler written for an MC68030 can be placed here without modifications. The Unsupported Data Type exception handler is newly-defined, it does not displace any MC68030/MC68882 exception handler. Therefore, the FPSP does not provide an alternate entry point for this exception.

The alternate entry points have the naming convention such that the specified exception handler is prefixed by "real_". For instance, the entry point for user-supplied BSUN exception handler is named "real_bsun".

For the floating-point exception handlers (BSUN, OPERR, SNAN, DZ, OVFL, and UNFL) previously written for an MC68882 based system, these handlers need to be modified slightly for use with the MC68040. Once these handlers are modified, they are then placed in the FPSP provided entry points.

3.9.4 Extract the Hardware Independent portion of the MC68882 handlers

To modify the existing MC68882 handlers, all of the code used in accessing the MC68882 generated frame needs to be stripped off. The code used in clearing an MC68882 exception (setting bit 27 of the BIU Flag) needs to be stripped off as well. Only the hardware independent portions of the MC68882 handlers may be used.

The hardware-independent portion probably requires the "exceptional operand" found in the MC68882 frame. A function "get_xop" is provided in the file *skeleton.sa* which extracts the exceptional operand from the MC68040 stack.

To aid the installer in rewriting the MC68882 exception handlers,The file *skeleton.sa* provides the minimum code necessary to clear the exception once the specific handler is entered.

Once the hardware-independent portion is written, the modified MC68882 handlers need to be integrated into the portion of the code which is hardware dependent. The minimum code needed by each exception handler is already provided by the FPSP within the file *skeleton.sa*. The following section describes the mechanics behind the written code.

3.9.5 MC68040 Minimum Exception Code

This section describes the minimum requirements for the user-supplied exception handlers. As mentioned in the previous sections, these minimum handlers are provided as part of the package, and this section is strictly for the user's information only.

As with the MC68882, if all exceptions are always disabled, no minimum code is necessary since the FPSP guarantees that these FPSP provided entry points are never entered on trap-disabled condition. Therefore, for existing systems that do not provide exception handlers for the MC68882, it is likely that the assumption that all exceptions are always disabled is valid, and therefore no user-defined MC68040 exception handlers are needed either.

The above paragraph should not be interpreted to mean that the FPSP provided exception handlers are unnecessary. On the contrary, the FPSP provided exception handlers are needed, and that these FPSP exception handlers provide the entry points for user-defined exception handlers. Whether or not the user-defined MC68040 exception handers are needed is the issue being discussed.

Assuming that it is possible that the exceptions are enabled at some point, the minimum exception handler is similar to that defined for an MC68882. As with the MC68882, the MC68040 requires that the first floating point instruction be an FSAVE. Unlike the MC68882, the MC68040 does not always require an equivalent FRESTORE. For an E1 exception, only the FSAVE requirement is needed, the state frame may be discarded. The E3 exception is more similar to that found in an MC68882. As with the MC68882, the E3 exception requires an FSAVE, an instruction that clears the exception in the resulting FSAVE stack, followed by an FRESTORE.

If both E3 and E1 exceptions exist at the same time, then the exception is handled as though it were an E3 exception. After which, the MC68040 re-traps to handle the E1 exception..

The E3 exception can only be reported by the following exception handlers: OVFL, UNFL, INEX. For these exception handlers, this is the minimum code requirement:

1) FSAVE
2) if E3 bit set, goto (4), else goto (3)

3) E1 exception, throw away stack and RTE
4) Clear E3 bit, FRESTORE, RTE

The E3 exception cannot be reported by the following exception handlers: SNAN, OPERR and DZ. Since only an E1 exception needs to be handled here, this is the minimum code requirement:

1) FSAVE
2) throw away stack and RTE

For the BSUN exception handler, the minimum code requirement is:

1) FSAVE
2) Do one of 4 methods described in MC68040 User's Manual
3) throw away stack and RTE

If the above minimum code requirements are not met, then, an infinitely looping exception sequence occurs.

3.9.6 Mem_read and Mem_write

The mem_write and mem_read subroutines are used by the FPSP to read and write from user space. These routines perform a UNIX system call to lcopyin and lcopyout. The FPSP provides a simple version of lcopyin and lcopyout for non-UNIX applications. Installation to UNIX-based systems requires that the FPSP provided lcopyin and lcopyout be deleted or commented out. For simplicity, these subroutines are found in the file *skeleton.sa*.

The production version of the FPSP is fully re-entrant. If a page fault occurs on either a mem_read or mem_write, the operating system may perform a page-in operation and still allow other processes to use the FPSP.

3.9.7 Increasing F-line Handler Performance

The FPSP was written to handle all possible cases of MC68040 vs MC68030/MC68882 problem areas. Any performance improvement in this handler increases floating-point performance. The F-line handling may be made quicker by directly pointing the vector table entry directly into the label "fpsp_unimp" found in the file *x_unimp.sa*, if these conditions are met:

1) That the system never has to execute an FMOVECR instruction in which bits 0 to 5 of the F-line word are non-zero.
2) An alternate F-line entry point is unnecessary.

This optimization saves a total of three instructions. (1 bra, 1 cmpi, 1 beq).

3.9.8 Stack Area Usage

To achieve code re-eentrace, the FPSP allocates context-sensitive variables on the stack. The FPSP does not require more than 512 bytes on the stack per context. This may be an installation concern for UNIX applications in which there is a limited UBLOCK area, because the system stack resides there.

3.9.9 ROM-based applications

One of the goals of the FPSP is to be able to fit in a read-only space of no more than 6 KBytes. There are two main sections that need to reside in ROM. The text section, and the initialized data section. The text section accounts for 65% while the initialized data section accounts for 35%.

3.10 REFERENCES

1. MC68881UM/AD, *MC68881/MC68882 Motorola Floating-Point Coprocessor User's Manual.* Motorola Inc., 1989
2. MC68040 Preliminary Design Specification, Revision 6.0, Motorola Inc.,
3. MC68020UM/AD, *MC68020 32-Bit Microprocessor User's Manual,* Motorola, Inc., 1990.
4. MC68030UM/AD, *MC68030 Enhanced 32-Bit Microprocessor User's Manual,* Motorola Inc., 1990
5. ANSI/IEEE Std. 754,1985 Standard for Binary Floating-Point Arithmetic

SECTION 4
MC68040 THERMAL CONSIDERATIONS

As microprocessors are becoming more complex and requiring more power, the need to efficiently cool the device becomes increasingly more important. In the past, the M68000 Family has been able to provide a 0-70°C ambient temperature part for speeds less than 40 MHz. However, the MC68040, which has a 50 MHz arithmetic logic unit (ALU) clock, is specified with a maximum power dissipation for a particular operating mode, a maximum junction temperature, and a thermal resistance from the die junction to the case. This provides a more accurate method of evaluating the environment, taking into consideration both the airflow and ambient temperature available. This also allows a user the information to design a cooling method which meets both thermal performance requirements and constraints of the board environment.

This section discusses the device characteristics for thermal management, several methods of thermal management, and an example of one method of cooling the MC68040.

4.1 MC68040 THERMAL DEVICE CHARACTERISTICS

The MC68040 presents some inherent characteristics which should be considered when evaluating a method of cooling the device. The following paragraphs discuss these die/package and power considerations.

4.1.1 The MC68040 Die and Package

The MC68040 is being placed in a cavity-down alumina-ceramic 179-pin PGA that has a typical thermal resistance from junction to case of 2.7°C/W. This package differs from previous M68000 Family PGA packages which were cavity up. This cavity-down design allows the die to be attached to the top surface of the package, which increases the ability of the part to dissipate heat through the package surface or an attached heat sink. Early MC68040 packages have bypass capacitor mounting pads located on the top which limit the heatsink contact area to approximately 1.48 in. by 1.48 in. to avoid shorting the pads. Later packages have these pads deleted, allowing arbitrary heatsink mounting. Refer to MC68040UM/AD, *MC68040 User's Manual* for more information. The specific dimensions and design of the particular heat sink will need to be determined by the system designer considering both thermal performance requirements and size requirements.

4.1.2 MC68040 Power Considerations

The MC68040 has a maximum power rating, which varies depending on the operating frequency and the output buffer mode combination being used. The large buffer output mode dissipates more power than the small, and the higher frequencies of operation dissipate more power than the lower frequencies. The following paragraphs discuss tradeoffs in using the different output buffer modes, calculation of specific maximum power dissipation for different modes, and the relationship of thermal resistances and temperatures.

4.1.2.1 THE MC68040 OUTPUT BUFFER MODE. The MC68040 is capable of resetting to enable for a combination of either large buffers or small buffers on the outputs of the miscellaneous control signals, data bus, and address bus/transfer attribute pins (refer to MC68040UM/AD, *MC68040 User's Manual* for more information). The large buffers offer quicker output times, which allow for an easier logic design. However, they do so by driving about 11 times as much current as the small buffers (refer to MC68040EC/D, *MC68040 Electrical Specifications* for current output). The designer should consider whether the quicker timings present enough advantage to justify the additional consideration to the individual signal terminations, the die power consumption, and the required cooling for the device. Since the MC68040 can be powered-up in one of eight output buffer modes upon reset, the actual maximum power consumption for a MC68040 rated at a particular maximum operating frequency is dependent upon the power up mode. Therefore, the MC68040 is rated at a maximum power dissipation for either the large buffers or small buffers at a particular frequency (refer to MC68040EC/D, *MC68040 Electrical Specifications*). This allows the possibility of some of the thermal management to be controlled upon reset. The following equation provides a rough method to calculate the maximum power consumption for a chosen output buffer mode:

$$P_D = P_{DSB} + (P_{DLB}-P_{DSB}) \times (PINS_{LB} / PINS_{CLB}) \qquad \text{(Equation 4-1)}$$

where:

P_D = Max. Power Dissipation for Output Buffer Mode Selected
P_{DSB} = Max. Power Dissipation for Small Buffer Mode (all Outputs)
P_{DLB} = Max. Power Dissipation for Large Buffer Mode (all Outputs)
$PINS_{LB}$ = Number of Pins Large Buffer Mode
$PINS_{CLB}$ = Number of Pins Capable of the Large Buffer Mode

Table 4-1 shows the simplified relationship on the maximum power dissipation for eight possible configurations of output buffer modes.

Table 4-1. Maximum Power Dissipation for Output Buffer Mode Configurations

Output Configuration			Maximum Power Dissipation
Data Bus	Address Bus and Transfer Attrib.	Misc. Control Signals	P_D
Small Buffer	Small Buffer	Small Buffer	P_{DSB}
Small Buffer	Small Buffer	Large Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*13\%$
Small Buffer	Large Buffer	Small Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*52\%$
Small Buffer	Large Buffer	Large Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*65\%$
Large Buffer	Small Buffer	Small Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*35\%$
Large Buffer	Small Buffer	Large Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*48\%$
Large Buffer	Large Buffer	Small Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*87\%$
Large Buffer	Large Buffer	Large Buffer	$P_{DSB} + (P_{DLB}-P_{DSB})*100\%$

To calculate the specific power dissipation of a specific design, the termination method of each signal must be considered. For example, a signal output that is not connected would not dissipate any additional power if it were configured in the large buffer rather than the small buffer mode.

4.1.2.2 RELATIONSHIPS BETWEEN THERMAL RESISTANCES AND TEMPERATURES. Since the maximum operating junction temperature has been specified to be 110°C (refer to the MC68040UM/AD, *MC68040 User's Manual.*) The maximum case temperature, T_C , in °C can be obtained from:

$$T_C = T_J - P_D \times \theta_{JC} \qquad \text{(Equation 4-2)}$$

where:

T_C = Maximum Case Temperature
T_J = Maximum Junction Temperature
P_D = Maximum Power Dissipation of the Device
θ_{JC} = Thermal Resistance between the Junction of the Die and the Case

In general, the ambient temperature, T_A , in °C is a function of the following formula:

$$T_A = T_J - P_D \times \theta_{JC} - P_D \times \theta_{CA} \qquad \text{(Equation 4-3)}$$

Where the thermal resistance from case to ambient, θ_{CA}, is the only user-dependent parameter once a buffer output configuration has been determined. As seen from equation (4-3), reducing the case to ambient thermal resistance, increases the maximum operating ambient temperature. Therefore, by utilizing such methods as heat sinks and ambient air cooling to minimize the θ_{CA}, a higher ambient operating temperature and/or a lower junction temperature can be achieved.

However, an easier approach to thermal evaluation uses the following formulas:

$$T_A = T_J - P_D \times \theta_{JA} \qquad \text{(Equation 4-4)}$$

or alternatively,

$$T_J = T_A + P_D \times \theta_{JA} \qquad \text{(Equation 4-5)}$$

where:

θ_{JA} = Thermal resistance from the junction to the ambient ($\theta_{JC} + \theta_{CA}$)

This total thermal resistance of a package, θ_{JA}, is a combination of its two components, θ_{JC} and θ_{CA}. These components represent the barrier to heat flow from the semiconductor junction to the package (case) surface (θ_{JC}) and from the case to the outside ambient (θ_{JC}). Although θ_{JC} is device related and cannot be influenced by the user, θ_{CA} is user dependent. Thus, good thermal management by the user can significantly reduce θ_{CA} achieving either a lower semiconductor junction temperature or a higher ambient operating temperature.

4.2 THERMAL MANAGEMENT TECHNIQUES

To attain a reasonable maximum ambient operating temperature, a user must reduce the barrier to heat flow from the semiconductor junction to the outside ambient (θ_{JA}). The only way to accomplish this is to significantly reduce θ_{CA} by applying such thermal management techniques as heat sinks and ambient air cooling.

The following paragraphs discuss some results of a thermal study of the MC68040 device without using any thermal management techniques; using only air-flow cooling, using only a heat sink, and using heat sink combined with air-flow cooling:

4.2.1 MC68040 Thermal Characteristics in Still Air

A sample size of three MC68040 packages was tested in free-air cooling with no heat sink. Measurements showed that the average θ_{JA} was 22.8°C/W with a standard deviation of 0.44°C/W. The test was performed with 3 W of power being dissipated from within the package. The test determined that θ_{JA} will decrease slightly for the increasing power dissipation range possible. Therefore, since the variance in θ_{JA} within the possible power dissipation range is negligible, it can be assumed for calculation purposes that θ_{JA} is valid at all power levels. Using the formulas introduced previously, Table 4-2 shows the results of a maximum power dissipation of 3 and 5 W with no heat sink or air-flow. (refer to Table 4-1 to calculate other power dissipation values.)

Table 4-2. Thermal Parameters with No Heat Sink or Air Flow

Defined Parameters			Measured	Calculated		
P_D	T_J	θ_{JC}	θ_{JA}	$\theta_{CA} = \theta_{JA} - \theta_{JC}$	$T_C = T_J - P_D * \theta_{JC}$	$T_A = T_J - P_D * \theta_{JA}$
3 Watts	110°C	2.7°C/W	22.8°C/W	20.8 °C/W	101°C	41.6°C
5 Watts	110°C	2.7°C/W	22.8°C/W	20.8 °C/W	96°C	- 4°C

As seen by looking at the ambient temperature results, most users will want to implement some type of thermal management to obtain a more reasonable maximum ambient temperature.

4.2.2 MC68040 Thermal Characteristics in Forced Air

A sample size of three MC68040 packages was tested in forced air cooling in a wind tunnel with no heat sink. This test was performed with 3 W of power being dissipated from within the package. As previously mentioned, since the variance in θ_{JA} within the possible power range is negligible, it can be assumed for calculation purposes that θ_{JA} is constant at all power levels. Using the previous formulas, Table 4-3 shows the results of the maximum power dissipation at 3 and 5 W with air-flow and no heat sink. (Refer to Table 4-1 to calculate other power dissipation values.)

Table 4-3. Thermal Parameters with Forced Air Flow and No Heat Sink

Thermal Mgt. Technique	Defined Parameters			Measured	Calculated		
Air Flow Velocity	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA}	T_C	T_A
100 LFM	3 W	110°C	2.7°C/W	12.7 °C/W	10.7 °C/W	101°C	71.9°C
250 LFM	3 W	110°C	2.7°C/W	11.0 °C/W	9.0°C/W	101°C	77.0°C
500 LFM	3 W	110°C	2.7°C/W	9.9°C/W	7.9 °C/W	101°C	80.3°C
750 LFM	3 W	110°C	2.7°C/W	9.5 °C/W	7.5 °C/W	101°C	81.5°C
1000 LFM	3 W	110°C	2.7°C/W	9.3°C/W	7.3 °C/W	101°C	82.1°C
100 LFM	5 W	110°C	2.7°C/W	12.7 °C/W	10.7 °C/W	96°C	46.5°C
250 LFM	5 W	110°C	2.7°C/W	11.0 °C/W	9.0°C/W	96°C	55.0°C
500 LFM	5W	110°C	2.7°C/W	9.9°C/W	7.9 °C/W	96°C	60.5°C
750 LFM	5 W	110°C	2.7°C/W	9.5 °C/W	7.5 °C/W	96°C	62.5°C
1000 LFM	5 W	110°C	2.7°C/W	9.3°C/W	7.3 °C/W	96°C	63.5°C

By reviewing the maximum ambient operating temperatures, it can be seen that by using the all-small-buffer configuration of the MC68040 with a relatively small amount of air flow (100 LFM), a 0-70°C ambient operating temperature can be achieved. However, depending on the output buffer configuration and available forced-air cooling, additional thermal management techniques may be required.

4.2.3 MC68040 Thermal Characteristics with a Heat Sink

In choosing a heat sink the designer must consider many factors: heat sink size and composition, method of attachment, and choice of a wet or dry connection. The following paragraphs discuss the relationship of these decisions to the thermal performance of the design noticed during experimentation.

The heat sink size is one of the most significant parameters to consider in the selection of a heat sink. Obviously a larger heat sink will provide better cooling. However, it is less obvious that the most benefit of the larger heat sink of the pin fin type used in the experimentation would be at still air conditions. Under forced-air conditions as low as 100 LFM, the difference between the θ_{CA} becomes very small (0.4°C/W or less). This difference continues to decrease as the forced air flow increases. The particular heat sink used in this test fits the package surface area available within the capacitor pads 1.48 in. x 1.48 in. on the early MC68040 and shows a nice compromise between height and thermal performance needs. Later

versions of this package do not include capacitor pads, allowing a larger, lower profile heatsink to beattached.

The heat sink base perimeter area was 1.24"x 1.30" and its height was 0.49". It was a pin-fin-type (i.e. bed of nails) design composed of Al alloy. The heat sink is shown in Figure 4-1 can be obtained through Thermalloy Inc. by referencing part number 2338B.

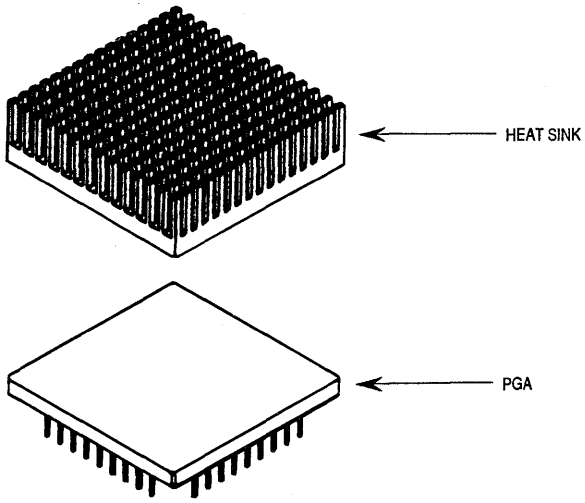


Figure 4-1. Heat Sink Example

All pin fin heat sinks tested were made from extrusion Al products . The planar face of the heat sink mating to the package should have a good degree of planarity; if it has any curvature, the curvature should be convex at the central region of the heat sink surface to provide intimate physical contact to the PGA surface. All heat sinks tested met this criteria. Nonplanar, concave curvature in the central regions of the heat sink will result in poor thermal contact to the package. A specification needs to be determined for the planarity of the surface as part of any heat sink design.

Although there are several ways to attach a heat sink to the package, it was easiest to use a demountable heat sink attach called "E-Z attach for PGA packages" developed by Thermalloy (see Figure 4-2). The heat sink is clamped to the package with the help of a steel spring to a plastic frame (or plastic shoes). Besides the height of the heat sink and plastic frame, no additional height is added to the package. The interface between the ceramic package. and the Al heat sink was evaluated for both dry and wet (i.e., thermal grease) interfaces in still air. The thermal grease reduced the Θ_{CA} quite significantly (about 2.5°C/W) in still air. Therefore, it was used in all other testing done with the heat sink. According to other testing, attachment with thermal grease provided about the same thermal performance as if a thermal epoxy were used.

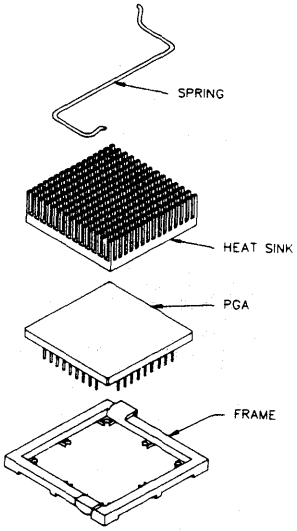


Figure 4-2. Heat Sink with Attachment

A sample size of one MC68040 package was tested in still air with the heat sink and attachment method previously described. This test was performed with 3 W of power being dissipated from within the package. Since the variance in θ_{JA} within the possible power range is negligible, it can be assumed for calculation purposes that $\emptyset_{JA}$ is constant at all power levels. Table 4-4 shows the result assuming a maximum power dissipation of the part at 3 and 5 W (refer to Table 4-1 to calculate other power dissipation values.)

Table 4-4. Thermal Parameters with Heat Sink and No Air Flow

Thermal Mgt. Technique	Defined Parameters			Measured	Calculated		
Heat Sink	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA}	T_C	T_A
2338B	3 W	110°C	2.7°C/W	15.0 °C/W	13.0 °C/W	101°C	65.0°C
2338B	5 W	110°C	2.7°C/W	15.0 °C/W	13.0 °C/W	96°C	35.0°C

4.2.4 MC68040 Thermal Characteristics with a Heat Sink and Forced Air

A sample size of three MC68040 packages was tested in forced-air cooling in a wind tunnel with a heat sink. This test was performed with 3 W of power being dissipated from within the package. As mentioned previously, the variance in θ_{JA} within the possible power range is negligible; it can be assumed for calculation purposes that $\emptyset_{JA}$ is valid at all power levels. Table 4-5 shows the results, assuming a maximum power dissipation at 3 and 5 W with air flow and heat sink

thermal management. (Refer to Table 4-1 to calculate other power dissipation values.)

Table 4-5. Thermal Parameters with Heat Sink and Air Flow

Thermal Mgt. Technique		Defined Parameters			Measured	Calculated		
Air Flow	Heat Sink	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA}	T_C	T_A
100 LFM	2338B	3 W	110°C	2.7°C/W	4.1°C/W	2.1°C/W	101°C	97.7°C
250 LFM	2338B	3 W	110°C	2.7°C/W	3.2°C/W	1.2°C/W	101°C	100.4°C
500 LFM	2338B	3 W	110°C	2.7°C/W	2.7°C/W	0.7°C/W	101°C	101.9°C
750 LFM	2338B	3 W	110°C	2.7°C/W	2.5°C/W	0.5°C/W	101°C	102.5°C
1000 LFM	2338B	3 W	110°C	2.7°C/W	2.4°C/W	0.4°C/W	101°C	102.8°C
100 LFM	2338B	5 W	110°C	2.7°C/W	4.1°C/W	2.1°C/W	96°C	89.5°C
250 LFM	2338B	5 W	110°C	2.7°C/W	3.2°C/W	1.2°C/W	96°C	94.0°C
500 LFM	2338B	5 W	110°C	2.7°C/W	2.7°C/W	0.7°C/W	96°C	96.5°C
750 LFM	2338B	5 W	110°C	2.7°C/W	2.5°C/W	0.5°C/W	96°C	97.5°C
1000 LFM	2338B	5 W	110°C	2.7°C/W	2.4°C/W	0.4°C/W	96°C	98.0°C

4.3 MC68040 THERMAL TESTING SUMMARY

Testing proved that a heat sink in combination with a relatively small amount of air flow (100 LFM or less) will easily realize a 0-70°C ambient operating temperature for the MC68040 with almost any configuration of the output buffers. A heat sink alone may be capable of providing all necessary cooling, depending on the particular heat sink height/size restraints, the maximum ambient operating temperature required, and the output buffer configuration chosen. Also, forced air cooling alone may attain a 0-70°C ambient operating temperature. However, this factor is highly dependent on the output buffer configuration chosen and the available forced air for cooling. Figure 4-3 is a summary of the test results of the relationship between θ_{JA} and air flow for the MC68040.

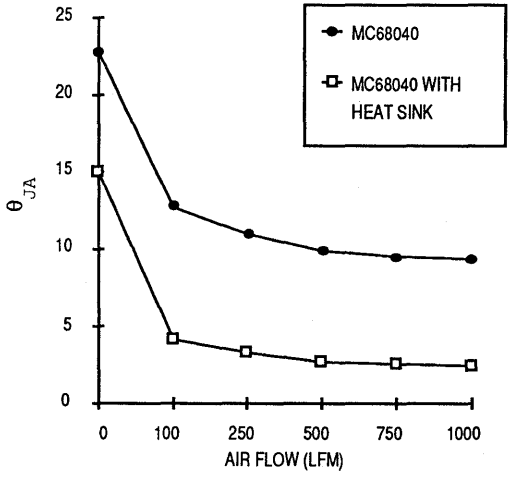


Figure 4-3. MC68040 Relationship of $\emptyset_{JA}$ and Air Flow

SECTION 5
TRANSMISSION LINES

A thorough digital design must transcend the "1" and "0" scope of strictly digital considerations and venture into the traditionally analog world of transmission lines. The effects of transmission lines are evident in all interconnections. However, with the advent of devices possessing extremely fast rise and fall times, these effects are more pronounced. The results of these effects, delays or ringing along an interconnection, can cause unpredictable behavior. To minimize these undesirable events, additional care is required during the design of interconnections and termination. This section serves as the tutorial intended to emphasize the importance of understanding transmission line effects in the application of printed circuit boards (PCBs). It covers the following subjects: PCB traces as transmission lines, transmission line analysis methods, termination schemes, general notes concerning layout, and examples for loaded traces and termination. Figure 5-1 illustrates stair-stepping delays and ringing.

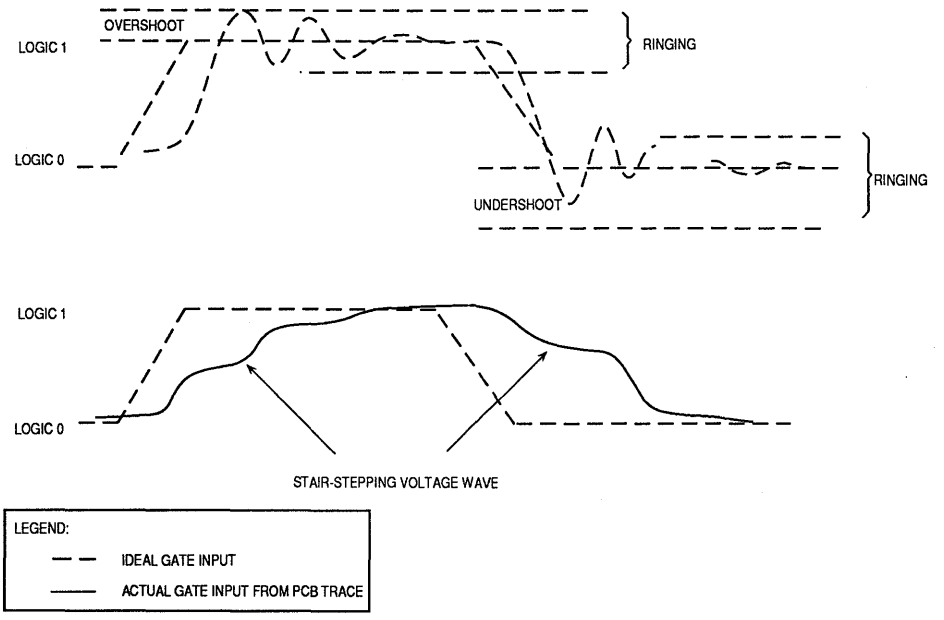


Figure 5-1. Transmission Waveforms

5.1 OVERVIEW

The basic guideline used to determine if a PCB trace needs to be examined for transmission line effects is that if the smaller of the driving device's rise or fall time is less than the twice the time required for a switching wave to propagate through a trace, the transmission line effects are not masked during the rise or fall time of the driving device. The factor of twice the propagation time through a trace is not as arbitrary as it might seem; this factor allows for the switching wave to be transmitted from the driving device, travel through the PCB trace to the receiving device, reflect off the receiving device, and return to the driving device. With this guideline established, the rise and fall times and propagation times of a trace become paramount in transmission line analysis. As devices have been designed to run at higher speeds, their rise and fall times have diminished. Propagation times through a PCB trace depend on the PCB materials and the routing and loading of the trace. Conventional PCB materials have not changed to sustain performance at the faster rise and fall times of driving devices. This situation, as well as the fact that routing and loading can vary greatly from trace to trace, has elevated the importance of investigating PCB traces for transmission line effects.

The lumped capacitive load model shown in many data books is not sufficient for a detailed examination of transmission line effects in PCBs. Two key elements to consider in transmission line analysis are the PCB traces' characteristic impedance, Zo, and its propagation delay, Tpd. The characteristic impedance is the ratio of the voltage to the current in a circuit, so it describes what current and voltage parameters the driving and receiving devices connected to a trace will experience. Mismatches in impedance between segments of the trace and devices connected to the trace cause reflections which results in performance-limiting ringing and delays. The propagation delay is important because it predicts if the effects of these reflections will be hidden during the rise and fall times of a circuit. To account for these elements, more detailed hand-analysis methods, such as the lattice diagram and the Bergeron plot methods, presented in this document, are required. The lattice diagram requires the following information: the rise and fall times of the driving devices, the output impedances of the driving devices, the input impedances of the receiving devices, and the capacitive loading values of the receiving devices. The Bergeron plot method requires the output voltage versus current curves for the driving devices and the input voltage versus current curves and the capacitive loading values of the receiving devices. A complete study also requires information about the PCB, such as the dielectric constant for the insulating materials used, the cross-section trace geometry, and the trace length.

Transmission line effects need to be examined prior to final PCB layout. Not all scenarios can be improved by adding termination. Occasionally, a trace with several segments and loads will have too much delay associated with the reflections and the characteristics of the driving device. Termination may absorb the reflections, but it may overload the driving device, preventing it from operating at its prescribed rate. Hand-analysis helps to highlight potential problem areas and

can provide general rules used in PCB layout once a system designer gains expertise with an output driver type, but, to account for complex routing patterns, computer programs with transmission line and layout emphasis provide more detailed solutions than hand-analysis. The Simulation Program with Integrated Circuit Emphasis (SPICE) has provisions for transmission line analysis, too. It is important that the designer knows basic transmission line theory to evaluate these programs' limitations and features.

Motorola understands the importance and complexity of this issue and is developing a solution that includes a tutorial on basic transmission line theory, a set of enhanced input and output electrical characteristics , and the release of models of the M88000 Family output buffers for use in transmission line programs. Solutions, from hand-analysis methods to results of transmission line programs, are useful in predicting transmission line effects, which need to be correlated with the hardware to verify the basic observations from the paper calculations and simulations. This application note serves as the tutorial intended to emphasize the importance of understanding transmission line effects in the application of PCBs. It covers the following subjects: PCB traces as transmission lines, transmission line analysis methods, termination schemes, general notes concerning layout, and examples for loaded traces and termination. Discussions of termination and examples are presented for exercise only and do not imply behavior of any Motorola devices.

Some techniques promoted in this document have been in practice for over 20 years in ECL designs. The premier source for ECL designs is the *MECL Design Handbook* (see reference 5-1). Since the techniques presented there are designed for ECL applications, they do not automatically work for all CMOS and TTL applications. CMOS and TTL outputs are different from ECL outputs in that ECL has similar output impedances for the low-to-high-driven and the high-to-low-driven cases; whereas, CMOS and TTL output impedances often vary by an order of magnitude from their low-to-high-driven and high-to-low-driven cases. Also, many CMOS and TTL outputs drive five volt swings in less than 2 ns (some actually reach down into the 500-ps range for rise and fall times). ECL outputs drive a 1-V swing in roughly 1 ns. These differences require close examination of the basic, entrenched termination schemes for ECL before they are used with CMOS and TTL.

5.2 PCB TRACES

The following paragraphs discuss the PCB traces as transmission lines, the two trace types, and device loading.

5.2.1 PCB Traces as Transmission Lines

The key factor in determining if a trace acts like a transmission line rather than a lumped load is the relationship between the driving device's rise or fall time and the propagation delay of the signal through the trace. Specifically, the trace should be analyzed as a transmission line, and termination should be considered if

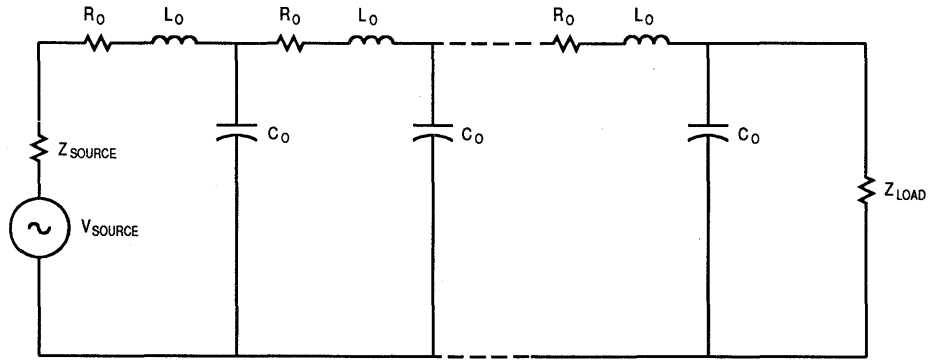
$$2 \times t_{pd} \times \text{trace length} > t_r \text{ or } t_f \text{ (minimum of the two)} \quad (5\text{-}1)$$

t_{pd} is the unit propagation delay of the signal, usually given in nanoseconds per unit length. t_r and t_f are the rise and fall times of the driving device. Note that the fastest of the rise and fall time values should be used. The high-level interpretation of Equation (5-1) is that, if the round-trip time for the switching waveform is greater than the rise or fall time of the driving device, the settling of the transmission line effects are not hidden during the rise and fall times of the driving device.

PCB traces have resistive, inductive, and capacitive effects distributed throughout them. These characteristics are used to develop transmission line models, and the basic means for approximating the distributed effects is a lumped load model. Figure 5-2 shows a transmission line modeled in lumped, constant terms, using the intrinsic resistance, inductance, and capacitance of a trace. This representation is reduced to a transmission line circuit that uses the characteristic impedance (Z_o) and propagation delay (t_{pd}) values to describe the trace. Referring to Figure 5-2, the effects of the intrinsic resistance (R_o) on Z_o are negligible, leaving only the effects of L_o and C_o to be considered in the calculation of the characteristic impedance and propagation delay per unit length:

$$Z_o = \sqrt{L_o/C_o}\ \Omega \quad (5\text{-}2)$$

$$t_{pd} = \sqrt{L_o \times C_o}\ \text{ns/length} \quad (5\text{-}3)$$



NOTE:

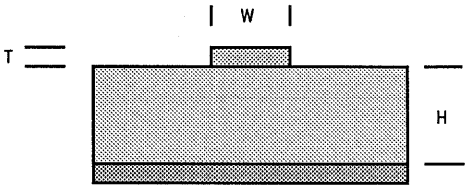
V_{SOURCE} = SWITCHING VOLTAGE SUPPLY
Z_{SOURCE} = OUTPUT IMPEDANCE OF VOLTAGE SUPPLY
R_O = INTRINSIC RESISTANCE OF TRANSMISSION LINE
L_O = INTRINSIC INDUCTANCE OF TRANSMISSION LINE
C_O = INTRINSIC CAPACITANCE OF TRANSMISSION LINE
Z_{LOAD} = LOAD IMPEDANCE

Figure 5-2. Lumped Transmission Line Approximation

If the intrinsic capacitance and inductance are known, they can be used in the previous equations. Usually, the propagation delay and the characteristic impedance are calculated from cross-section geometries and dielectric materials used in the PCB traces. The equations using the geometries and materials are based on Equations (5-2) and (5-3) but then require considerations that treat the traces as if they are operating in the transverse electromagnetic mode. If more information is desired for that derivation, see Reference 5-1. For this discussion, only multilayer board types (microstrip and stripline) are analyzed. For wire-wrapped applications, consult Reference 5-1.

5.2.2 PCB Trace Types

The two basic types of traces are the microstrip and the stripline. Each type can be modified to form derivatives such as the embedded microstrip, the dual symmetrical stripline, and the dual asymmetrical stripline configurations. Cross-section diagrams of these trace types are shown in Figure 5-3.



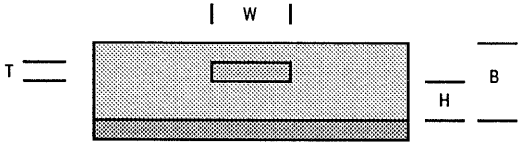
$$Z_O = \frac{87}{\sqrt{E_R + 1.41}} \ln\left\{\frac{5.98\ H}{0.8\ W + T}\right\}$$

WHERE: $0.1 \le W/T \le 3.0$, $1 \le E_R \le 15$

E_R = DIELECTRIC CONSTANT
H = DIELECTRIC THICKNESS
T = TRACE THICKNESS
W = TRACE WIDTH

$$t_{PD} = 1.017\ \sqrt{0.475\ E_R + 0.67}\ \text{ns/ft}$$

(a) Surface Microstrip



$$Z_O = \frac{K}{\sqrt{0.805\ E_R + 2}} \ln\left\{\frac{5.98\ H}{0.8\ W + T}\right\}$$

WHERE: $60 \le K \le 65$

E_R = DIELECTRIC CONSTANT
H = DIELECTRIC THICKNESS BETWEEN TRACE AND POWER/GROUND PLANE
B = OVERALL DIELECTRIC THICKNESS
T = TRACE THICKNESS
W = TRACE WIDTH

$$t_{PD} = 1.017\ \sqrt{0.475\ E_R + 0.67}\ \text{ns/ft}$$

OR:

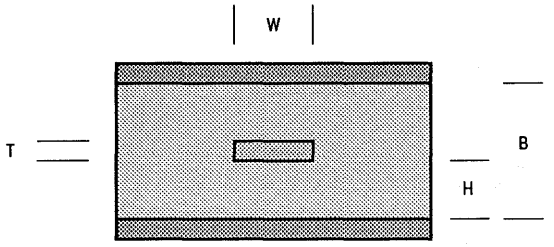
$$Z_O = \frac{87}{\sqrt{E_R' + 1.41}} \ln\left\{\frac{5.98\ H}{0.8\ W + T}\right\}$$

WHERE: $E_R' = E_R \left\{1 - e^{\left(\frac{-1.55\ B}{H}\right)}\right\}$

$$t_{PD} = 1.017\ \sqrt{0.475\ E_R' + 0.67}\ \text{ns/ft}$$

(b) Embedded Microstrip

Figure 5-3. Cross-section Trace Diagrams (1 of 3)



$$Z_O = \frac{60}{\sqrt{E_R}} \ln\left\{\frac{4B}{0.67\,p\,W\left(0.8 + \frac{T}{W}\right)}\right\}$$

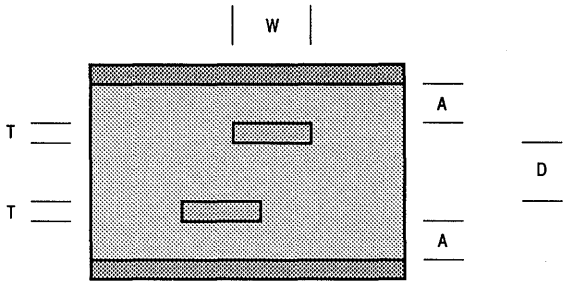
$$t_{PD} = 1.017 \times \sqrt{E_R} \quad \text{ns/ft}$$

WHERE:
E_R = DIELECTRIC CONSTANT
H = DIELECTRIC THICKNESS BETWEEN TRACE AND POWER/GROUND PLANE
B = OVERALL DIELECTRIC THICKNESS
T = TRACE THICKNESS
W = TRACE WIDTH

VALID FOR $\frac{W}{(B-T)} < 0.35$ AND $\frac{T}{B} < 0.25$

(c) Stripline

Figure 5-3. Cross-section Trace Diagrams (2 of 3)



$$Z_O = \frac{2YZ}{Y+Z} \qquad \text{WHERE } Y = \frac{60}{\sqrt{E_R}} \ln\left\{\frac{8A}{0.67\,p\,W\left(0.8+\frac{T}{W}\right)}\right\}$$

$$\text{WHERE } Z = \frac{60}{\sqrt{E_R}} \ln\left\{\frac{8\,(A+D)}{0.67\,p\,W\left(0.8+\frac{T}{W}\right)}\right\}$$

$$t_{PD} = 1.017 \times \sqrt{E_R} \text{ ns/ft}$$

WHERE:
E_R = DIELECTRIC CONSTANT
A = DIELECTRIC THICKNESS BETWEEN TRACE AND POWER/GROUND PLANE
D = DIELECTRIC THICKNESS BETWEEN TRACES
T = TRACE THICKNESS
W = TRACE WIDTH

OR:

$$Z_O = \frac{80\left[1-\left(\frac{A}{4\,(A+D+T)}\right)\right]}{\sqrt{E_R}} \ln\left\{\frac{1.9\,(2A+T)}{0.8W+T}\right\}$$

$$t_{PD} = 1.017 \times \sqrt{E_R} \text{ ns/ft}$$

WHERE:
E_R = DIELECTRIC CONSTANT
A = DIELECTRIC THICKNESS BETWEEN TRACE AND POWER/GROUND PLANE
D = DIELECTRIC THICKNESS BETWEEN TRACES
T = TRACE THICKNESS
W = TRACE WIDTH

LEGEND:

DIELECTRIC

TRACE

GROUND OR POWER PLANE

(d) Dual Stripline

Figure 5-3. Cross-section Trace Diagrams (3 of 3)

The most frequently used types of dielectrics are glass-epoxy (G-10) and one of its derivatives (FR-4). The dielectric constants for G-10 can be characterized generally as 4.5 to 5.0, and the dielectric constant for FR-4 can be characterized as 4.5 to 5.2, although PCB manufacturers have seen values as low as 4.0 for both

constants. The dielectric constant varies with frequency; Reference 5-2 provides a discussion on this. Trace dimensions can vary greatly depending on their applications. For multilayer boards with overall board thicknesses of 0.062 inches, dielectric thicknesses of layers range from 0.005-0.016. Controlled impedance boards, in which all traces on the board have characteristic impedances within a specified range of several Ω, usually have the controlled characteristic impedance in the range of 55-75 Ω due to constraints on manufacturing PCB traces, such as maximum dielectric thicknesses and minimum trace widths.

5.2.3 Device Loading

When a trace is loaded with devices, inductance and capacitance from the devices add to the trace's inductance and capacitance. Figure 5-4 illustrates the loading down the line. This loading alters propagation delay and characteristic impedance values as shown in Equations (5-4) and (5-5):

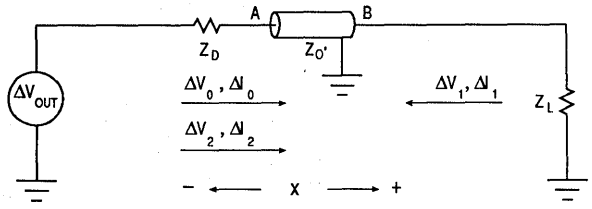
$$t_{pd}' = t_{pd} \times \sqrt{1+(C_d/C_o)} \text{ ns/length} \quad (5\text{-}4)$$

$$Z_o' = Z_o / \sqrt{1+(C_d/C_o)}\ \Omega \quad (5\text{-}5)$$

where C_d is the distributed capacitance of the receiving devices (i.e. total load capacitance/trace length) and C_o is intrinsic capacitance of the trace. Sockets and vias also add to the distributed capacitance (sockets ≈ 2 pF and vias ≈ 0.3-0.8 pF). Since $t_{pd} = \sqrt{L_o x C_o}$ and $Z_o = \sqrt{L_o/C_o}$, C_o can be calculated as follows:

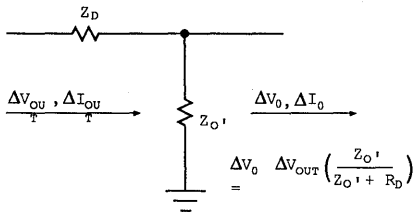
$$C_o = 1000 \times (t_{pd} / Z_o) \text{ pF/length} \quad (5\text{-}6)$$

This loaded propagation delay value must be used when deciding whether a trace should be considered a transmission line ($2 \times t_{pd}' \times$ trace length $> t_r$ or t_f).

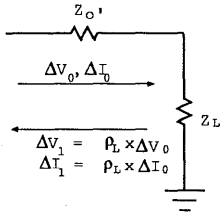


ΔV_{OUT} = VOLTAGE SOURCE, V_{OH} AND V_{OL}
Z_D = OUTPUT IMPEDANCE OF DRIVING DEVICE
Z_L = LOAD IMPEDANCE
ΔV_0 = INCIDENT VOLTAGE
ΔI_0 = INCIDENT CURRENT
A = DRIVING END OF TRACE
B = RECEIVING END OF TRACE
Z_O' = LOADED CHARACTERISTIC IMPEDANCE
ΔV_1 = REFLECTED VOLTAGE AT LOAD
ΔV_2 = REFLECTED VOLTAGE AT DRIVING DEVICE
ΔI_1 = REFLECTED CURRENT AT LOAD
ΔI_2 = REFLECTED CURRENT AT DRIVING DEVICE

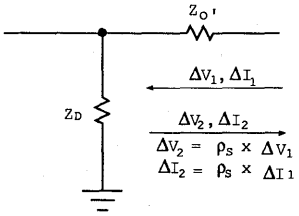
(a) Transmission Line Representation with Z_O' and Load Resistance



(b) Voltage Divider and Incident Wave at Driving Device



(c) Incident Wave and Reflected Wave at Receiving Device



(d) Reflected Wave at Source

NOTE: These are simplified, intermediate representations of the transmission line model.

Figure 5-4. Simplified Transmission Line Representation

The results of these equations are not as straightforward as they seem. C_d, the distributed capacitance per length of the trace, depends on the capacitive load of the receiving devices, sockets, and vias. To mask transmission line effects, slower rise and fall times are recommended. A heavily loaded trace slows the rise and fall times of the devices due to the increased RC time constant associated with the increased distributed capacitance and the filtering of high-frequency components from the switching signal.. Heavily loaded traces seem advantageous until the loaded trace condition is examined. A high C_d raises the loaded propagation delay and lowers the loaded characteristic impedance. The higher loaded propagation delay value increases the likelihood that transmission line effects will not be masked during the rise and fall times, and a lower loaded characteristic impedance often exaggerates impedance mismatches between the driving device and the PCB trace. Thus, the apparent benefits of a heavily loaded trace are not realized unless the driving device is designed to drive large capacitive loads.

To reduce the effects of the distributed capacitance (C_d), a higher intrinsic capacitance (C_o) is used. A higher C_o correlates to a higher unloaded propagation delay. The higher C_o counteracts the higher C_d values, but often the unloaded propagation delay is too high for transmission line effects to be avoided. Examples 1 and 2 (**5.8 LOADED MICROSTRIP** and **5.9 LOADED STRIPLINE** illustrate these considerations. Calculations are performed on microstrip and stripline PCB configurations to demonstrate how loading affects each configuration. The microstrip has lower unloaded propagation delay and intrinsic capacitance values than the stripline. These examples show that the microstrip may be faster than the stripline, but it is affected by loading more than the stripline, which is the consequence of having a lower intrinsic capacitance. In each of these configurations, the best way to limit transmission line effects is to keep PCB traces as short as possible, which lowers the round-trip time of the signal, increasing the likelihood that transmission line effects will be masked during the rise and fall times of the signal.

Loading also alters the characteristic impedance of the trace. As with the loaded propagation delay, a high ratio between the distributed capacitance and the intrinsic capacitance exaggerates the effects of loading on the characteristic impedance. Because the loading factor divides into the characteristic impedance, the characteristic impedance is lowered when the trace is loaded. Reflections on a loaded PCB trace, which cause ringing and stair-stepping delays, have greater amplitudes when the loaded characteristic impedance differs substantially from the driving device's output impedance and the receiving device's input impedance. A complete discussion of these reflections is presented in **5.3.1 Lattice Diagram**.

5.3. TRANSMISSION LINE ANALYSIS TECHNIQUES

After the loaded characteristic impedance and the loaded propagation delay values of the PCB traces are calculated, the interaction between the devices on the trace and the trace is ready to be defined. The two basic occurrences are ringing

and stair-stepping delays. Ringing causes unwanted crossings of logic thresholds, which can be detected by receiving devices with inputs that recognize very fast switching events. In cases with severe ringing, undershoot is produced, which can cripple receiving devices by violating the minimum voltage level allowed at their inputs. Delays caused by stair-stepped voltages limit the overall switching speed of an interconnection. Transmission line analysis methods, from hand-analysis methods to sophisticated computer programs, must be able to predict this behavior.

Lattice diagram and Bergeron plot methods are recommended for hand analysis of transmission line effects. The lattice diagram method requires several calculations to derive reflection coefficients, which then are used to determine the amplitude of reflections at points of unequal impedance throughout a transmission line. References 5-1 and 5-3 (see **5.18 REFERENCES**) provide a complete derivation of this method. The Bergeron plot is a graphic approach requiring little calculation. It uses the current versus voltage curves of the devices tied to the trace as well as the loaded characteristic impedance of the trace to determine voltages at the driving and receiving devices. References 5-4, 5-5, and 5-6 (see **5.18 REFERENCES**) give basic descriptions of this method. These methods provide information necessary to compile voltage versus time plots, which show the severity and occurrence of transmission line effects.

5.3.1 Lattice Diagram

The following explanation of the lattice diagram method is derived from the finite-length transmission line with a resistive load representation (see Figure 5-4(a)). The first event in this derivation is that a voltage wave (ΔV_{out}) is sent down the line in the positive-X direction (toward the load). At the output of the driving device, the voltage is split in proportion to the ratio of the output impedance of the driving device and the loaded characteristic impedance (see Figure 5-4(b)). A new voltage (ΔV_0), called the incident voltage, now travels down the transmission line.

Throughout the transmission line portion of this circuit, the voltage-to-current ratio equals the loaded characteristic impedance ($\Delta V_0/\Delta I_0 = Z_o'$). At the load, the voltage-to-current ratio equals the load impedance ($V_L/I_L = Z_L$). At the junction of the transmission line and the load, to satisfy Kirchoff's circuit equations, a new reflected voltage wave traveling in the negative-X direction (toward the driving device) is introduced; the voltage-to-current ratio as it travels down the transmission line equals the loaded characteristic impedance ($\Delta V_1/\Delta I_1 = Z_o'$). The incident voltages and currents add with the reflected voltages and currents to equal the voltage and current at the load ($V_L = \Delta V_1 + \Delta V_0$ and $I_L = \Delta I_1 + \Delta I_0$). This concept is depicted in Figure 5-4(c).

The identical situation of accounting for Kirchoff's circuit equations by adding new, reflected voltages and currents occurs at the driving device's output. The voltage and current equations at the driving device's output are derived identically as those at the load, which require that a new voltage and current (ΔV_2 and ΔI_2) be introduced, using the Kirchoff circuit equations to derive voltage and current at the

driving device ($V_d/I_d = Z_d$, $V_d = \Delta V_1 + \Delta V_2$, and $I_d = \Delta I_1 + \Delta I_2$). Figure 5-4(d) is a representation of this concept.

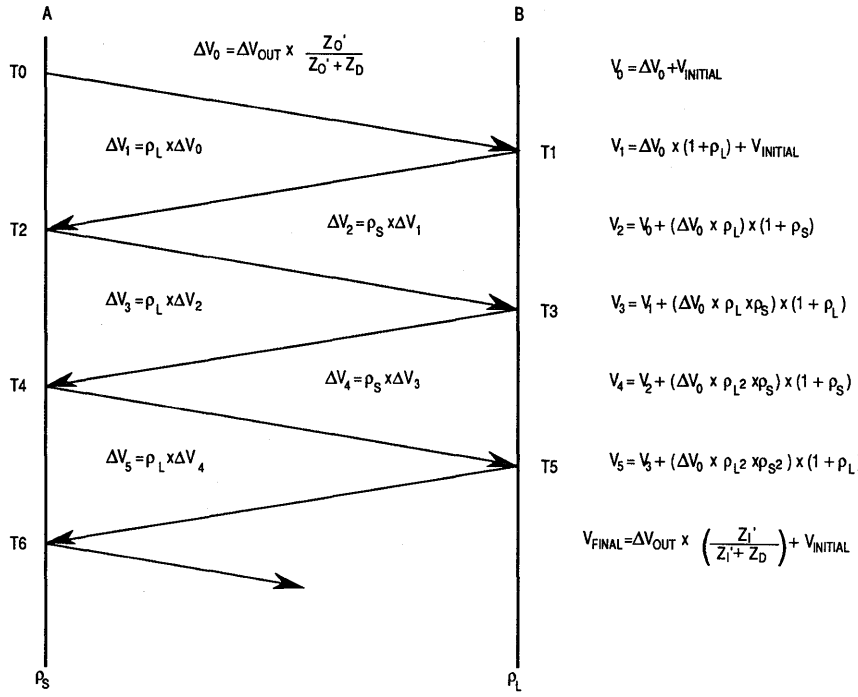
The ratio of the reflected voltage to that of the incident voltage at the load is defined as the load reflection coefficient, ρ_L; the ratio of the reflected voltage and its subsequent reflected voltage at the source is defined as the source reflection coefficient, ρ_S. They are described as follows:

$$\rho_L = \frac{Z_L - Z_O'}{Z_L + Z_O'} \tag{5-7}$$

$$\rho_S = \frac{Z_d - Z_O'}{Z_d + Z_O'} \tag{5-8}$$

where Z_L is the impedance at the receiving device's input, Z_d is the impedance at the driving device's output, and Z_O' is the loaded characteristic impedance.

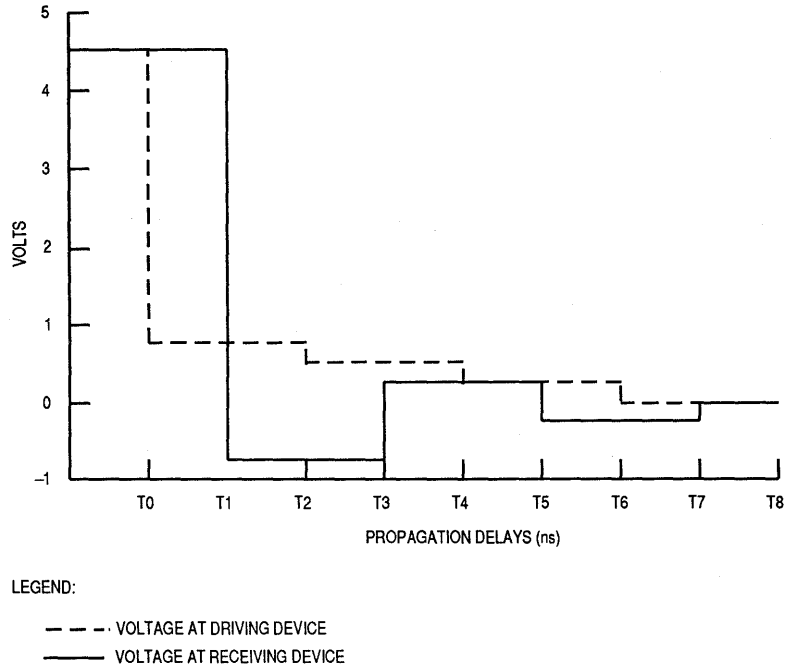
Once computed, the reflection coefficients are used to determine the voltage and current reflected at each discontinuity by multiplying the incoming voltage and current by the reflection coefficients. The voltages and currents traveling into and out of each discontinuity sum to equal the voltages and currents at the discontinuity. If Z_L or $Z_d > Z_O'$, then the reflected wave is positive; if Z_L or $Z_d < Z_O'$, then the reflected wave is negative. See Figure 5-5(a) for this representation of the lattice diagram.



NOTE: ρ_L and ρ_S are reflection coefficients and $V_{INITIAL}$ is the steady-state voltage prior to the switching of the gate. Each TD ÓØ" Ì propagation delay (T_{PD}') in duration. A and B indicate the driving and receiving ends of the trace, respectively

(a) Lattice Diagram

Figure 5-5. Lattice Diagram (Sheet 1 of 2)



(b) Voltage Versus Time Plot

Figure 5-5. Lattice Diagram (Sheet 2 of 2)

The two basic types of transmission line effects are ringing and stair-stepped switching events. Since CMOS and TTL devices have large input impedances, their load reflection coefficients are close to 1, indicating that the full incident wave from the driving device is reflected back to the driving device. The ringing or stair-stepping events are dependent on the values of the driving device's output impedance and the trace's loaded characteristic impedance. Specifically, when $\rho_L \approx 1$, ringing is caused when the loaded characteristic impedance is greater than the driving device's output impedance, resulting in a negative source reflection coefficient and a large initial voltage step; stair-stepping is present when the loaded characteristic impedance is less than the driving device's output impedance, resulting in a positive source reflection coefficient and a small initial voltage step. Some CMOS or TTL devices have grossly different output impedances for the low-to-high-driven and high-to-low-driven states, which produce ringing in high-to-low switching and stair-stepping delays in low-to-high switching. In these cases, termination may reduce ringing while increasing the stair-stepping, requiring the designer to choose which effect is worse and to terminate appropriately. Examples 3-5 illustrate the effects of loading and termination for transmission line effects.

This derivation of the lattice diagram method illustrates several details concerning transmission line effects. If the load impedance matches the loaded characteristic impedance, there are no reflections after the incident wave arrives at the load, which is the goal of most termination schemes. Also, this derivation uses a single transmission line with discontinuities at the load and driving device only. Often, a loaded trace connected to several devices is approximated in this manner rather than using a separate discontinuity for each device input. This approximated transmission line representation is accurate if the stubs off the main trace are very short (< 1 in) or if the trace is laid out using a daisy-chain format. If the designer is not afraid of a lattice diagram that looks like random molecules bouncing off various surfaces, then the nonapproximate method is straightforward, because the derivation for reflection coefficients at each discontinuity is identical to the derivation of the reflection coefficients at the driving device and receiving device. An example of this multiple discontinuity situation is shown on page 222 of Reference 1.

The detailed mathematical representation of the lattice diagram is described as follows:

$$V(t) = \Delta V_0(t) \times \{ U(t-t_{pd}) + \rho_L \times U[t-t_{pd}(2l-X)] + \rho_L \times \rho_S \times U[t-t_{pd}(2l+X)] + \rho_L^2 \times \rho_S \times U[t-t_{pd}(4l-X)] + \rho_L^2 \times \rho_S^2 \times U[t-t_{pd}(4l+X)] + ...\} + V_{initial} \quad (5\text{-}9)$$

where:

$\Delta V_0(t)$ = voltage at driving device from the voltage divider equation
$\Delta V_0(t) = \Delta V_{out} \times [Z_o' / (Z_o' + Z_d)]$

where ΔV_{out} is the voltage output swing of the driving device, Z_d is the output impedance of the driving device, and Z_o' is the loaded characteristic impedance of the PCB trace.

t_{pd} = propagation delay of the line
$U(t)$ = unit step function
l = length of trace
X = any point along the trace
ρ_L = load reflection coefficient
ρ_S = source reflection coefficient
$V_{initial}$ = quiescent voltage throughout the trace prior to switching

The derivation of this representation is found on page 123 of Reference 5-1. The simplified representation, complete with equations, is given in Figure 5-5(a), which shows how to compute the voltages at the driving and receiving ends of a transmission line using reflection coefficients. The vertical line on the left represents the starting position (X = 0) of the incident wave (driving device); whereas, the vertical line on the right represents the end position (X = l) of the incident wave and the point of the first reflected wave (receiving device). The voltages and currents at each endpoint are equal to the voltages and currents flowing into and out them. The time required to travel the length of the transmission

line (PCB trace) is equal to t_{pd}', which are the units of the vertical scale. Units of length are used for the horizontal scale.

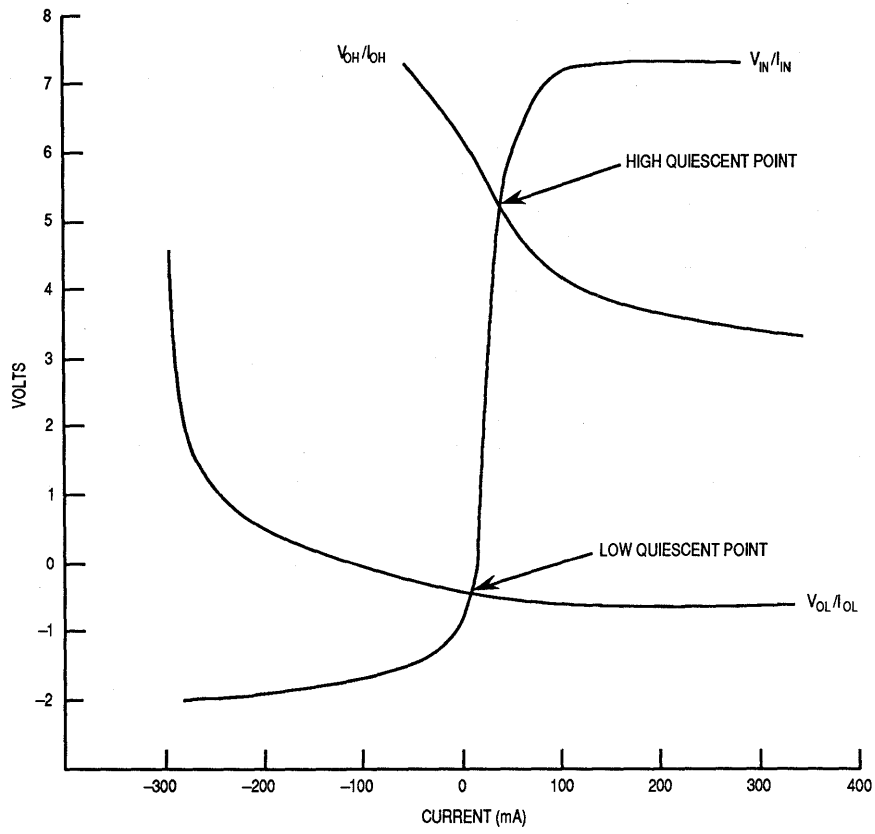
The lattice diagram method does make an important assumption in that unless the Z_d and Z_L are specifically given by a data sheet, they have to be calculated over the particular voltage of interest. Not all devices behave linearly over the range of interest. For instance, if a high-to-low transition is to be analyzed, V_{ol}/I_{ol} is used to calculate Z_d for the driving device; for a low-to-high transition, V_{oh}/I_{oh} is used. Frequently, these voltage and current values are given as test conditions or worst-case numbers in a device's specification and do not represent the typical or best-case performance of the device. It is also difficult to find input voltage and current curves, but they are usually approximated by using Schottky diode clamps for CMOS and TTL devices. If the device does not respond linearly and the V_{out}/I_{out} curves for the driving device and the V_{in}/I_{in} curves for the receiving device are available, then the Bergeron plot method is advised.

The lattice diagram is translated into a voltage versus time plot to show the response at each end of the transmission line (see Figure 5-5(b)). The time required for each voltage wave to travel through the trace is t_{pd}', therefore each voltage level is plotted versus t_{pd}' directly from the lattice diagram. In general, the voltage changes at the driving device at time $2n \times t_{pd}'$; the voltage changes at the receiving device at time $(2n + 1) \times t_{pd}'$.

5.3.2 Bergeron Plot

This method does not have the legacy of applications that the lattice diagram method has, but it provides the same basic information with few calculations. It relies on a graphic means of describing the reflections on the trace.

The current versus voltage curves for the devices connected to the trace and the loaded characteristic impedance of the trace itself are required for this method. The axes are the voltage and current ranges for the driving device and receiving device. The curves plotted on the graph are the V_{oh}/I_{oh} and V_{ol}/I_{ol} of the driving device and the V_{in}/I_{in} of the receiving device. See Figure 5-6 for a basic representation.



NOTE: V_{OL}/I_{OL} and V_{OH}/I_{OH} are output curves for the driving device; V_{IN}/I_{IN} is the input curve for the receiving device.

Figure 5-6. Bergeron Plot

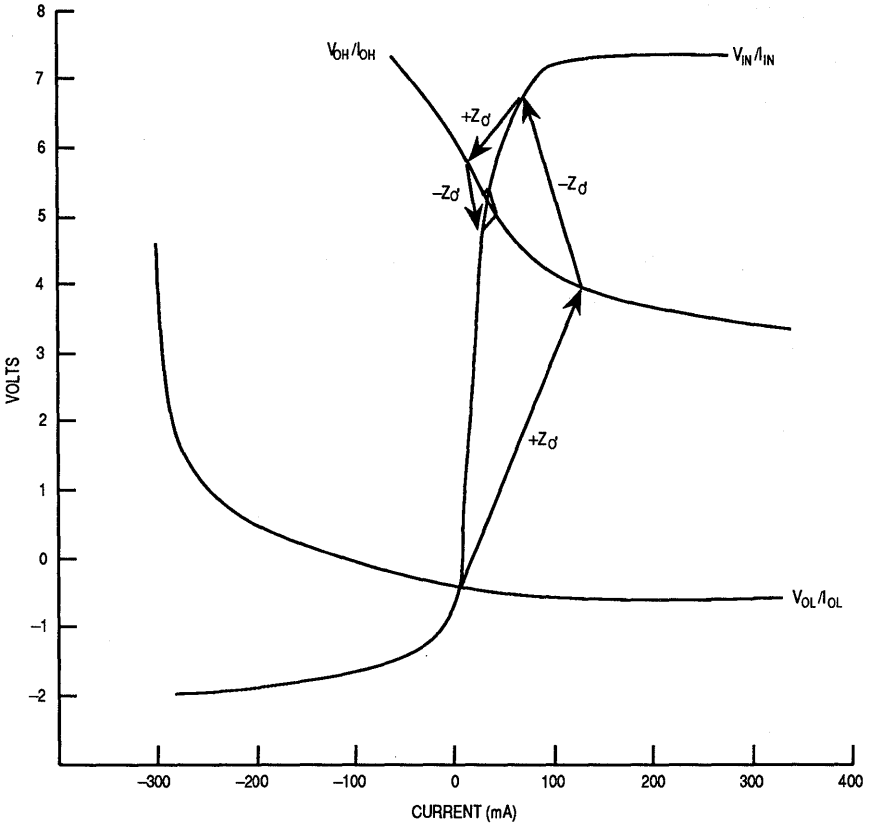
The starting point for a transition depends on the quiescent values of the circuit, which are determined by the intersections of the output curves of the driving device with the input curves of the receiving device. These points are the voltage and current values to which the circuit settles in a stable high or low state prior to the transition. In tracking either a high-to-low or a low-to-high transition, a line of slope $\pm Z_o'$ is drawn between the curves. The means for calculating Z_o' is presented in Equation (5-5).

For the high-to-low transition, the first line is drawn starting at the logic-high quiescent point (intersection of V_{oh}/I_{oh} and V_{in}/I_{in}) and has a slope of $-Z_o'$ (see Figure 5-7). The line ends at the V_{ol}/I_{ol} line, with the intersection indicating the voltage and current values at the output of the driving device and the input of the transmission line.

The second line drawn has a $+Z_o'$ slope from this second intersection point to the V_{in}/I_{in} line. This point indicates the current and voltage at the input of the receiving device and the end of the transmission line after one propagation delay since the signal has traveled one trace length.

The third line is drawn from this point to the V_{ol}/I_{ol} line at a slope of $-Z_o'$. This point is the voltage and current at the driving device after two propagation delays, since the signal is reflected back to the driving device from the receiving device. The fourth line on the plot has a slope of $+Z_o'$ and ends at the V_{in}/I_{in}. This procedure continues until the quiescent point for the logic low state is reached.

The low-to-high transition is handled similarly. The starting point is at the logic-low quiescent point. The first line has a $+Z_o'$ slope drawn to the V_{oh}/I_{oh} line. The second line has a $-Z_o'$ slope drawn from the intersection to the V_{in}/I_{in} line. Identical to the high-to-low case, intersection points that lie on a V_{oh}/I_{oh} or V_{ol}/I_{ol} line are values at the driving device; those on the V_{in}/I_{in} line are values at the receiving device. Figure 5-7 illustrates the low-to-high switching.



NOTE: V_{OL}/I_{OL} and V_{OH}/I_{OH} are output curves for the driving device; V_{IN}/I_{IN} is the input curve for the receiving device; and Z_O' is the loaded characteristic impedance of the trace.

Figure 5-7. Low-to-High Switching

The Bergeron plot results are easily transferred to a voltage versus time plot. The first intersection in the plot gives voltage at time t_0, which is the instant that the driving device switches. The second intersection marks the voltage and current at the receiving device, which occurs at time t_1 (t_{pd}' after t_0). The next change at the driving device occurs at $2t_{pd}'$. In general, the voltage switches at the driving device at time $2n \times t_{pd}'$, whereas the voltage switches at the receiving device at time $(2n + 1) \times t_{pd}'$. The two graphs can be superimposed if desired. See Figure 5-5(b) from the lattice diagram illustration.

Basic Bergeron plot exercises are shown in Examples 6 and 7. Examples 8-10 (see **5.15 BERGERON PLOT WITH PARALLEL TERMINATION**) step through series and parallel termination results for various values.

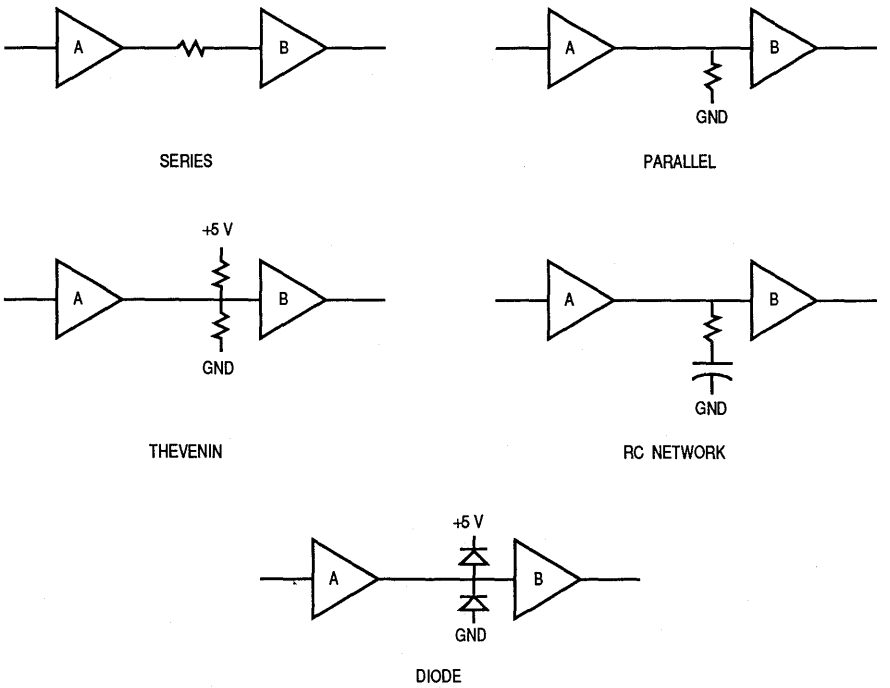
5.4 TERMINATIONS

To dissipate the undesired effects of unmatched traces and loads, termination of traces is utilized. No standard termination works universally due to the complexities of layout geometries, power considerations, component count, and other factors that are discussed in the following paragraphs. In some cases, a combination of these schemes works best.

Five of the most frequently used terminations are as follows:

1. Series Termination Resistor
2. Parallel Termination Resistor
3. Thevenin Network
4. RC Network
5. Diode Network

Figure 5-8 and Table 5-1 provide a synopsis of these implementations.



NOTES:
1. A- driving device; B - receiving device.
2. Termination near A should be at the driving device's output, and those near B should be the receiving device's input.

Figure 5-8. Trace Terminations

Table 5-1. Termination Types and Their Properties

Termination Type	Added Parts	Delay Added	Power Requirements	Parts' Values	Comments
Series	1	Yes	Low	$R_S = Z_o' - Z_d$	Good dc Noise Margin
Parallel	1	Small	High	$R = Z_o'$	Power Consumption Is a Problem
Thevenin	2	Small	High	$R = 2 \times Z_o'$	High Power for CMOS
RC Network	2	Small	Medim	$R = Z_o'$ $C = 300pF$	Check Bandwidth and Added Capacitance
Diode	2	Small	Low	—	Limits Overshoot; Some Ringing at Diodes

5.4.1 Series Termination Resistor

The series termination resistor is preferred when the load is lumped at the end of the trace, when the driving device's output impedance (Z_d) is less than the loaded characteristic impedance of the trace (Z_o'), or when a minimum number of components is required. It is placed near the driving device, and its value is equal to ($Z_o' - Z_d$). When the series resistance and output impedance equal Z_o', as prescribed, the voltage wave is split evenly, causing half of the voltage to be transmitted to the receiving device. A receiving device with a very high input impedance sees the full waveform immediately (due to the reflection at its end of the trace), but the driving device does not see the full waveform until 2 x t_{pd}' (see **5.11 LATTICE DIAGRAM WITH SERIES TERMINATION—EXAMPLE 4**). Because devices often have different output impedance values for their high and low cases, choosing a series resistance value is not always straightforward.

5.4.2 Parallel Termination Resistor

The parallel termination resistor utilizes a single resistor whose value equals Z_o' tied to ground or a positive voltage, usually V_{CC} or +3 V. It adds some delay, as the RC time-constant effects of the trace are increased when the termination resistance is included. Its major disadvantage is that it dissipates much dc power, since its value is small (50-150 Ω). Examples 5 and 8 illustrate these considerations concerning parallel termination.

5.4.3 Thevenin Network

The Thevenin network connects one resistor to ground and a second resistor to V_{CC}, whose Thevenin equivalent approximately equals Z_o'. To avoid settling of the voltage at a point between the high and low logic levels that causes reduced noise margins, careful consideration of the ratio of resistors is required. This technique works well with TTL families, but care is required when using CMOS devices because the switching voltage is at 50% of the waveform. A balanced level near the threshold causes greater power dissipation and potential crossings of the threshold, resulting in unreliable output states. Because this method serves as a pullup and a pulldown termination scheme, it works well for clock signals.

5.4.4 RC Network

The RC network performs well in CMOS and TTL systems. The resistor serves to match the impedance of the trace; the capacitor holds the dc signal component, allowing the ac current to flow to the ground during the switching of logic states. Some delay is presented, but the power dissipation is much less than the parallel scheme. The resistor equals Z_0', and the capacitor is small (<600 pF). Their RC time constant must be greater than twice the loaded line propagation delay. This scheme is recommended for buses that have similar layouts for all their lines (and similar Z_0s) because RC termination networks are available in single-in-line packages (SIPs), which require much less space than a two-discrete-component solution.

5.4.5 Diode Network

The diode termination network is used frequently for termination on differential networks. It limits overshoot to approximately 1 V and has low power dissipation, but the diodes' response at the switching frequency needs to be verified. An important observation is that the energy is not absorbed in this method. Overshoot is limited at each receiver that has these diodes at their inputs, but the overall energy is not absorbed; thus' reflections occur throughout the trace.

5.5 MISCELLANEOUS FACTORS

The following paragraphs describe factors that affect the transmission: supply voltage, temperature, processes, crosstalk, and layout.

5.5.1 Supply Voltage

The supply voltage to the devices affects their rise and fall times. Specifically, a higher supply voltage creates faster rise and fall times because the output transistors are being driven by a higher-than-nominal voltage.

5.5.2 Temperature

Temperature alters the performance of the output devices. A higher temperature adds to CMOS devices' propagation delay times, slowing the rise and fall times.

5.5.3 Process

The best-case and worst-case processes can have a large effect on the devices. Generally, best-case factors speed up the switching, and worst-case factors delay the switching. Since devices with best-case processes have faster rise and fall times, they may cause more transmission line situations than typical devices and become the worst-case devices for system designers.

5.5.4 Crosstalk

To reduce transmission line effects, the most compact layout with short traces is advised. However, traces that run close to each other for several inches are suspect to crosstalk problems. To avoid this problem, increased spacing or shielding between traces must be implemented. Increasing space between traces is difficult in dense layouts, which causes crosstalk to be another complicated issue. References 5-1, 5-7, 5-8, and 5-9 (see **5.18 REFERENCES**) discuss this subject in detail.

5.5.5 Layout

The following general comments relate to PCB layout and components:

1. Evenly distributed devices along a trace are preferred to lumped loads because there are fewer large discontinuities in impedances.
2. Avoid stubs and T's in layout for critical signals, which cause impedance discontinuities. Daisy-chain and star methods are preferred.
3. Sockets and vias add small amounts of capacitance (less than 2 pF and 0.5 pF, respectively). This amount can be added into calculations for distributed capacitance, if desired.
4. One package type of the future is tape-automated bonding (TAB). The pin orientation and package size require less interconnect area, potentially reducing trace lengths and transmission line effects. Another method used for layout is the multichip module, which mounts several devices together on silicon substrates to reduce interconnect area.
5. Although exotic materials are available, they are not normally used in digital PCB construction. Generally, they have lower dielectric constants, meaning their traces have smaller propagation delays. Reference 5-10 provides a list of these materials.
6. Controlled-impedance PCBs give a predictable environment for transmission line behavior. They can be expensive, but they are recommended for high-performance or prototyping applications.
7. Bidirectional signals often use parallel, RC, or Thevenin terminations at both ends of the PCB trace.
8. Multiwire configurations for PCBs have applications in this high-speed world of digital designs. They have lower dielectric constants, providing faster unloaded propagation delay times.
9. A detailed means of accounting for all types of signal integrity issues is described in Reference 5-11. It focuses on noise budgets that examine tolerances throughout a system.

5.6 TIME-DOMAIN REFLECTOMETER

Reference 5-1 (see **5.18 REFERENCES**) promotes a means of lab verification using a time-domain reflectometer (TDR), which is basically a step generator that stimulates a PCB trace for an oscilloscope display to monitor. The TDR is recommended for designers who desire to observe transmission lines on a PCB.

Because it has a 1-V swing and uses an unpopulated PCB, some expertise is necessary to translate the results to the CMOS and TTL world.

5.7 CONCLUSION

PCB layout is not trivial. A poor layout can prevent a well-simulated design from working properly. With devices achieving faster rise and fall times and PCB materials not improving, the transmission line effects must be resolved, usually by using terminations. This solution involves more components and more power consumed by the printed circuit assembly, but it provides improved performance and greater reliability.

The discussions presented in this document are designed to give a background into transmission line effects. To fully understand these effects, predicted results whether from hand analysis or computer-aided programs, need to be correlated with actual hardware. This complete investigation reduces the unpredictability associated with transmission line effects.

5.8 LOADED MICROSTRIP – EXAMPLE 1

A MC68030 with 3 ns rise and fall times drives a unidirectional 4-in surface microstrip trace with six devices (e.g., FAST family) distributed along the trace. Is termination necessary for transmission line effects?

5.8.1 Microstrip Geometry

Given as W = 0.010 in, t = 0.002 in, H = 0.012 in, and E_r = 4.7.

Calculate Characteristic Impedance and Propagation Delay:

Using the surface microstrip equations in Figure 5-3(a):

$$Z_o = \frac{87}{\sqrt{E_r + 1.41}} \times \ln\left\{\frac{5.98 \times H}{0.8xW + t}\right\} = \frac{87}{\sqrt{4.7 + 1.41}} \times \ln\left\{\frac{5.98 \times 12}{0.8x10 + 2}\right\} = 69.4\ \Omega$$

$$t_{pd} = 1.017 \times \sqrt{.475xE_r+0.67} = 1.73\ \text{ns/ft} = 0.144\ \text{ns/in.}$$

5.8.2 Consider Loading

C_d is the distributed capacitance per length of the load, which is the total input capacitance of the receiving devices divided by the length of the trace. The loading of each device is given in its specifications For this case, each C_L = 7 pF. Since there are six devices distributed along the 4-in trace, C_d = 6 x C_L / Trace Length = 42 pF/ 4 in = 10.5 pF/in.

C_o is the intrinsic capacitance of the trace. $C_o = t_{pd}/Z_o$ = 0.0250 nF/ft = 25.0 pF/ft = 2.08 pF/in.

$t_{pd}' = t_{pd} \times \sqrt{1+10.5/2.08} = 4.26$ ns/ft. This is the new one-way propagation time for the signal from the MC68030.

5.8.3 Transmission Line Effects

If $2 \times t_{pd}' \times$ Trace Length $\leq t_r$ or t_f, then the ringing and other transmission line effects are masked during the rise and fall times, and no termination will be needed.

$2 \times t_{pd}' \times$ Trace Length = 2 x 4.25 ns/ft x 1/3 ft = 2.83 ns.

t_r and t_f = 3 ns; since 2.83 ≤ 3, no termination is required, but this is a marginal case. Note that if the guideline promoted by some device manufacturers of $3 \times t_{pd}' \times$ Trace Length is used, then termination would be required.

For comparison, if a 69-Ω stripline was used instead of a 69-Ω microstrip, would termination be required?

$t_{pd} = 1.017 \times \sqrt{E_r} = 2.20$ ns/ft.

$C_o = t_{pd}/Z_o =$ 35 pF/ft = 2.91 pF/in.

C_d is same as above (= 10.5 pF/in).

$t_{pd}' = t_{pd} \times \sqrt{1+C_d/C_o} = 4.72$ ns/ft.

$2 \times t_{pd}' \times$ Trace Length = 2 x 4.72 ns/ft x 1/3 ft = 3.14 ns.

The rise and fall times are 3 ns. Since 3.14 > 3 ns, termination is needed for the 69-Ω stripline case.

In comparison, note that C_o is greater for the stripline, which tends to lessen the effect of loading in the t_{pd}' equation. However, the unloaded t_{pd} is substantially greater for the stripline than the microstrip, and this factor prevents the transmission line effects from being masked during the rise and fall times, according to these equations. Notice that the unloaded propagation delay calculations depend only on E_r, the dielectric constant, and not on the trace geometries.

5.9 LOADED STRIPLINE – EXAMPLE 2

A MC88100 with rise and fall times of 2 ns drives four MC88200s distributed over an 8-in stripline trace.

5.9.1 Stripline Geometry

Assume a stripline with B = 0.020 in, W = 0.006 in, t = 0.0014 in, and $E_r = 4.6$.

From Figure 5-3(c), the equations for Z_o and t_{pd} for a stripline are:

$$Z_o = \frac{60}{\sqrt{E_r}} \times \ln\left\{\frac{4xB}{0.67\pi W \times (0.8 + t/W)}\right\} = \frac{60}{\sqrt{4.6}} \times \ln\left\{\frac{4x20}{0.67\pi x6 \times (0.8+1.4/6)}\right\} = 50.7\ \Omega$$

$$t_{pd} = 1.017 \times \sqrt{E_r} = 2.18 \text{ ns/ft} = 0.182 \text{ ns/in.}$$

Calculate the intrinsic capacitance of the trace:

$$C_o = t_{pd}/Z_o = 0.043 \text{ nF/ft} = 43.0 \text{ pF/ft} = 3.58 \text{ pF/in.}$$

5.9.2 Consider Loading

Each MC88200 has a capacitive load of 15 pF. The total load is 60 pF for the four MC88200s. C_d is the distributed load per length for the trace. C_d = 60 pF/ 8 in = 7.5 pF/in.

$$t_{pd}' = t_{pd} \times \sqrt{1+C_d/C_o} = 2.18 \times \sqrt{1+7.50/3.58} = 3.84 \text{ ns/ft.}$$

5.9.3 Transmission Line Effects

2 x t_{pd}' x Trace Length < t_r or t_f is the condition of interest.

2 x 3.84 ns/ft x 2/3 ft = 5.10 ns. Since this is not less than t_r or t_f (2 ns), termination is recommended to absorb transmission lines effects.

Look at a 50.7-Ω microstrip line loaded similarly.

$$t_{pd} = 1.017 \times \sqrt{.475E_r+.67} = 1.72 \text{ ns/ft.}$$

$$C_o = t_{pd}/Z_o = 2.83 \text{ pF/in.}$$

$$t_{pd}' = t_{pd} \times \sqrt{1+C_d/C_o} = 3.29 \text{ ns/ft.}$$

Is this a transmission line? 2 x t_{pd}' x Trace Length = 2 x 3.29 ns/ft x 2/3 ft = 4.38 ns. Since this is greater than t_r or t_f, termination is recommended.

For a trace this long, the rise and fall times are too fast to handle without termination. Trace length does not directly figure into the calculations for t_{pd} or Z_o, but it is an important factor in the relationship that determines if a trace is a transmission line.

5.10 LATTICE DIAGRAM – EXAMPLE 3

Chip A is driving an 8-in 90-Ω stripline trace on a glass-epoxy PCB (Er = 4.7) loaded with four devices (chip B) of 15-pF load capacitance each. Chip A has an output low impedance of 20 Ω and an output high impedance of 120 Ω. Chip B has

a very large input impedance (100 kΩ). Voltages at each end of the trace are examined by using a lattice diagram.

5.10.1 Calculate Z_o' and t_{pd}'

$Z_o' = Z_o / \sqrt{1+C_d/C_o}$

$t_{pd} = t_{pd} \times \sqrt{1+C_d/C_o}$

C_d = (4 devices x (15 pF/device)) / 8 in of trace = 7.5 pF/in.

To calculate C_o, first calculate t_{pd} for a stripline of E_r = 4.7 using the equations in Figure 5-3(c). $t_{pd} = 1.017 \times \sqrt{E_r} = 2.20$ ns/ft

Calculate intrinsic capacitance: $C_o = t_{pd}/Z_o = 2.04$ pF/in

Inserting the C_o and C_d values yields: $Z_o' = 41.6\ \Omega$ and $t_{pd}' = 4.76$ ns/ft.

The following procedure is outlined in the section entitled 5.3.1 **Lattice Diagram** (see Figure 5-5(a)).

5.10.2 Switching Levels

$V_{oh} = 4.75$ V, $V_{ol} = 0.25$ V.

5.10.3 Output High to Output Low

$\Delta V_{out} = V_{final} - V_{initial} = V_{ol} - V_{oh} = -4.50$ V.

Use $Z_d = 20\ \Omega$ because the final value is the driven low case.

The voltage divider at the driving end of the trace for $Z_d = 20\ \Omega$ and $Z_o' = 41.6\ \Omega$ gives: $\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o'+Z_d} = -3.04$ V.

5.10.4 Calculate Reflection Coefficients

$$\rho_s = \frac{Z_d-Z_o'}{Z_d+Z_o'} = \frac{20-41.6}{20+41.6} = -0.351 \qquad \rho_L = \frac{Z_L-Z_o'}{Z_L+Z_o'} = \frac{100k-41.6}{100k+41.6} \approx 1\ (=0.999)$$

The voltages and currents down the trace are modified proportionally with the reflection coefficients. $V_{initial} = V_{oh} = 4.75$ V.

$V_0 = \Delta V_0 + V_{initial} = 1.71$ V.

$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = -1.33$ V.

$V_2 = \Delta V_0 \times \rho_L \times (1+ \rho_s) + V_0 = -0.261$ V.

$V_3 = \Delta V_0 \times \rho_L \times \rho_s \times (1 + \rho_L) + V_1 = 0.803$ V.

$V_4 = \Delta V_0 \times \rho_L^2 \times \rho_s \times (1 + \rho_s) + V_2 = 0.430$ V.

$V_5 = \Delta V_0 \times \rho_L^2 \times \rho_s^2 \times (1 + \rho_L) + V_3 = 0.058$ V.

$V_6 = \Delta V_0 \times \rho_L^3 \times \rho_s^2 \times (1 + \rho_s) + V_4 = 0.188$ V.

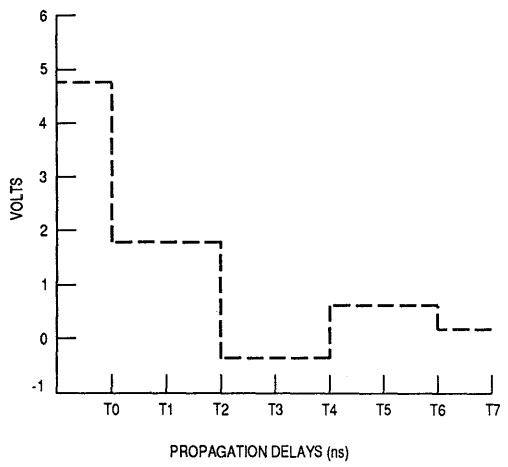
$V_7 = \Delta V_0 \times \rho_L^3 \times \rho_s^3 \times (1 + \rho_L) + V_5 = 0.319$ V.

From Figure 5-5(a), V_{final} is calculated:

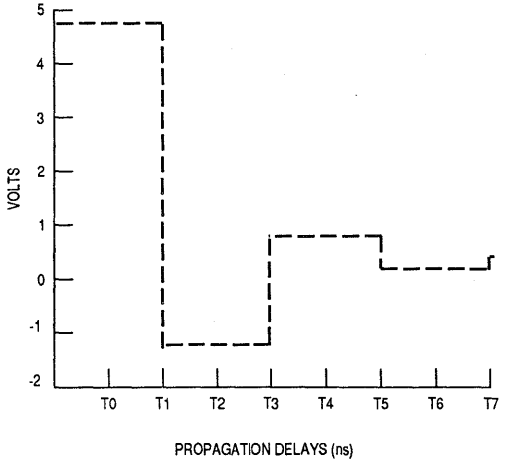
$$V_{final} = \Delta V_{out} \times \frac{Z_L}{Z_L + Z_d} + V_{initial} = -4.5 \times \frac{100k}{100k + 20} + 4.75 = 0.251 \text{ V.}$$

Notice that the voltages are settling towards 0.251 V, which is the final value in this transition.

The voltage versus time plots are show in the following graphs:



(a) Driving Device



(b) Receiving Device

5.10.5 Output Low to Output High

$\Delta V_{out} = V_{final} - V_{initial} = V_{oh} - V_{ol} = 4.50$ V.

For this case, $Z_d = 120\ \Omega$, since the final state is the high-driving state.

The voltage divider at the driving end of the trace for $Z_d = 120\ \Omega$ and $Z_o' = 41.6\ \Omega$ gives: $\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o' + Z_d} = 1.16$ V.

5.10.6 Calculate Reflection Coefficients

$$\rho_S = \frac{Z_d - Z_o'}{Z_d + Z_o'} = \frac{120-41.6}{120+41.6} = 0.485 \quad \rho_L = \frac{Z_L - Z_o'}{Z_L + Z_o'} = \frac{100k-41.6}{100k+41.6} \approx 1 \; (= 0.999)$$

The voltages and currents down the trace are modified proportionally with the reflection coefficients. The initial voltage is the logic-low state = 0.25 V.

$$V_0 = \Delta V_0 + V_{initial} = 1.41 \text{ V.}$$

$$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = 2.57 \text{ V.}$$

$$V_2 = \Delta V_0 \times \rho_L \times (1 + \rho_S) + V_0 = 3.13 \text{ V.}$$

$$V_3 = \Delta V_0 \times \rho_L \times \rho_S \times (1 + \rho_L) + V_1 = 3.69 \text{ V.}$$

$$V_4 = \Delta V_0 \times \rho_L^2 \times \rho_S \times (1 + \rho_S) + V_2 = 3.96 \text{ V.}$$

$$V_5 = \Delta V_0 \times \rho_L^2 \times \rho_S^2 \times (1 + \rho_L) + V_3 = 4.23 \text{ V.}$$

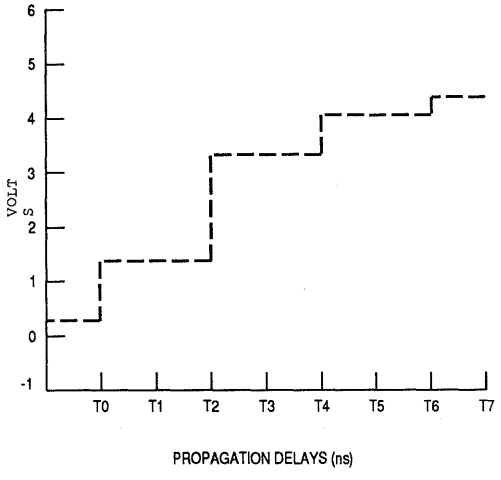
$$V_6 = \Delta V_0 \times \rho_L^3 \times \rho_S^2 \times (1 + \rho_S) + V_4 = 4.36 \text{ V.}$$

$$V_7 = \Delta V_0 \times \rho_L^3 \times \rho_S^3 \times (1 + \rho_L) + V_5 = 4.50 \text{ V.}$$

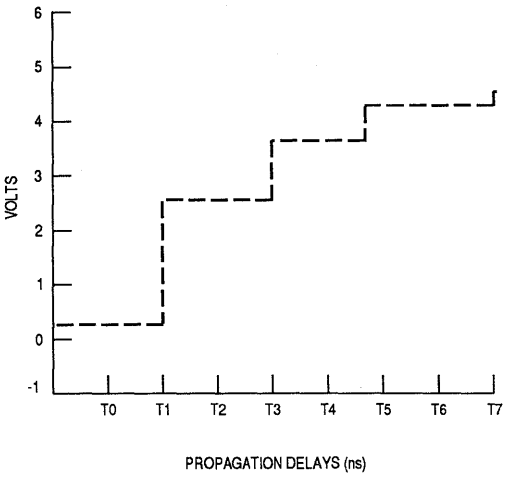
$$V_{final} = \Delta V_{out} \times \frac{Z_L'}{Z_L' + Z_d} + V_{initial} = 4.5 \times \frac{100k}{100k + 20} + 0.25 = 4.75 \text{ V.}$$

In this case, the voltage creeps up toward the final value of 4.75 V. The ringing effects are not as severe as the high-to-low case, but the high-voltage value is not reached until several propagation delays have transpired. The stair-stepping occurs because the loaded characteristic impedance is lower than the output impedance of the driving device. In the first case, ringing occurs because the output impedance of the driving device is lower than the loaded characteristic impedance of the trace.

The voltage versus time plots are shown in the following graphs:



(a) Driving Device



(b) Receiving Device

5.11 LATTICE DIAGRAM WITH SERIES TERMINATION – EXAMPLE 4

The conditions are the same as Example 3 (see **5.10 LATTICE DIAGRAM**). A series termination resistor of 25 Ω is inserted to reduce ringing in the high-to-low switching state. The lattice diagram analysis for both states is performed to show the series termination tradeoffs in this situation.

5.11.1 Series Termination

For series termination, a resistor is placed in series at the driving device's output, altering the output impedance value used in the calculations.

$Z_d' = Z_d + Rs$, where Rs is the series resistor and Z_d' is the new output impedance value used in calculations.

5.11.2 Output High-to-Low Switching Case

Choose Rs = 25 Ω. Now $Z_d' = Z_d + Rs = 25 + 20 = 45\ \Omega$.

Z_o' remains the same (41.6 Ω).

5.11.3 Recalculate ρ_s

$$\rho_s = \frac{Z_d - Z_o'}{Z_d + Z_o'} = \frac{45-41.6}{45+41.6} = 0.0393 \qquad \rho_L = \frac{Z_L - Z_o'}{Z_L + Z_o'} = \frac{100k-41.6}{100k+41.6} \approx 1\ (= 0.999)$$

5.11.4 Calculate Voltages

Output voltage levels are V_{oh} = 4.75 V, V_{ol} = 0.25 V.

For the high-to-low case, $\Delta V_{out} = V_{final} - V_{initial} = V_{ol} - V_{oh} = -4.5$ V.

The voltage divider at the driving end of the trace yields:

$$\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o'+Z_d} = -2.16 \text{ V.}$$

The voltages and currents down the trace are modified proportionally with the reflection coefficients. Use initial voltage of high case (= 4.75 V).

$V_0 = \Delta V_0 + V_{initial} = 2.59$ V.

$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = 0.429$ V.

$V_2 = \Delta V_0 \times \rho_L \times (1+ \rho_s) + V_0 = 0.344$ V.

$V_3 = \Delta V_0 \times \rho_L \times \rho_s \times (1 + \rho_L) + V_1 = 0.259$ V.

$V_4 = \Delta V_0 \times \rho_L^2 \times \rho_s \times (1 + \rho_s) + V_2 = 0.256$ V.

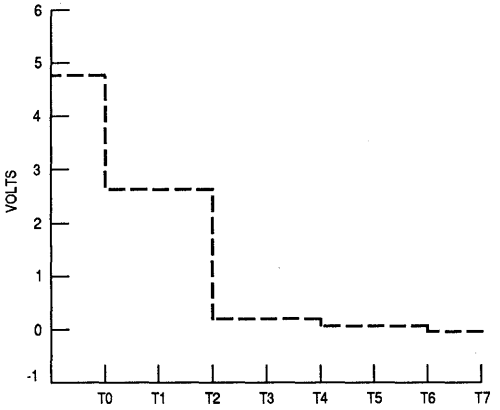
$V_5 = \Delta V_0 \times \rho_L^2 \times \rho_s^2 \times (1+ \rho_L) + V_3 = 0.252$ V.

$V_6 = \Delta V_0 \times \rho_L^3 \times \rho_s^2 \times (1+ \rho_s) + V_4 = 0.252$ V.

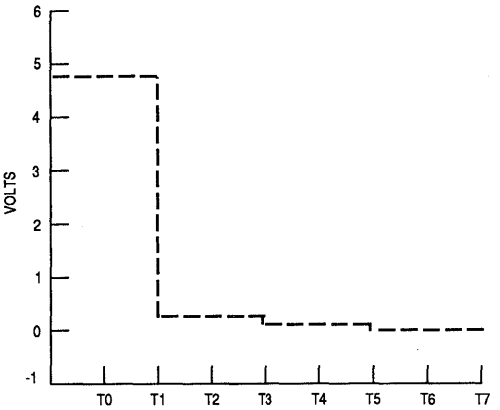
$V_7 = \Delta V_0 \times \rho_L^3 \times \rho_s^3 \times (1+ \rho_L) + V_5 = 0.252$ V.

The termination reduces the ringing, as the voltage settles to its final voltage level of 0.252 V. It is important to note that the first waveform (V_0) is a half-voltage waveform. This places the output in the region between logic levels in which noise on the trace may cause receiving devices to switch unpredictably, but only for a short time (2 x t_{pd}'). Also, the devices at the receiving end of trace see the full waveform after one propagation delay; whereas, the driving device (and devices near it) see it after two propagation delays.

The voltage versus time plots are shown in the following graphs:



(a) Driving Device



(b) Receiving Device

This is a good fix for the ringing in the high-to-low switching event. To get a complete idea of the impact of the series resistor, the low-to-high event must be examined.

5.11.5 Output Low-to-High Switching

$\Delta V_{out} = V_{final} - V_{initial} = V_{oh} - V_{ol} = 4.5$ V.

For this case, $Z_d' = Z_d + Rs = 120\ \Omega + 25\ \Omega = 145\ \Omega$, since the final state is the high-driving state and $Z_{oh} = 120\ \Omega$.

The voltage divider at the driving end of the trace, for $Z_d' = 145\ \Omega$ and $Z_o' = 41.6\ \Omega$ yields: $\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o' + Z_d} = 1.00$ V.

5.11.6 Calculate Reflection Coefficients

$$\rho_s = \frac{Z_d - Z_o'}{Z_d + Z_o'} = \frac{145\text{-}41.6}{145+41.6} = 0.554 \qquad \rho_L = \frac{Z_L - Z_o'}{Z_L + Z_o'} = \frac{100k\text{-}41.6}{100k+41.6} \approx 1\ (= 0.999)$$

The voltages and currents down the trace are modified proportionally with the reflection coefficients. The initial voltage is the logic low state = 0.25 V.

$V_0 = \Delta V_0 + V_{initial} = 1.25$ V.

$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = 2.25$ V.

$V_2 = \Delta V_0 \times \rho_L \times (1 + \rho_s) + V_0 = 2.81$ V.

$$V_3 = \Delta V_0 \times \rho_L \times \rho_S \times (1 + \rho_L) + V_1 = 3.37 \text{ V.}$$

$$V_4 = \Delta V_0 \times \rho_L^2 \times \rho_S \times (1 + \rho_S) + V_2 = 3.67 \text{ V.}$$

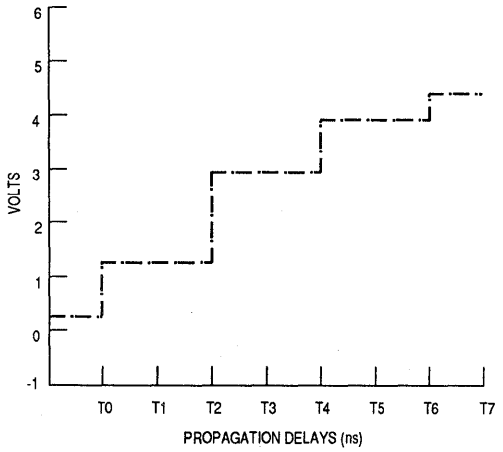
$$V_5 = \Delta V_0 \times \rho_L^2 \times \rho_S^2 \times (1 + \rho_L) + V_3 = 3.98 \text{ V.}$$

$$V_6 = \Delta V_0 \times \rho_L^3 \times \rho_S^2 \times (1 + \rho_S) + V_4 = 4.15 \text{ V.}$$

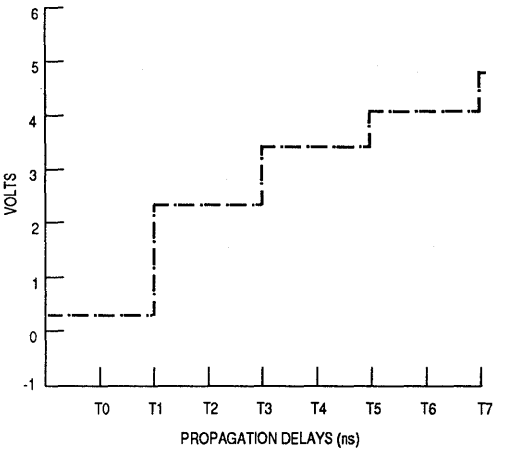
$$V_{final} = \Delta V_{out} \times \frac{Z_L'}{Z_L' + Z_d} + V_{initial} = 4.5 \times \frac{100k}{100k+145} + 0.25 = 4.74 \text{ V.}$$

The series termination exaggerates the low-to-high stair-stepping from the unterminated state because the output impedance of the driving device is larger than the unterminated state ($Z_d' = 145\ \Omega$; $Z_d = 120\ \Omega$), which is exaggerated when the termination is added. This illustrates one of the sequences involved in terminating lines—one undesirable effect is reduced while another is exaggerated.

The voltage versus time plots are shown in the following graphs:



(a) Driving Device



(b) Receiving Device

5.12 LATTICE DIAGRAM WITH PARALLEL TERMINATION – EXAMPLE 5

The trace characteristics are identical to the previous two examples. Now a parallel resistor is placed at the end of the trace to absorb the energy of the switching wave. This implementation also represents the ac switching events that occur when a RC network is used, since the capacitor, if chosen properly, limits dc current flow and does not contribute to ringing.

In the parallel termination scheme, a resistor of value Z_o' is placed at the receiving device's end of the trace. Since $Z_o' = 41.6\ \Omega$, choose $Z_L' = 40\ \Omega$.

5.12.1 High-to-Low Switching

Z_d = 20 Ω in the low-driving case. Z_o' = 41.6 Ω.

5.12.2 Recalculate Reflection Coefficients

$$\rho_S = \frac{Z_d - Z_o'}{Z_d + Z_o'} = \frac{20-41.6}{20+41.6} = -0.351 \quad \rho_L = \frac{Z_L - Z_o'}{Z_L + Z_o'} = \frac{40-41.6}{40+41.6} = 0.0196$$

This is the same source reflection coefficient as the unterminated case. The change in reflection coefficients due to the parallel termination is seen in ρ_L:

5.12.3 Calculate Voltages

$\Delta V_{out} = V_{final} - V_{initial} = V_{ol} - V_{oh} = -4.5$ V.

The voltage divider at the driving end of the trace gives:

$$\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o' + Z_d} = -4.5 \times \frac{41.6}{41.6+20} = -3.04 \text{ V.}$$

Using $V_{initial}$ = 4.75 V, the voltages are:

$V_0 = \Delta V_0 + V_{initial} = 1.71$ V.

$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = 1.65$ V.

$V_2 = \Delta V_0 \times \rho_L \times (1 + \rho_S) + V_0 = 1.67$ V.

$V_3 = \Delta V_0 \times \rho_L \times \rho_S \times (1 + \rho_L) + V_1 = 1.67$ V.

The effect of driving the line low is not achieved because of the relationship between Z_d and Z_L'. The final value depends on the voltage divider at the load where Z_L' is the value of the parallel impedance of the termination resistor and the load impedance. Since the termination resistor (R_t) is much smaller than the load impedance (R_t = 40 Ω, Z_L = 100 kΩ), it is effectively the parallel impedance value. (Recall calculations for parallel resistance from circuit theory.

$$Z_L' = \frac{R_t \times Z_L}{R_t + Z_L} \approx 40\Omega).$$

$$V_{final} = \Delta V_{out} \times \frac{Z_L'}{Z_L' + Z_d} + V_{initial} = -4.5 \times \frac{40}{40+20} + 4.75 = 1.75 \text{ V.}$$

Most driving devices can drive to the low-logic state, because the output impedance of the driving device changes as additional current is pumped into the circuit. With the nondynamic impedances used in the lattice diagram method, the final output low value is not reached, which is a limitation of this hand-analysis method.

For illustration of the parallel method, choose a larger Z_L' that allows the final output low value to be reached. For this case, choose Z_L' = 300 Ω. Check final voltage:

$V_{final} = \Delta V_{out} \times \frac{Z_L'}{Z_L'+Z_d} + V_{initial} = -4.5 \times \frac{300}{300+20} + 4.75 = 0.531$ V. This is an acceptable final voltage because it is less than the maximum input logic-low level (0.8 V).

5.12.4 Recalculate Reflection Coefficients

$\rho_S = \frac{Z_d-Z_o'}{Z_d+Z_o'} = \frac{20-41.6}{20+41.6} = -0.351$ $\rho_L = \frac{Z_L-Z_o'}{Z_L+Z_o'} = \frac{300-41.6}{300+41.6} = 0.756.$

The voltage divider at the driving end of the trace yields:

$$\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o'+Z_d} = -3.04 \text{ V.}$$

Using $V_{initial} = 4.75$ V, the voltages at each end are:

$V_0 = \Delta V_0 + V_{initial} = 1.71$ V.

$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = -0.588$ V.

$V_2 = \Delta V_0 \times \rho_L \times (1+ \rho_S) + V_0 = 0.218$ V.

$V_3 = \Delta V_0 \times \rho_L \times \rho_S \times (1 + \rho_L) + V_1 = 0.828$ V.

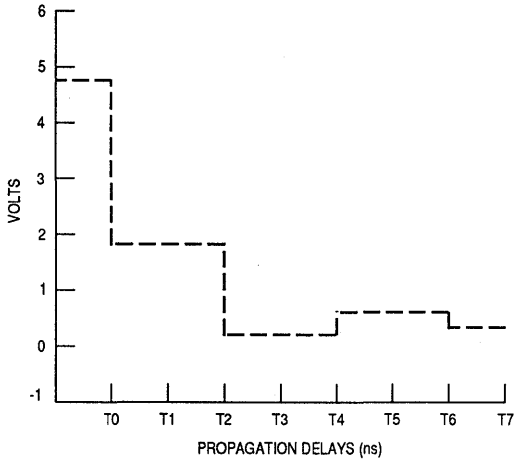
$V_4 = \Delta V_0 \times \rho_L^2 \times \rho_S \times (1 + \rho_S) + V_2 = 0.614$ V.

$V_5 = \Delta V_0 \times \rho_L^2 \times \rho_S^2 \times (1+ \rho_L) + V_3 = 0.453$ V.

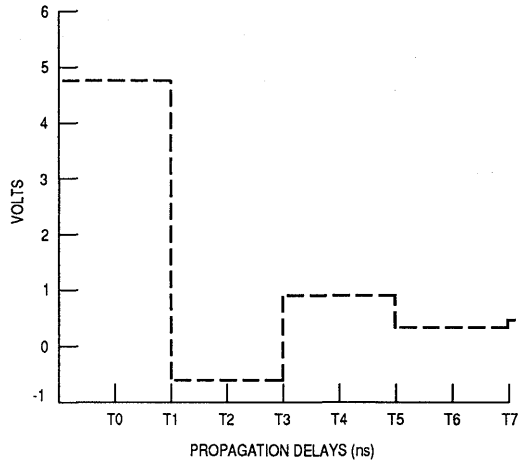
$V_6 = \Delta V_0 \times \rho_L^3 \times \rho_S^2 \times (1+ \rho_S) + V_4 = 0.509$ V.

$V_7 = \Delta V_0 \times \rho_L^3 \times \rho_S^3 \times (1+ \rho_L) + V_5 = 0.552$ V.

The voltage versus time plots are shown in the following graphs:



(a) Driving Device



(b) Receiving Device

As shown earlier, the final voltage is 0.531. In this case, notice that the first voltage (V0 = 1.71 V) at the driving device falls in the intermediate zone between logic states because Z_o' and Z_d are close in value (41.6 and 20.0 Ω's, respectively), similar to the unterminated case. The ringing is reduced at the load but still exists. As with the series termination, this reduces the ringing in the high-to-low switching event. The low-to-high switching event must be examined also.

5.12.5 Output Low-to-High Switching

$\Delta V_{out} = V_{final} - V_{initial} = V_{oh} - V_{ol} = 4.5$ V.

For this case, $Z_d = 120$ Ω, since the final state is the high-driving state. The voltage divider at the driving end of the trace, for $Z_d = 120$ Ω and $Z_o' = 41.6$ Ω:

$$\Delta V_0 = \Delta V_{out} \times \frac{Z_o'}{Z_o'+Z_d} = 1.16 \text{ V.}$$

5.12.6 Calculate Reflection Coefficients

$$\rho_S = \frac{Z_d-Z_o'}{Z_d+Z_o'} = \frac{120-41.6}{120+41.6} = 0.485 \quad \rho_L = \frac{Z_L-Z_o'}{Z_L+Z_o'} = \frac{300-41.6}{300+41.6} = 0.756$$

The initial voltage is the logic-low state = 0.25 V.

$V_0 = \Delta V_0 + V_{initial} = 1.41$ V.

$V_1 = \Delta V_0 \times (1 + \rho_L) + V_{initial} = 2.29$ V.

$V_2 = \Delta V_0 \times \rho_L \times (1+ \rho_S) + V_0 = 2.71$ V.

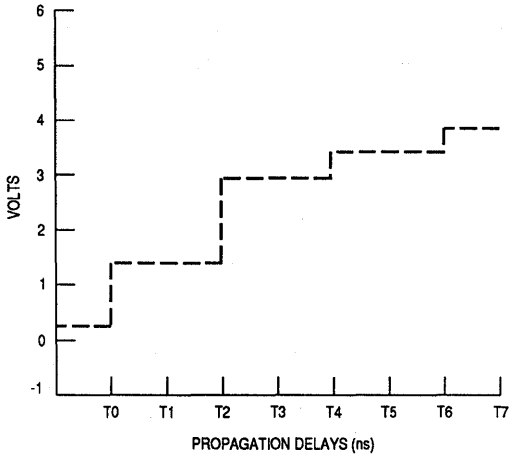
$V_3 = \Delta V_0 \times \rho_L \times \rho_S \times (1 + \rho_L) + V_1 = 3.04$ V.

$V_4 = \Delta V_0 \times \rho_L^2 \times \rho_S \times (1 + \rho_S) + V_2 = 3.19$ V.

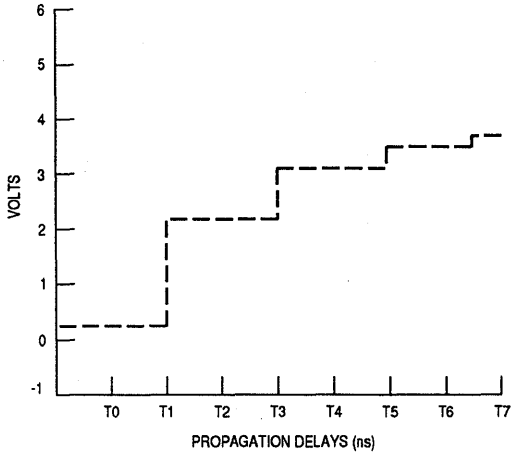
$V_5 = \Delta V_0 \times \rho_L^2 \times \rho_S^2 \times (1+ \rho_L) + V_3 = 3.31$ V.

$$V_{final} = \Delta V_{out} \times \frac{Z_L'}{Z_L'+Z_d} + V_{initial} = 4.5 \times \frac{300}{300+120} + 0.25 = 3.46 \text{ V.}$$

The voltage versus time plots are shown in the following graphs:



(a) Driving Device



(b) Receiving Device

The final voltage predicted (3.46 V) is not the final output high voltage desired (4.75 V), but it is the new quiescent value, assuming the driving device does not supply additional current. Since the output impedance of the driving device is greater than the loaded characteristic impedance, stair-stepping during the switching occurs. Any additional load exaggerates the stair-stepping, which is the case when either parallel or series termination is added. In this case, termination degrades performance by increasing the stair-stepping delays.

In general, the major drawback with the parallel method of termination is the current consumed. The dc current for the high state would be:

$$I_{final} = \frac{\Delta V_{out}}{Z_L' \times Z_d} + I_{initial} = \frac{4.5}{320} + 14.9\ mA = 28.9\ mA$$

This dc current requirement violates the drive capability of many devices.

To solve this dc current drive issue, the RC termination scheme is recommended. In the first switching events of the circuit, the capacitor is not charged to the new voltage level and allows current to flow, causing the RC termination to act identically as the parallel resistor tied to ground. After the voltage settles, the capacitor charges to the final voltage level and no dc current flows. The recommended values for the capacitor range between 200-600 pF, and the resistor's value is near Z_o'. The RC time constant must be less than twice the loaded propagation delay; otherwise, the RC network adds to the ringing.

For this case, use $Z_L' = 300\ \Omega$ and C = 300 pF. Since the time constant is 90 ns, this scheme should be satisfactory. The voltage and current levels during switching follow those given in the parallel termination resistor calculations for $R_t = 300\ \Omega$.

For designers who would like to experiment with various trace configurations and termination schemes, a spreadsheet of the formulas provides quick comparisons of the trade-offs involved in transmission line effects.

5.13 BERGERON PLOT – EXAMPLE 6

A Motorola FACT device drives two MC88200s over an unterminated 5-in surface microstrip laid out in a daisy-chain configuration with a characteristic impedance of 70 Ω and a dielectric constant of 4.7.

The Bergeron plot method will be used as outlined in this document and in Reference 5-6. The V_{in} vs. I_{in} curve for the MC88200s is given in this example. These are hand-drawn approximations, not simulated or measured values (i.e., these are for exercise only).

5.13.1 Calculate Z_o'

$$Z_o' = \frac{Z_o}{\sqrt{1+C_d/C_o}}$$

Find the intrinsic and distributed capacitance values:

$$C_o = t_{pd} / Z_o = 1.017 \times \sqrt{.475E_r+.67} / Z_o = 1.73 \text{ ns/ft} / 70\ \Omega = 2.06 \text{ pF/in.}$$

$$C_d = 2 \times 15 \text{ pF}/ 5 \text{ in} = 6 \text{ pF/in.}$$

$$Z_o' = \frac{70}{\sqrt{1+6.00/2.06}} = 35.4\ \Omega.$$ For simplicity, use 35 Ω.

This value of 35 Ω is used as a slope throughout the Bergeron plot analysis. To graph it, select voltages and currents whose differences provide an impedance of 35 Ω. For example, to get a 35-Ω slope, choose one voltage and current point at the 0-V and 0-mA point. The second point should be 3.5 V and 100 mA for a positive 35-Ω slope or 3.5 V and -100 mA for a negative 35-Ω slope (Z_o' = V/I). Lines drawn parallel to this are used throughout the plot.

5.13.2 Bergeron Plot Method

1. Start at a quiescent point (intersection of V_{ol}/I_{ol} or V_{oh}/I_{oh} and V_{in}/I_{in}). For the high-to-low transition, start at the high quiescent point; for the low-to-high transition, start at the low quiescent point.

2. Draw line with slope of $+Z_o'$ for a low-to-high transition or $-Z_o'$ for a high-to-low transition. This intersects with a V_{ol}/I_{ol} or V_{oh}/I_{oh} line, which indicates the voltage and current seen at the discontinuity between the output driving device and the trace.

3. Draw a line of slope ± Z_o' whose sign is opposite to that of the first line drawn. This line intersects with the V_{in}/I_{in} line, which provides the voltage and current seen at the receiving device's input from the trace.

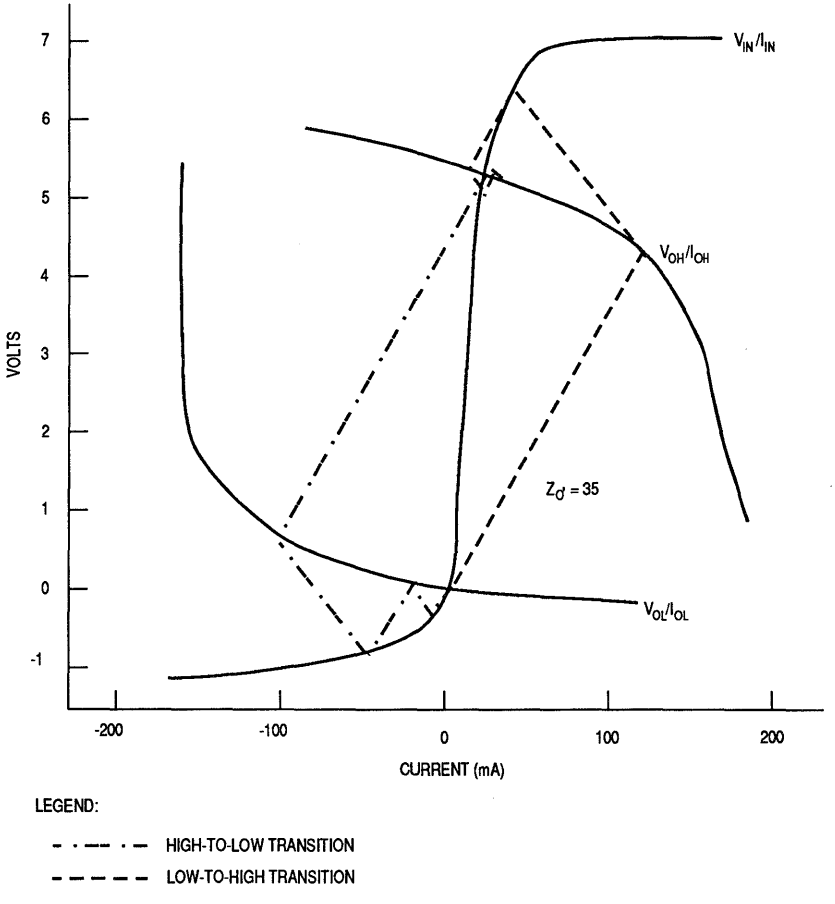
4. Repeat steps 2 and 3 until the Z_o' lines converge on the new quiescent point.

5. Take the voltage values at the intersections and plot these with respect to time. Intersection points on the V_{ol}/I_{ol} or V_{oh}/I_{oh} curves are voltages at the driving device; whereas points on the V_{in}/I_{in} curves are voltages at the receiving device. Each voltage step from a V_{ol}/I_{ol} or V_{oh}/I_{oh} line to a V_{in}/I_{in} line occurs in time t_{pd}'.

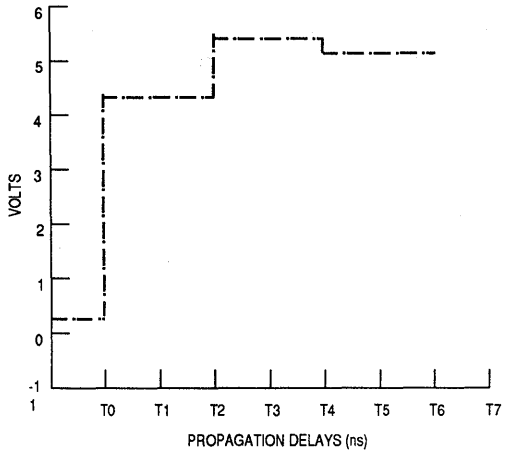
5.13.3 Summary

In the low-to-high case, the receiving devices experience voltages as high as 6.4 V; in the high-to-low case, they experience voltages as low as -1 V.

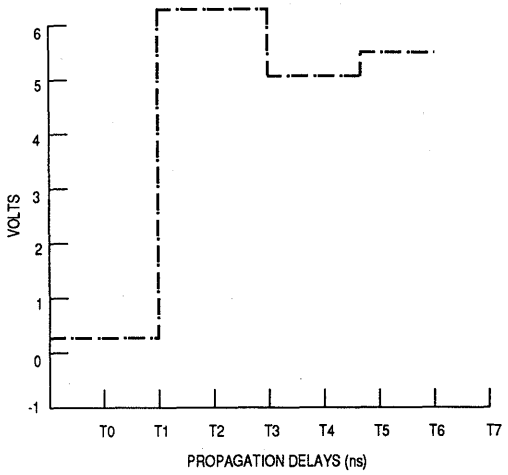
The Bergeron method requires much less calculation than the lattice diagram method. It also does not assume a linear value for Z_0 of the driving device. One drawback for the Bergeron plot is the availability of V/I curves. If the devices used have these curves available, then this is the preferred method of hand analysis. Also, these V/I curves vary over temperature, process, and supply voltage. To account for these factors in a worst-case manner may not always be straightforward. It is important to note this as a limitation.



The voltage vs. time plots are shown in the following graphs:

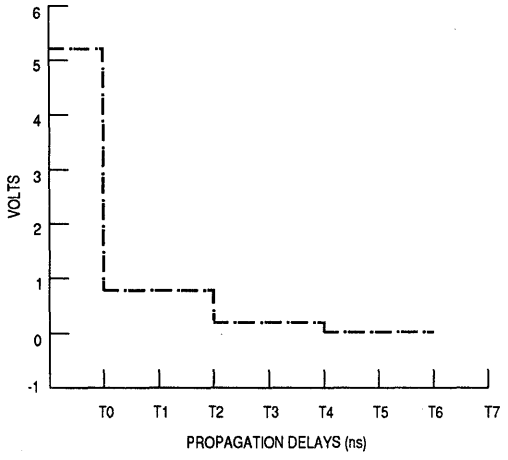


(a) Driving Device

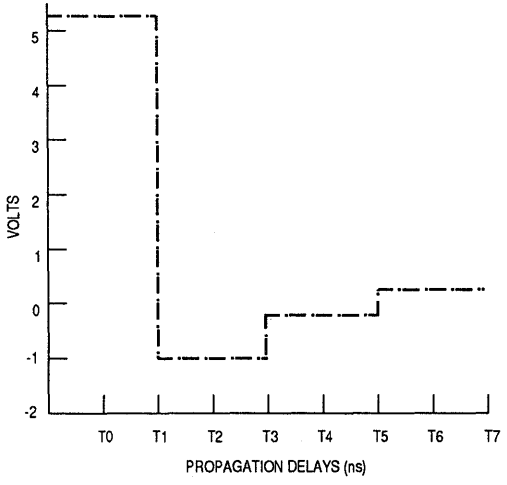


(b) Receiving Device

Unterminated Low-to-High Switching Results



(a) Driving Device



(b) Receiving Device

Unterminated High-to-Low Switching Results

5.14 BERGERON PLOT FOR DEVICES WITH HIGH DRIVE CAPABILITIES – EXAMPLE 7

Determine the transmission line effects when a device designed to drive a heavy load is connected to four devices with a load capacitance of 15 pF each over an 8-in stripline trace that is laid out in daisy-chain configuration.

5.14.1 Trace Characteristics

B = 0.016 in, W = 0.006 in,t= 0.0014 in, and E_r = 4.5.

From stripline equations in Figure 5-3(c), Z_o = 45 Ω and t_{pd} = 2.16 ns/ft = 0.180 ns/in.

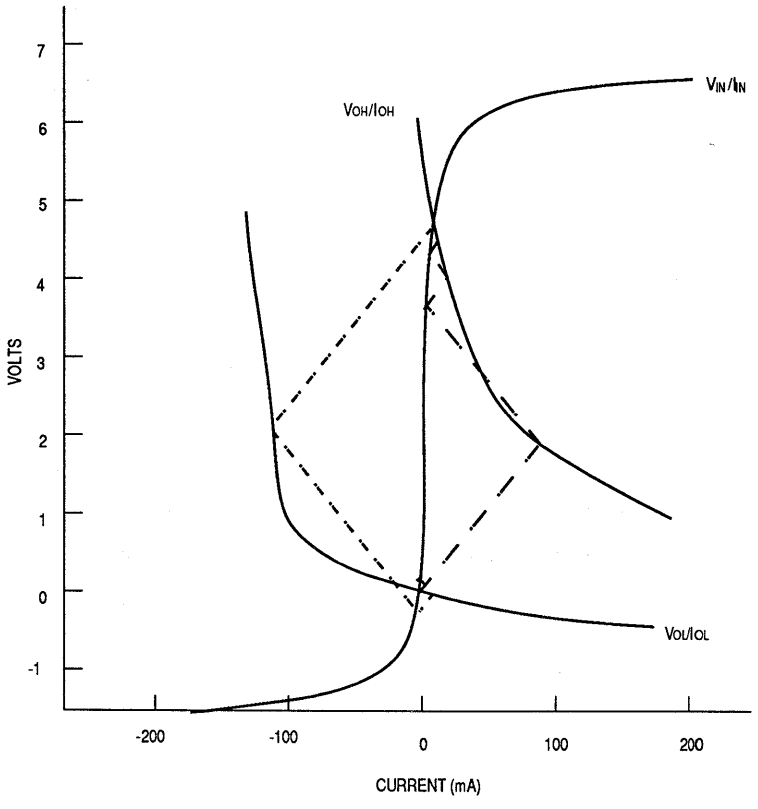
5.14.2 Calculate Effects of Load

C_d = 4 devices x 15 pF/device / 8 in = 7.5 pF/in.

$C_o = t_{pd} / Z_o$ = 2.16 ns/ft / 45 Ω x 1000 pF/1 nF x 1 ft/12 in = 4 pF/in.

$$Z_o' = \frac{Z_o}{\sqrt{1+C_d/C_o}} = 26.5\ \Omega \approx 25\ \Omega.$$

Perform Bergeron plot with Z_o' = 25 Ω.

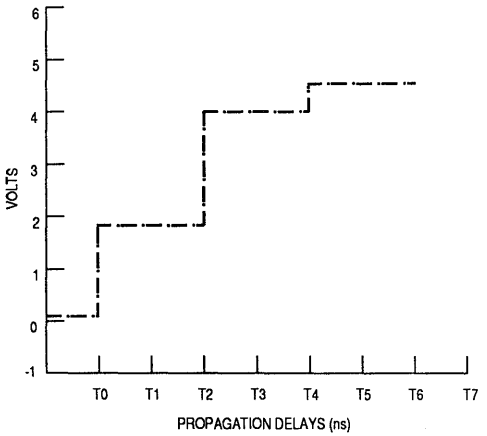


LEGEND:

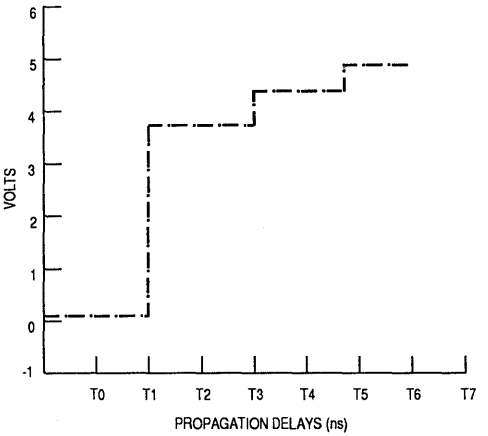
HIGH-TO-LOW TRANSITION

LOW-TO-HIGH TRANSITION

The voltage vs. time plots are shown in the following graph:

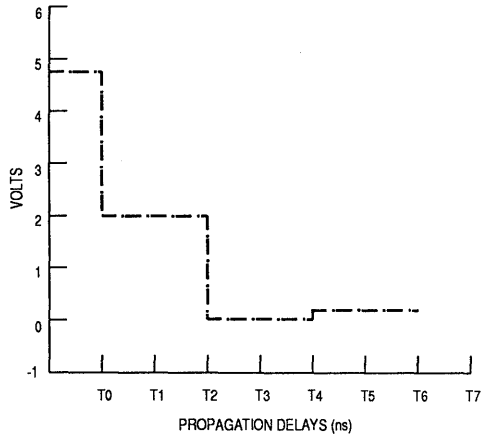


(a) Driving Device

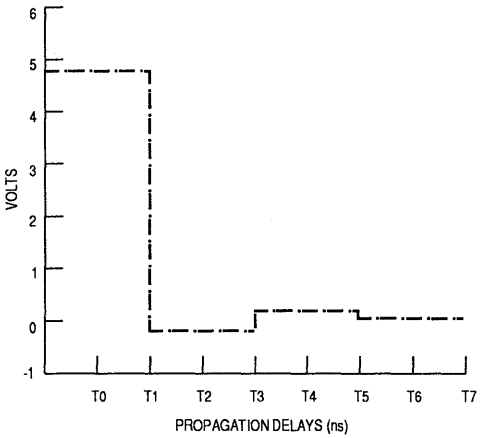


(b) Receiving Device

Unterminated Low-to-High Switching Results



(a) Driving Device



(b) Receiving Device

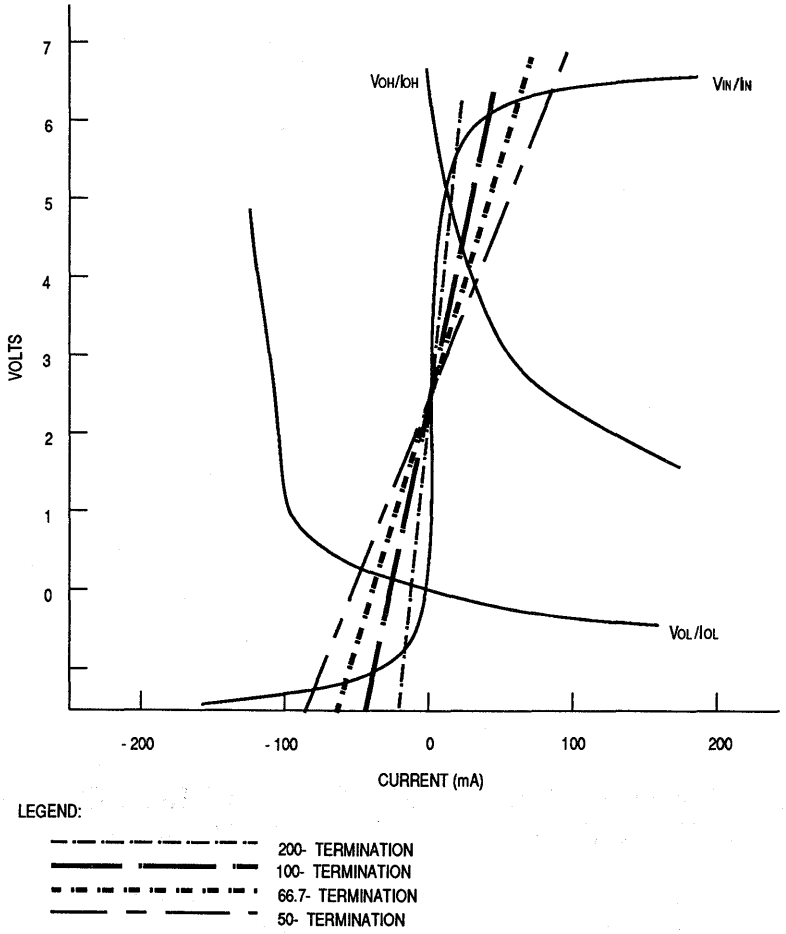
Unterminated High-to-Low Switching Results

5.15 BERGERON PLOT WITH PARALLEL TERMINATION – EXAMPLE 8

The device and trace configuration is identical to the previous example. The parallel form of termination is examined to demonstrate how the V/I curves are altered when it is used.

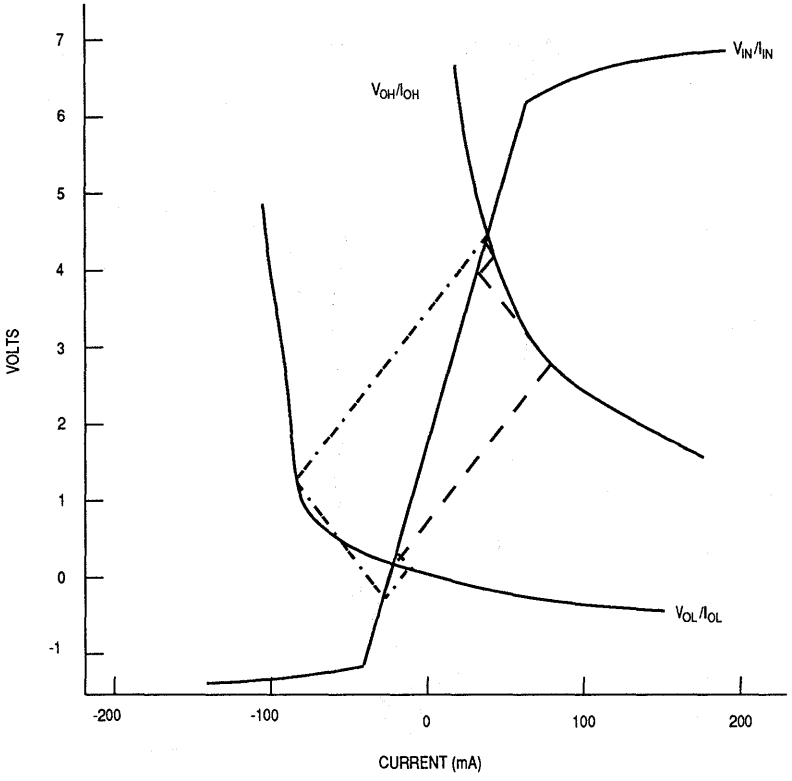
For parallel terminations, the V_{in}/I_{in} curve is modified such that its slope matches the value of the termination. Several different values are shown in the following graph.

Notice that the clamping effect from the diodes is applied, but the linear region follows the value of the termination.



To terminate the trace, used in Example 7 (see **5.14 BERGERON PLOT DEVICES WITH HIGH-DRIVE CAPABILITES**), use a 66.7-Ω resistor tied to ground. The V_{in}/I_{in} curve is altered by changing the linear region of the curve to follow the slope of the parallel resistor (66.7 Ω). This could be implemented as the RC, Thevenin or parallel method.

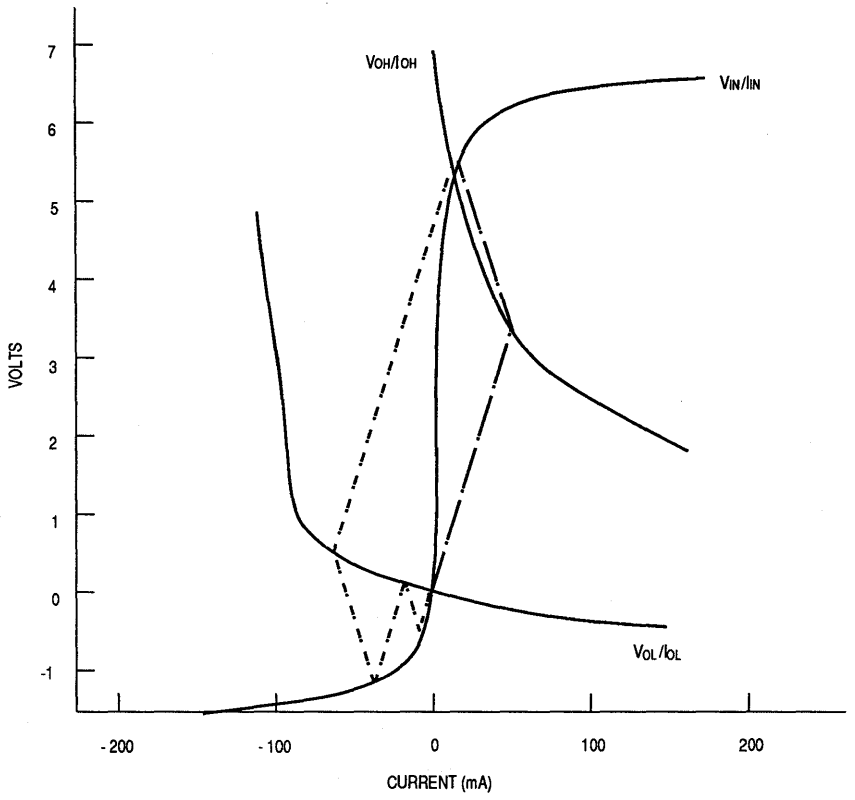
Although the ringing is reduced, there is not much difference from the unterminated state because this particular driving device handles the heavily loaded trace well.



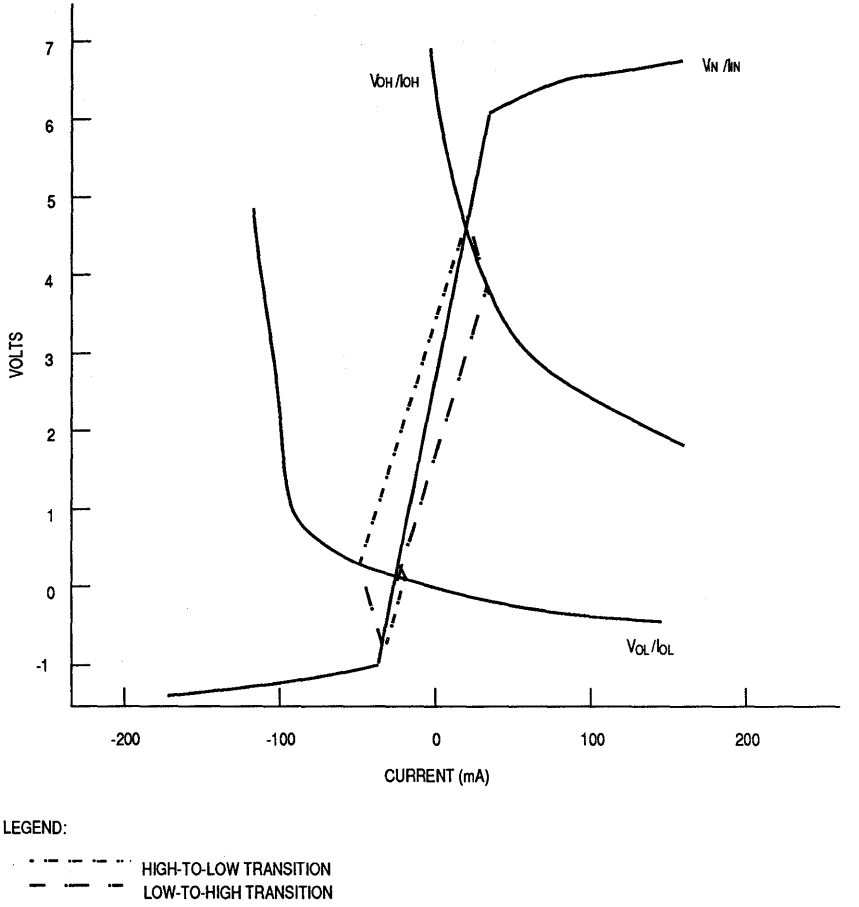
5.16 BERGERON PLOT FOR A HEFTY OUTPUT DRIVING DEVICE CONNECTED TO A LIGHTLY LOADED TRACE – EXAMPLE 9

The same driving and receiving curves are used as in the previous two examples, but a different characteristic impedance is used to illustrate what happens when a driving device that handles the heavily loaded trace well is loaded lightly. Notice that the undershoot is greater and the ringing is substantial in the high-to-low switching case. There is less stair-stepping than in the case with $Z_0' = 25\ \Omega$. The

characteristic of this output buffer shown in Examples 7-9 is if the trace is heavily loaded to match the high-to-low drive capabilities, then stair-stepping occurs in the low-to-high drive case; if the trace is lightly loaded to match the low-to-high drive capabilities, then ringing in the high-to-low drive case occurs. The designer must understand which case is worse for the design and account for it accordingly.



To alleviate the ringing from the lightly loaded condition shown previously, terminate using a 100-Ω resistor implemented in parallel or RC fashion. The V_{in}/I_{in} curve is modified by using the slope of the parallel termination (100 Ω) for the linear region. This absorbs the ringing and overshoot well.



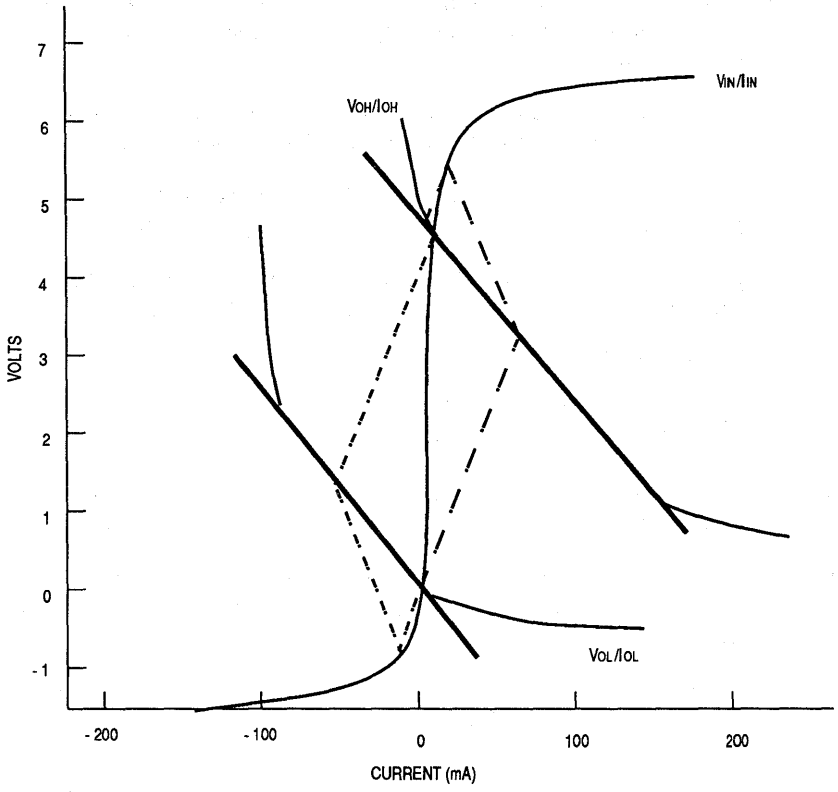
5.17 BERGERON PLOT FOR SERIES TERMINATION – EXAMPLE 10

For demonstration, use a trace with a loaded characteristic impedance of 50 Ω and devices with voltage versus current curves similar to the devices used in the previous three examples. The driving devices have similar output impedances for the high-to-low and low-to-high cases.

If a series resistor method is chosen for termination, the output curves of the driving device are modified. Z_d, the output impedance of the driving device, adds to R_S, the series resistor, to give the new output impedance, $Z_{d'} = 25\ \Omega$. This is the slope of the output curves for the driving device at both the high and the low quiescent points.

Below is an example of a 25-Ω series termination scheme. The total output impedance is shown in the high and low states as a line with slope of 25.

In this case, overshoot and undershoot occur. The stair-stepping is no longer evident, and the ringing is minimal.



LEGEND:

- HIGH-TO-LOW TRANSITION
- LOW-TO-HIGH TRANSITION
- 25- SERIES TERMINATION
- V/I CURVES ASIDE FROM TERMINATION

5.18 REFERENCES

5-1. Blood, William R., *MECL System Design Handbook*, Motorola Inc.,1983.

5-3. Seshadri, S. R., *Fundamentals of Transmission Lines and Electromagnetic Fields*, Reading, Massachusetts: Addison-Wesley Publishing Co., Inc., 1971.

5-4. Morris, Robert L. and Miller, John R., *Designing with TTL Integrated Circuits*, New York, New York: McGraw-Hill Book Company, 1971.

5-5. Stehlin, Robert A., "Bergeron Plots Predict Delays in High-Speed TTL Circuits", *EDN*, vol. 29, no. 23, Nov, 1984, pp. 293-298.

5-6. Singleton, Robert S.,, "No Need to Juggle Equations to Find Reflection--Just Draw Three Lines", *Electronics*, vol. 41, no. 22 ,Oct, 1968, pp 93-9.

5-7. *Motorola FACT Data*, Motorola Inc., 1988

5-8. Wakeman, Larry, "Transmission-Line Effects Influence High-Speed CMOS", *EDN*, vol. 29, no. 12, June, 1984, pp. 171-77.

5-9. DeFalco, John A., "Reflection and Crosstalk in Logic Circuit Interconnections", *IEEE Spectrum*, vol. 7 no. 7, July 1970, pp. 44-50.

5-10. Ritchey, Lee W., "Controlled Impedance PCB Design", *Printed Circuit Design*, vol. 6 no. 6, June,1989, pp. 23-28 (Errata,.vol. 6, no. 8, Aug 1989, p. 78)

5-11. Tummala and Rymaszewski, *Microelectronics Packaging Handbook*, NY, NY: Van Nostrand Reinhold, 1989.

SECTION 6
SPICE Modeling Information

This section is in development and will be furnished at a later date. Ensure that your registration cards are filled out and mailed in so that your copy can be sent to you.

SECTION 7
MC68040 to MC68020/MC68030 Bus Adapter Design

The MC68040 is user object code compatible with the MC68020/MC68030 and the MC68881/MC68882. Given an existing MC68030/MC68882 or an MC68020/MC68881 design, a quick upgrade may be implemented by an MC68040 emulation board that plugs into the existing design.

The bus adapter is used to translate MC68040 bus cycles into single/multiple MC68030-like bus cycles. The bus adapter design consists mostly of programmable array logic (PALs) and standard off-the-shelf discrete components. If a lower chip count is needed, a high-speed CMOS ASIC implementation of this design may be used to integrate all of this logic into a single chip. Figure 7-1 shows a simplified block diagram of the bus adapter.

The bus adapter is designed to include the following features:

- Support for an asynchronous interface
- Dynamic bus sizing for 8-, 16-, and 32-bit ports
- Retry mechanism handled independently of the MC68040
- The bus-arbitration protocol found in the MC68020 emulated

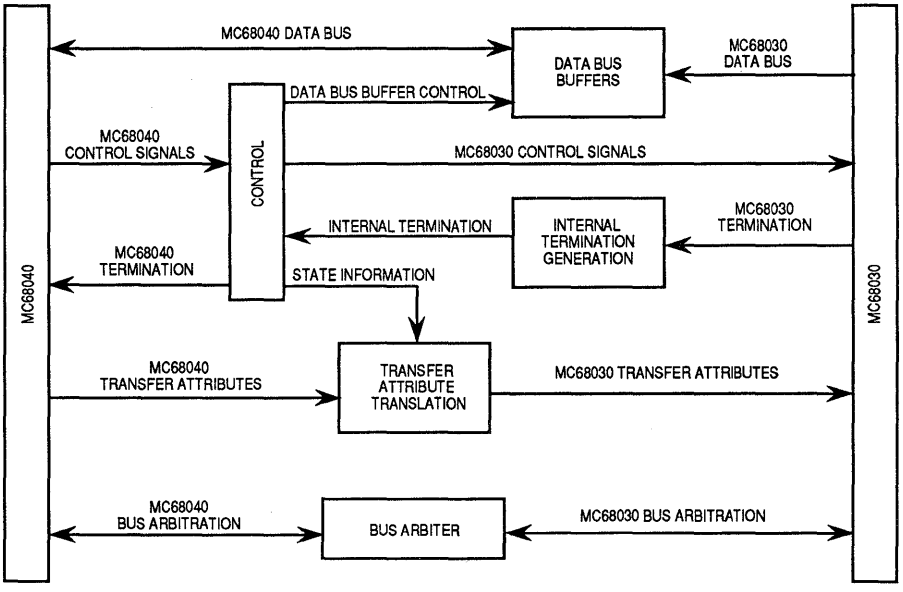


Figure 7-1. MC68040 to MC68020/MC68030 Bus Adapter

7.1 COMPATIBILITY

A daughter board design needs to address system software and hardware compatibility. The following paragraphs discuss compatibility issues in the design of the bus adapter.

7.1.1 Software

System software issues include differences in the memory management structure, internal cache initialization and management, software emulation of unimplemented floating-point instructions, emulation of unsupported instructions, and the exclusion of the instruction continuation exception model (i.e., bus error). These software issues and the solutions are application specific. The system designer should decide whether existing firmware and software should be modified and/or whether additional firmware be added onto the daughter board to address the system software issues. Therefore, detailed discussion of system software issues are not addressed.

7.1.2 Hardware

The main challenge in solving bus compatibility is dynamic bus sizing. The bus adapter must have the capability of performing multiple bus cycles per operand request. The bus adapter handles this problem by having DMA capability to perform multiple bus cycles to the system. From the MC68040's perspective, the bus adapter is performing a single MC68040 bus access.

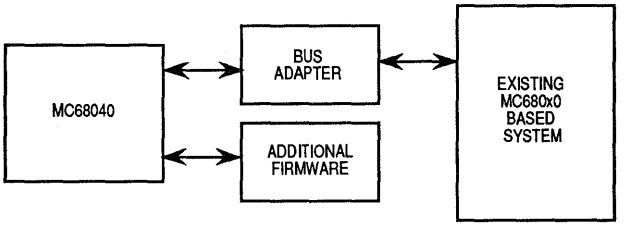
The second challenge in the bus adapter design is that the MC68040 may interpret a retry (both $\overline{TA}$ and $\overline{TEA}$ asserted) as a bus error. The bus adapter uses the MC68040's feature to perform burst-inhibited line tranfers. This means that a line transfer is broken down into four separate long-word bus cycles. The MC68040 recognizes a retry only on the first long-word bus access. Any retry indication on subsequent long-word transfers may be interpreted as a bus error. This problem is handled by having the bus adapter perform retrys without the MC68040's knowledge.

The third challenge in the bus adapter design is related to the definition of a line transfer. A line transfer is considered a single operand request by the MC68040. If a line transfer is subdivided into four separate long-word bus transfers, the MC68040 will not relinquish the bus until all long words had been transferred. Since the bus adapter requires the completion of a bus translation procedure before relinquishing the bus, the average bus arbitration latency is increased dramatically as compared to an actual MC68030. This is a limitation of the bus adapter, and applications requiring low bus arbitration latencies may need to modify the bus adapter design.

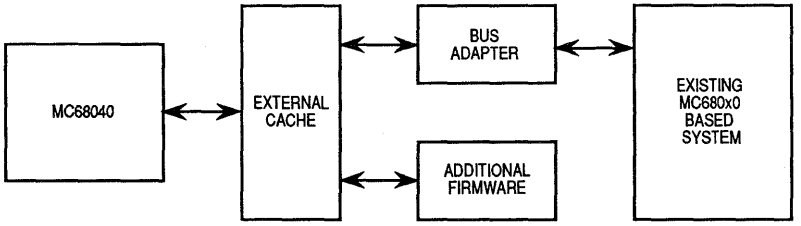
7.2 CONFIGURATIONS

Figure 7-2 shows two possible implementations of a daughter board. In the basic configuration, the additional firmware and scratchpad RAM are included in the daughter board. This additional firmware is optional if the firmware in the existing

system can accommodate the additional system software. All bus accesses, excluding those that access the daughter board resources, are made through the bus adapter. The basic configuration with an external cache is similar to the basic configuration, except that an external cache is included in the daughter board. In this configuration, only accesses that miss the external cache are made through the bus adapter.



BASIC CONFIGURATION



BASIC CONFIGURATION WITH EXTERNAL CACHE

Figure 7-2. Configurations for Daughter Board

7.3 OVERVIEW

The bus adapter is a state machine design mplemented by using PALs, registered transceivers, and TTL discrete components. The design intercepts the necessary control signals from the MC68040 and translates them into MC68030 equivalents. This process is then reversed when the termination signals from the MC68030-based system are considered.

The following paragraphs present a general overview of the units and operation of the bus adapter.

7.3.1 Units

The bus adapter consists of five main units. The main control unit is responsible for defining the bus cycle boundaries. Its function is to determine the sequence of bus cycles that occur and to generate the address strobe ($\overline{AS}$) and data strobe ($\overline{DS}$) signals to control the MC68030-based system. It provides the MC68040 termination signals, the signals which control the data bus buffer unit, and the control signals to

the transfer attribute generation unit. The main control unit is implemented using four PALs and a few discrete components.

The data bus buffer unit is used to multiplex the data byte lanes and is used as temporary registers during read cycles. Under the control of the main control unit, the data bus buffers implement dynamic bus sizing. The data bus buffer unit consists of eight 74F543 transceivers.

The internal termination generation circuitry is responsible for decoding and synchronizing the termination signals from the existing MC68030 system for use by the main control unit. The outputs of this unit are used only internally by the bus adapter. These outputs comply with the timing and logic requirements imposed by the main control unit.

The transfer attribute unit generates the lower four address bits, the size encoding, and the function codes needed by the existing MC68030 based system. This unit is controlled by the main control unit. The outputs are three-stateable and governed by the bus arbitration unit via the THREE-STATE signal. This unit is implemented by a single PAL and several discrete components. The logic in this unit may be simplified significantly if the function code outputs are not needed by the MC68030 system.

The bus arbitration unit emulates the MC68020 bus arbitration protocol. The signals produced are necessary to arbitrate the MC68040 bus. When the MC68040 bus is granted to the bus adapter, the bus arbitration unit three-states the signals produced by the transfer attribute unit.

7.3.2 Operation

The bus adapter may perform several bus cycles to the MC68030 system per bus translation request. The decision to perform multiple bus cycles for any bus translation request depends entirely on the operand size requested and the port size. For instance, a long-word access to an 8-bit port requires four byte accesses, where as a long-word access to a 16-bit port requires only two word accesses.

For example, if the MC68040 performs a long-word read cycle and the port size is a byte wide, the bus adapter performs four read bus cycles on the simulated MC68030 bus side before supplying the entire long-word operand to the MC68040. The first byte is latched in the data bus buffer unit. In the same manner, the second, third, and fourth bytes are stored in the data bus buffer unit. Once all four bytes are latched, the bytes are assembled and presented to the MC68040 as a single long word. The bus adapter terminates the MC68040 bus cycle (with a $\overline{TA}$) at the end of the fourth bus access. The bus adapter treats a line access as it would a long-word access. The only difference is that when transfer acknowledge ($\overline{TA}$) is asserted, transfer burst inhibit ($\overline{TBI}$) is asserted as well, therefore forcing fake-burst accesses.

If a retry is indicated on the second byte access, the second byte access is repeated, with the first successful byte transaction intact. The reason for having the bus adapter perform the retry cycles is that on a fake-burst transfer, a retry is

interpreted as a bus error. The bus adapter does not guarantee that a relinquish and retry operation would succeed. In most cases, the retry would be performed, but the bus will not be relinquished if the bus access is in the middle of a bus translation procedure. Note that an MC68040 line transfer is considered as one bus translation tenure.

The main function of the bus adapter during write cycles is to duplicate valid data on valid byte lanes. For example, if the MC68040 requested a byte access at address 3, valid data would be present on the lower lower (LL) byte lane since the MC68040 expects only a long-word port. However, on the MC68030 side of the bus, a byte port expects the byte data to be present on the upper upper (UU) byte lane. A word port expects the byte information to be in the upper middle (UM) byte lane and a long-word port expects that the data be present on the LL byte lane. The bus adapter duplicates the valid byte data (on the LL byte lane) onto the three byte lanes to anticipate all possible port sizes.

Note that an actual MC68030 or MC68020 duplicates the same byte data on all four of the byte lanes, instead of only three byte lanes supplied by the bus adapter. A full implementation of the MC68020/MC68030 dynamic bus sizing protocol costs more buffers than the eight already being used. As long as the byte select generation logic recommended is used, the valid byte lanes will be driven with valid data. The byte lanes that are "don't care" lanes would be driven but with no guarantees as to what the data would be.

If a bus error accurs on any bus access within a bus translation boundary, then all of the data stored within the bus adapter would be lost. This operation is different from that of an actual MC68030 or MC68020 and is a limitation of the bus adapter.

If the system port size is inconsistent within a long-word boundary, the bus adapter signals a bus error. The MC68030 imposes this restriction as well, with the exception that a violation of this restriction does not imply a bus error. The MC68020 does not impose this restriction.

7.4 SIGNAL IDENTIFICATION

For designs that can use the bus adapter as is, then the bus adapter may be used in a "black-box" approach in which only the signals necessary for interface with the system be discussed in detail. The signals designated as 040 signals may be connected directly to the MC68040. In the same manner, the signals designated as 030 signals are connected to the appropriate MC68030 bus counterpart. Figure 7-3 shows the signal connections for the MC68040 to MC68030 bus adapter design. The following paragraphs describe the signals used in the MC68040 to MC68030/MC68020 bus adapter.

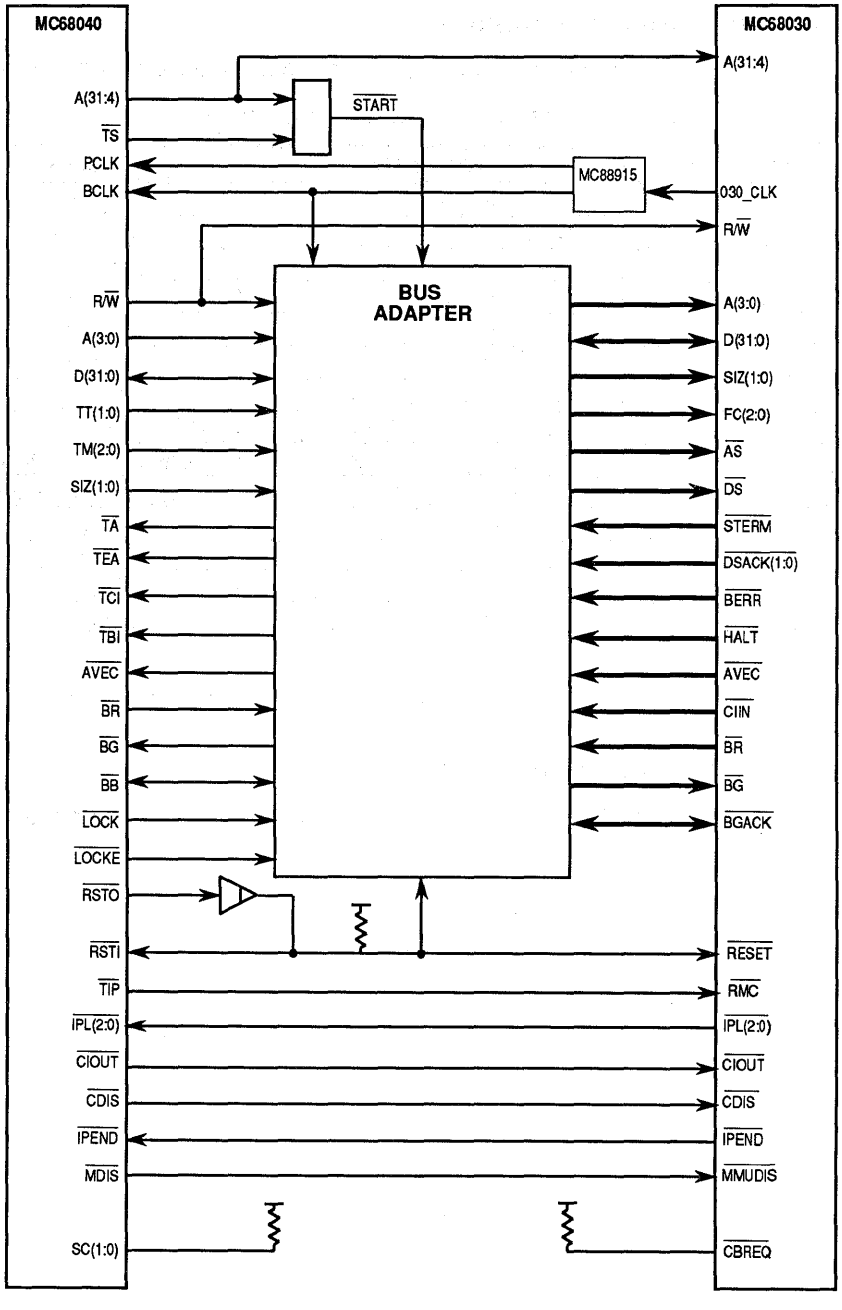


FIGURE7-3

Figure 7-3. Bus Adapter Signal Connections

7.4.1. MC68040 Signal Connections

Table 7-1 lists the MC68040 signals that are connected directly to the bus adapter.

Table 7-1. MC68040 Signal Connections

Signal Name	Description	Property
A3-A0	Lower Address	Input
D(31:0)	Data	Input/Output
TT(1:0)	Transfer Type	Input
TM(2:0)	Transfer Modifier	Input
SIZ(1:0)	Transfer Size	Input
R/$\overline{W}$	Read/Write	Input
$\overline{TA}$	Transfer Acknowledge	Output,Three-stateable
$\overline{TEA}$	Transfer Error Acknowledge.	Output,Three-stateable
$\overline{TCI}$	Transfer Cache Inhibit	Output,Three-stateable
$\overline{TBI}$	Transfer Burst Inhibit	Output,Three-stateable
$\overline{AVEC}$	Autovector	Output
$\overline{BR}$	Bus Request	Input
$\overline{BG}$	Bus Grant	Output
$\overline{BB}$	Bus Busy	Input
$\overline{LOCK}$	Bus Lock	Input
$\overline{LOCKE}$	Bus Lock End	Input

7.4.2 MC68030 Signal Connections

Table 7-2 lists the MC68030 signals that are connected directly to the bus adapter.

Table 7-2. MC68030 Signal Connections

Signal Name	Description	Property
030_A(3:0)	Address	Output
030_D(31:0)	Data	Input/Output
030_SIZ(1:0)	Size	Output
030_FC (2:0)	Function Codes	Output,Three-stateable
030_$\overline{AS}$	Address Strobe	Input/Output
030_$\overline{DS}$	Data Strobe	Output,Three-stateable
$\overline{STERM}$	Synchronous Term.	Input
$\overline{DSACK}$(1:0)	Data, Size Ack.	Input
$\overline{BERR}$	Bus Error	Input
$\overline{HALT}$	Halt	Input
030_$\overline{AVEC}$	Autovector	Input
030_$\overline{CIIN}$	Cache Inhibit In	Input
030_$\overline{BR}$	Bus Request	Input
030_$\overline{BG}$	Bus Grant	Output
030_$\overline{BGACK}$	Bus Grant Ack.	Input

$\overline{HALT}$ is never driven by the bus adapter. If the $\overline{HALT}$ signal be used to signal a double bus fault, the MC68040 processor status signals may be decoded, and then an open-collector device is used to drive $\overline{HALT}$ low when a double bus fault occurs.

7.4.3 MC68040 to MC68030 Signal Connections

Table 7-3 lists the MC68040 and MC68030 signals that are not directly used by the bus adapter but must be connected directly to each other to complete the bus interface.

Table 7-3. MC68040 to MC68030 Signal Connections

MC68040 Signals		MC68030 Signals	
Name	**Description**	**Name**	**Description**
A31–A4	Upper Address lines	A31–A4	Address lines
$\overline{TIP}$	Transfer in Progress	$\overline{RMC}$	Read/Modify/Write Cycle
$\overline{IPL}$(2:0)	Interrupt Priority Level	$\overline{IPL}$(2:0)	Interrupt Priority Level
$\overline{CIOUT}$	Cache Inhibit Out	$\overline{CIOUT}$	Cache Inhibit Out
$\overline{CDIS}$	Cache Disable	$\overline{CDIS}$	Cache Disable
R_$\overline{W}$	Read/Write	R_$\overline{W}$	Read/Write
$\overline{IPEND}$)	Interrupt Pending	$\overline{IPEND}$	Interrupt Pending
$\overline{MDIS}$	MMU Disable	$\overline{MMUDIS}$	MMU Disable

7.4.4 Miscellaneous Signals

Table 7-4 lists the remaining signals used by the bus adapter. The BCLK signal used by the bus adapter has the same phase relation with the MC68040 BCLK. The clock supplied to the MC68030 system must be the inverse of BCLK. Figure 7-4 shows the relationship between BCLK, PCLK, and the MC68030-based system clock.

The $\overline{\text{START}}$ signal signifies the start of an access that needs to be translated. This signal must be asserted one PAL setup time before the rising edge of CLK. This signal may simply be a combinational circuit which includes high-order address bits qualified by $\overline{\text{TS}}$ and $\overline{\text{TIP}}$. If the bus adapter is used in conjunction with an external cache, then this signal is asserted only when an external cache miss occurs. Figure 7-5 shows different implementations of $\overline{\text{START}}$, depending on system implementation.

Table 7-4. Miscellaneous Signals

Signal Name	Description	Property
BCLK	Clock for Adapter	Input
$\overline{\text{RESET}}$	Bus adapter RESET	Input
$\overline{\text{START}}$	Bus Adapter Chip Select	Input

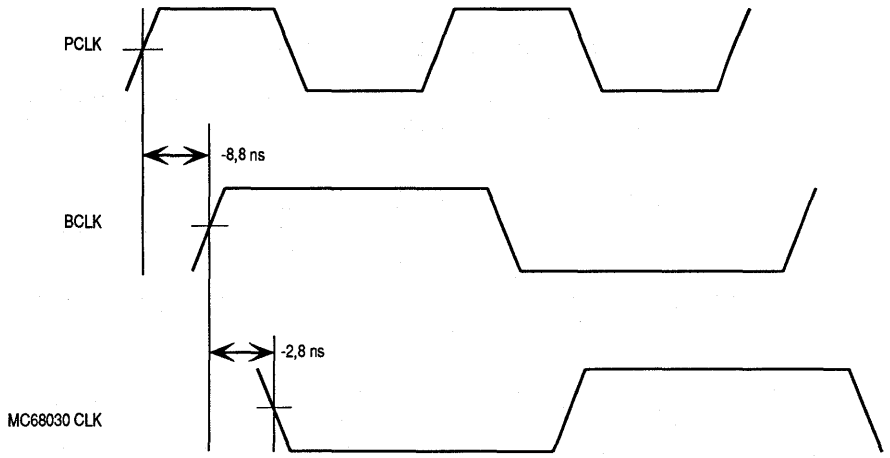


Figure 7-4. Clock Timing

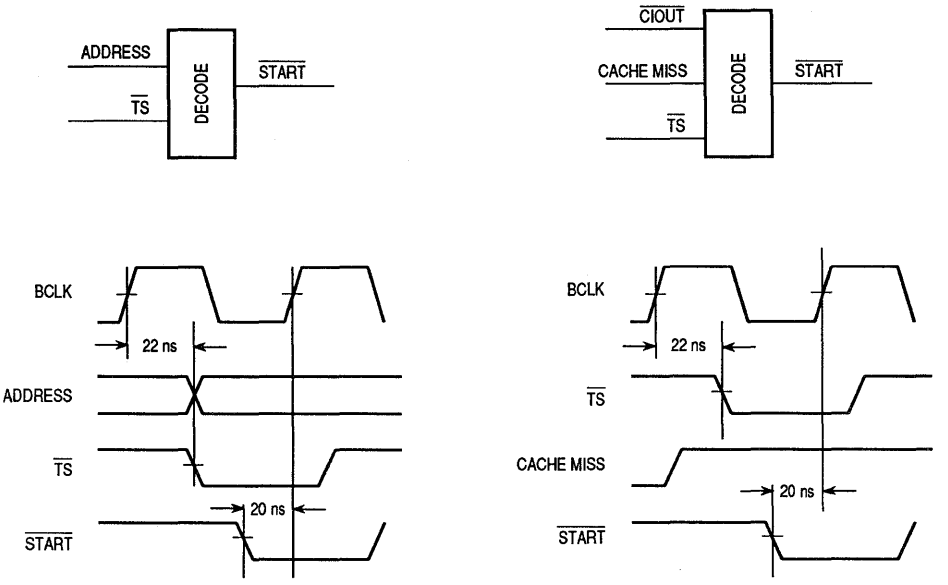


Figure 7-5. START Timing

For the bus adapter, RESET is an input-only signal. To generate the MC68030 RESET signal, the MC68040 RSTO signal must be connected to the MC68040 RSTI signal through an open collector buffer. The resulting signal has the same function of an MC68030 bidirectional RESET pin. Figure 7-3 shows the relationship between the MC68040 RSTI and RSTO and the MC68030-based RESET.

7.4.5 Unsupported Signals

The MC68030 CBREQ line must be tied high since bursting is not supported.

The following MC68030 signals are unsupported by the bus interface: OCS, ECS, DBEN,REFILL, STATUS. Designs that rely on these signals may not work properly.

7.5 BUS ADAPTER OPERATION

The following paragraphs describe the function of the bus adapter. This information is provided for designers interested in modifying the design to suit specific applications. The bus adapter consist of the main control unit, the data bus buffer unit, the internal termination generation unit, the transfer attribute translation unit, and the bus arbitration unit.

7.5.1 Main Control Unit

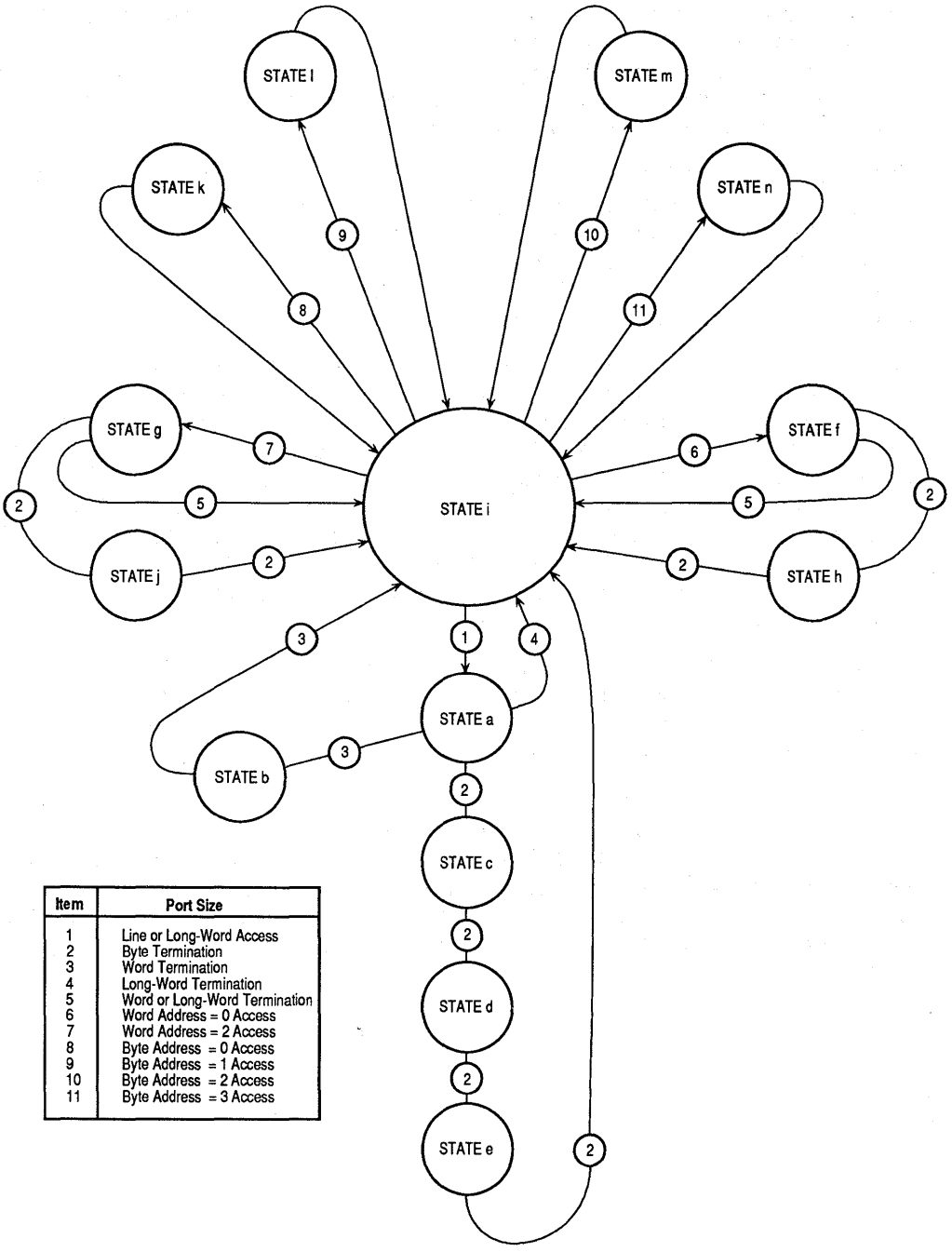
The main control unit contains four different PALs: the control PAL, the output enable PAL, the latch enable PAL, and the termination generation AS/DS PAL. The following paragraphs describe each PAL in the main control unit.

7.5.1.1 CONTROL PAL. The control PAL implements the two most important state machines in this design. Both state machines are semi-independent by nature, but are related to each other in that the output of one of the state machines is used as an input to the other. The master state machine (see Figure 7-6) is responsible for determining the bus cycle to be run at any time. Each master state represents an MC68030 bus cycle. For instance, given a long-word access to a byte port, there are four different master states (states a, c, d, and e) that are "touched" during that bus translation tenure.

Starting with a null state (state i), the master state machine waits for a clock edge in which the $\overline{\text{START}}$ input is asserted. When this signal is asserted, the master state machine goes to one of seven possible states, depending on the size and address of the transfer. When the transfer is complete as indicated by the slave state machine, the master state machine examines the type of termination encountered and sequences to the next state.

If a retry termination is detected, the master state machine remains in the same state. If a bus error termination is detected, the master state machine goes to a bus error state before going back to the idle state. Inconsistent port termination within a long-word boundary also results in a bus error, — e.g., given a long-word transfer to a byte port, once the first byte is transferred, the subsequent bytes are transferred correctly only if a byte port termination is indicated on each of the transfers. If a termination such as a word termination occurs on the second byte transfer, the bus adapter signals a bus error.

$\overline{\text{STERM}}$ termination has the same effect as long-word termination on the master state machine. Refer to **7.5.3 Internal Termination Generation** for further information on $\overline{\text{STERM}}$.



Item	Port Size
1	Line or Long-Word Access
2	Byte Termination
3	Word Termination
4	Long-Word Termination
5	Word or Long-Word Termination
6	Word Address = 0 Access
7	Word Address = 2 Access
8	Byte Address = 0 Access
9	Byte Address = 1 Access
10	Byte Address = 2 Access
11	Byte Address = 3 Access

Figure 7-6. Master State Diagram

The slave state machine sequences within the boundaries of a master state and is used internally within the main control unit. The slave state machine is responsible for sequencing states on a clock-by-clock basis. Information produced by the slave state machine is used to subdivide a master state into bus cycle phases. Figure 7-7 shows a diagram of the slave state machine.

The slave state machine starts with an idle state (S3). S3 sequences to S0 if a non idle master state is detected, which provides at least one clock between a started master state and a started slave state. The reason is that the lower address lines are a combinatory logic dependent on the master state; whereas, $\overline{AS}$ is asserted when S0 is asserted. This clock delay is used to emulate and exceed the one-half clock delay between the assertion of address and $\overline{AS}$.

When in state S0, the slave state machine waits for a valid termination. (A valid termination occurs when $\overline{BERR}$, $\overline{DSACKx}$, or $\overline{AVEC}$ is asserted for two consecutive clock cycles or, when the latched $\overline{STERM}$ ($\overline{LSTERM}$) signal is asserted for one clock cycle.) The relationship between $\overline{LSTERM}$ and $\overline{STERM}$ is that $\overline{LSTERM}$ is a half-clock delay from $\overline{STERM}$. This half clock delay is used to put $\overline{LSTERM}$ on the same clock edge as the other termination signals. When a valid termination occurs, S1 is reached. The termination information is latched on the clock edge in which S0 changes to S1, because, during that clock edge, all termination signals (including late bus errors and late retries) will be valid and stable. This information is then used to sequence from S1 to the six other states.

If a bus error termination is detected, the SB state is entered, then the S3 state. The SB state is used to generate the $\overline{TEA}$ signal to the MC68040. If a retry termination is detected, SR0 is entered. The slave state machine stays in this state until it detects a negation of $\overline{HALT}$. It then sequences to SR1, then to S3.

If a normal termination is detected during the last bus cycle of a bus translation tenure and if $\overline{HALT}$ is negated, S2 is entered, followed by S3. S2 is used to generate the $\overline{TA}$ signal to the MC68040. However, if $\overline{HALT}$ is asserted, the slave state machine proceeds to a waiting state (S2W) until $\overline{HALT}$ is negated, then continues to S2. If a normal termination is detected and it is not the last bus cycle of a bus translation tenure, the state machine proceeds S3W. After $\overline{HALT}$ is negated, S3 is entered.

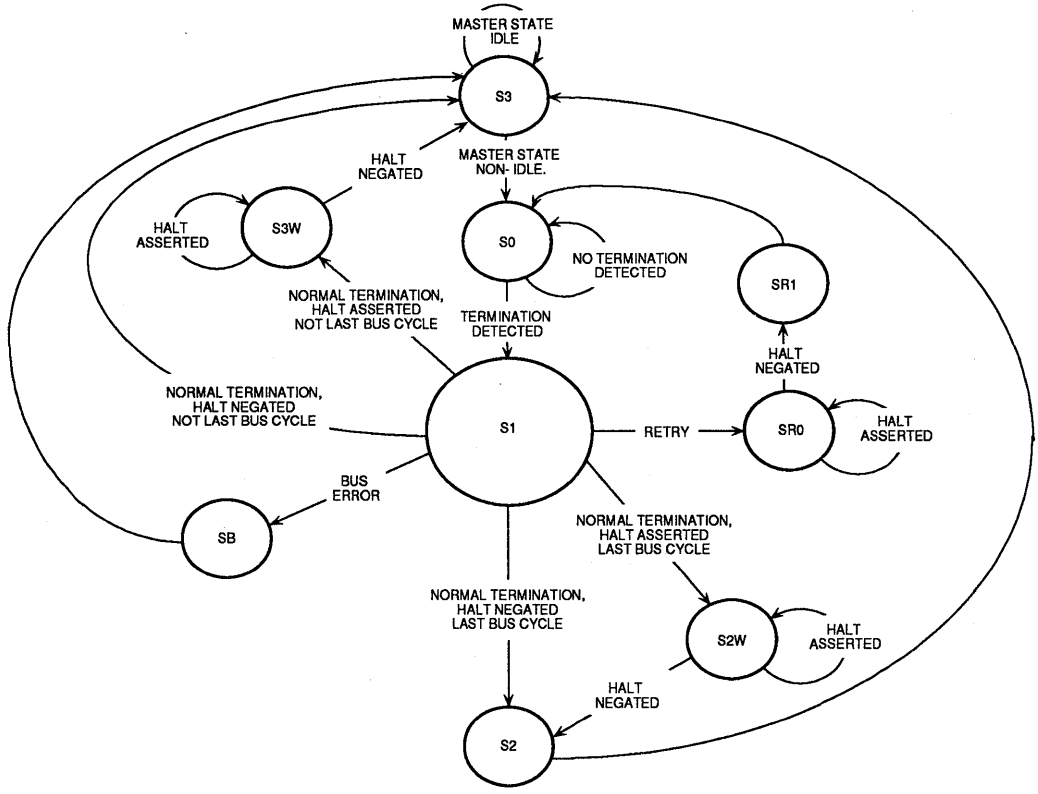


Figure 7-7. Slave State Diagram

7.5.1.2 OUTPUT ENABLE PAL. The role of the output enable (OE) PAL is to duplicate valid data onto the MC68030 side of the bus during write cycles. Once the address strobe ($\overline{AS}$) to the MC68030 side of the bus is negated, the buffers may be three-stated. The slave state (s1) is used to negate the buffers.

The equations in the OE PAL have the following format:

```
xoe := wr & ( master states) & ! s1  #
   rd & ( (termination) & s1 # xoe & s2w);
```

where

x	=	one of a, b2, b1, c2, c1, d3, d2, d1
master states	=	combinations of master states
termination	=	combination of byte, word, long word
s1	=	slave state
s2w	=	slave state
wr	=	write cycle
rd	=	read cycle

The last term "rd & *x*oe & s2w" is used to keep *x*oe asserted if $\overline{\text{HALT}}$ is asserted.

Note that on the MC68030 side of the bus (see Figure 7-8) only buffers c1 and d1 drives the lower middle (LM) and lower lower (LL) byte lanes. Therefore, to ensure that these byte lanes are driven, buffers c1 and d1 are always driven during write cycles. The upper upper (UU) and upper middle (UM) byte lanes, on the other hand, have multiple buffers contending for these byte lanes.

For example, given a long-word write to a byte port, the bus adapter does four bus cycles with the master state sequencing through A-C-D-E. During the first bus cycle, buffer a drives the UU byte lane, and on the next bus cycle, buffer b2 drives the UU byte lane. During the third bus cycle, the UU byte lane is driven by buffer c2, and on the final bus cycle, buffer d3 drives the UU byte lane.

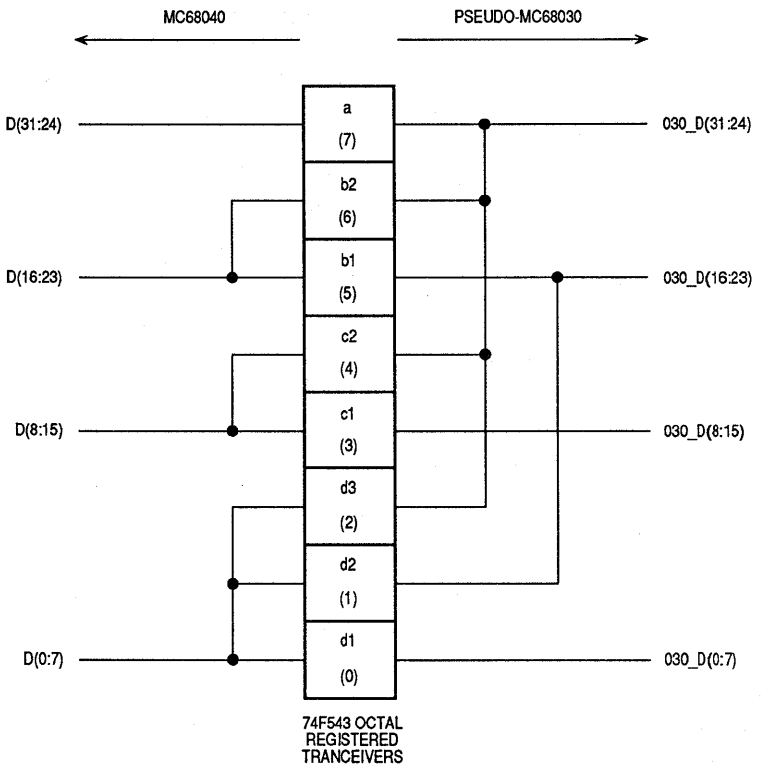


Figure 7-8. Data Bus Connections

Table 7-5 shows the assignment of buffers as a function of the master state. This explains the term "wr & (*master states*) # ! s1."

Table 7-5. Buffer Assignments

Master State	Buffers Enabled
A	a,b1,c1,d1
B	c2, d2, c1, d1
C	b2, d2, c1, d1
D	c2, d2, c1, d1
E	d3, d2, c1, d1
F	a, b1, c1, d1
G	c2, d2, c1, d1
H	b2, b1, c1, d1
J	d3, d2, c1, d1
K	a, b1, c1, d1
L	b2, b1, c1, d1
M	b1, c2, c1, d1
N	d3, d2, c1, d1

During read cycles, the OE PAL is responsible for generating the correct byte lanes only on the final bus cycle of a translation sequence. For example, a long word to a byte port has the master state sequence A-C-D-E. In this case, only on the final bus cycle, E, do the buffers need to present data to the MC68040. The OE generation cannot depend solely on the master state during read cycles. Consider a byte read from address 3. In this case, the MC68040 LL byte lane must contain the data. However, depending on the byte port returned by the system, the data could be supplied to any of three different buffers. This requires that the OE generation be a function of the slave state as well. Slave state (S1) is used to assert the OE generation during read cycles. During this slave state, the internal termination signals are stable. Figure 7-9 shows the relative timing of the slave states and the OE bus ($\overline{\text{OEBUS}}$) signals, which explains the existence of s1 in the term "rd & (termination) & s1."

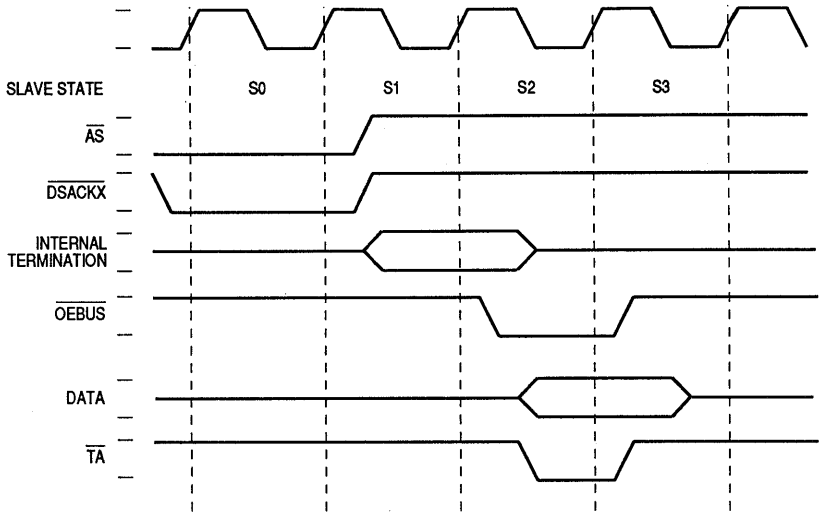


Figure 7-9. OEBUS Timing

Table 7-6 shows a summary of the master state sequences for read cycles as a function of requested operand and byte port responses. As previously mentioned, only the last box of each master state sequence is needed to supply data to the MC68040.

The UU byte can only be driven by buffer a; therefore, buffer a may be driven at all times during read cycles.

The UM byte can be driven by either b2 or b1. Note that b2 is used in the columns entitled "Long Response" and "Word Response", but never in "Byte Response" column. This makes it possible to divide the buffer ownership solely as a function of termination.

The LM byte can be driven by either c2 or c1. Note that c2 appears only under the "Byte" and "Word Response" columns, whereas, c1 appears only in the "Long Response" column. Again, buffer ownership is divided as a function of termination.

The LL byte can be driven by either d3, d2, or d1. Note that d1 appears only under "Long Response"; d2, only under the "Word Response"; d3 under the "Byte Response". Again, buffer ownership is divided as a function of termination.

Table 7-6. Read Cycle Sequence

	Long Response		Word Response		Byte Response			
Long Word		A	A	B	A	C	D	E
	rearm	a,b1,c1,d1	a,b1	c2,d2	a	b2	c2	d3
	maintain		a,b1			a	a,b2	a,b2,c2
	Output enable	a,b1,c1,d1	a,b1,c2,d2					a,b2,c2,d3
Word, Address 0		F	F		F	H		
	rearm	a,b1	a,b1		a	b2		
	maintain				a			
	Output enable	a,b1	a,b1		a,b2			
Word, Address 2		G	G		G	J		
	rearm	c1,d1	c2,d2		c2	d3		
	maintain				c2			
	Output enable	c1,d1	c2,d2		c2,d3			
Byte, Address 0		K	K		K			
	rearm	a	a		a			
	Output enable	a	a		a			
Byte, Address 1		L	L		L			
	rearm	b1	b1		b2			
	Output enable	b1	b1		b2			
Byte, Address 2		M	M		M			
	rearm	c1	c2		c2			
	Output enable	c1	c2		c2			
Byte, Address 3		N	N		N			
	rearm	d1	d2		d3			
	Output enable	d1	d2		d3			

NOTES:

1. Rearm — This term has a net effect of releasing previously latched data and is also responsible for ensuring that the buffers needed to latch data are ready to latch data prior to end of the bus cycle.
2. Maintain — This term is used to prevent data from being lost during the intermediate master states.

In all these cases, the master states are not involved in the generation of OE during read cycles. Hence, *termination* exists in the term: "rd & (*termination*) & s1."

7.5.1.3 LATCH ENABLE PAL. The latch enable (LE) PAL is used during read to indicate to the transceivers when to latch data and when to release the data stored in the transceivers.

The PAL equations generated have the following form:

```
                                    xle := (at # st) #
                                    xle & !neg_le & (rearm ) #
                                        (maintain );
```

where

x	=	one of a, b2, b1, c2, c1, d3, d2, d1
rearm	=	combinations of master states
maintain	=	combinations of master states
neg_le	=	slave states s3 or sr1
at	=	asynchronous termination
st	=	synchronous termination

The term "(at st)" is asserted when a valid termination is detected. All of the LE signals will assert on the following clock transitions. The net effect is that all LE signals that were previously negated will latch the data on their respective byte lanes. If a certain LE signal were previously asserted, no change in state would occur, making the register keep previously latched data. Once data is latched, it remains latched until the slave state s3 or sr1 is detected.

Refer to Table 7-6 for the following example. For example, a long word read from a byte port that has the sequence A-C-D-E. During master state A, the first byte data would be latched by buffer a. Once this is done, the byte must be maintained throughout master states C,D, and E. During master state C, the buffer b2 is rearmed, and when it is asserted (by the "at#st" term), it is maintained in the master state D and E. If a retry occurs, master state C is rerun and rearmed before its reassertion. As long as the master state is C, buffer a remains asserted, regardless of the number of retries that occur to redo the bus cycle C. Table 7-7 reorganizes the rearm and maintain terms so that they can be used to generate the PAL equations. Close examination of the PAL equations shows that there are some extra rearm and/or maintain terms attributed to each buffer. These extra terms help in the reduction of PAL terms.

Table 7-7. Latch Enable Generation Summary

Buffer	Rearm	Maintain
a	A,F,K	B, C, D, E, H
b2	C, H, L	D, E
b1	A, F, L	B
c2	B, D, G, M	E, J
c1	A, G, M	
d3	E, J, N	
d2	B, G, N	
d1	A, N, G	

7.5.1.4 Termination Generation, $\overline{AS}$/ $\overline{DS}$ PAL.; This PAL decodes the slave state machine and the termination signals generated by the internal termination generation unit to produce the MC68040 termination signals.

The $\overline{TA}$ is basically a decode of the slave state s2. The $\overline{TEA}$ is a decode of slave state sb. The $\overline{TBI}$ is a combination of slave state s2 whenever a line transfer is indicated, which has the effect of forcing fake-burst transfers.

The TCI and AVEC terms are different because the generation unit contains an internal output. The internal outputs have the function of extending the input terms, $\overline{RCIIN}$ and $\overline{RAVEC}$, so that they are iguaranteed to be present when the slave state s2 is reached.

The MC68030 $\overline{AS}$ and $\overline{DS}$ signals are also generated in this PAL. When a no-wait-state write cycle occurs, $\overline{DS}$ is not asserted, as in the MC68030.

7.5.2 Data Bus Buffers

The data bus buffer unit is used to multiplex the data lanes and to store temporary data. This buffer consists of eight 74F543 registered transceivers (see Figure 7-8).

The R_$\overline{W}$ signal is used to determine the direction of the transfers. All eight CEAB signals are connected together and are connected to inverse of the R_$\overline{W}$ signal. Refer to **7.6 SCHEMATICS and PARTS** for connections.

All eight of the LEAB signals are always negated since no latching of data is done during write cycles.

Each transceivers has its OEAB and OEBA connected for cost-reduction purposes. Eight OE signals are bussed together to form $\overline{OEBUS7:0}$, which is controlled by the master control unit.

The LEBA signal on each of the eight transceivers are bussed together to form the LEBUS(7:0). This signal is controlled by the main control unit.

7.5.3 Internal Termination Generation

Figure 7-10 shows a summary of the relative timing between all of the internal termination signals, slave states, and BCLK. $\overline{RBERR}$ and $\overline{RHALT}$ are simply clocked versions of the $\overline{BERR}$ and $\overline{HALT}$ lines, respectively.

The $\overline{AVEC}$ signal from the MC68030 system is gated with $\overline{IACK}$ (from the bus adapter) to prevent the bus adapter from incorrectly recognizing $\overline{AVEC}$ as a valid termination signal on noninterrupt acknowledge cycles. The resulting signal is then gated to generate $\overline{RAVEC}$. Gating of the MC68030-generated $\overline{AVEC}$ with $\overline{IACK}$ is unnecessary if the MC68030-supplied $\overline{AVEC}$ is asserted only during interrupt acknowledge cycles and not grounded (some designs that solely use autovectoring may have this signal grounded).

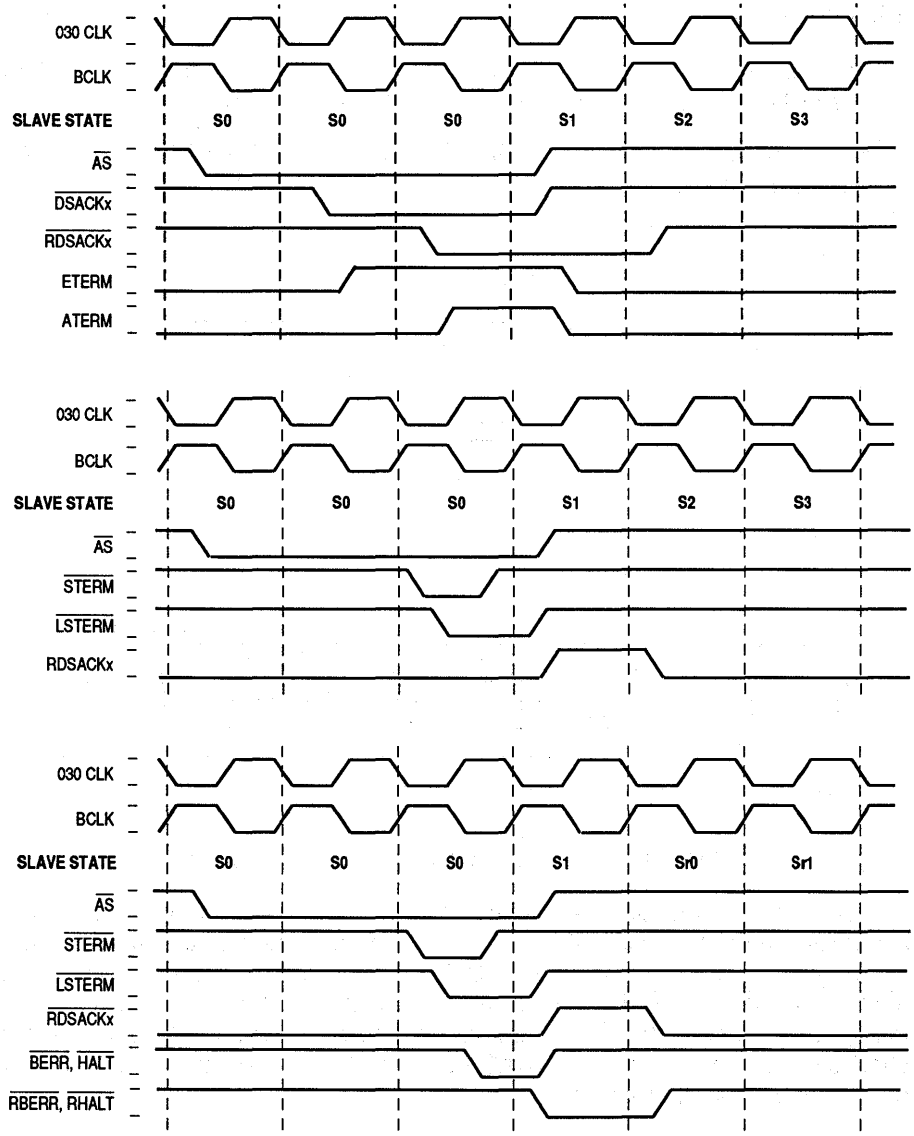


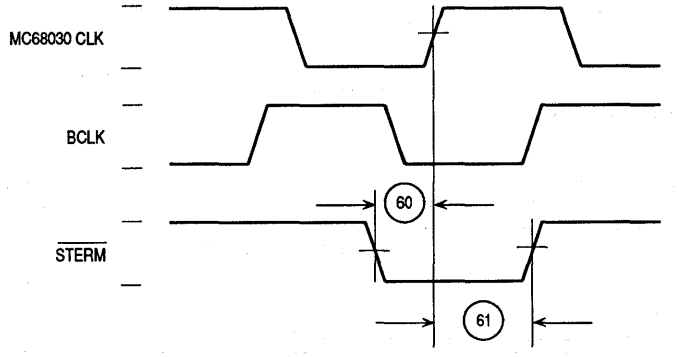
Figure 7-10. Internal Termination Signal Timing

RDSACK0 contains two terms: a clocked $\overline{\text{DSACK0}}$ and a clocked $\overline{\text{LSTERM}}$. RDSACK1 also contains two terms: a clocked $\overline{\text{DSACK1}}$ and a clocked $\overline{\text{LSTERM}}$.

ETERM is asserted whenever any of these lines are asserted: $\overline{BERR}$, $\overline{DSACK0}$, $\overline{DSACK1}$, $\overline{AVEC}$.

ATERM is asserted whenever any of the following signals are asserted with $\overline{ETERM}$ asserted and $\overline{AS}$ asserted: $\overline{RBERR}$, $\overline{RDSACK0}$, $\overline{RDSACK1}$, $\overline{RAVEC}$.

$\overline{LSTERM}$ is a latched version of $\overline{STERM}$. This latching of $\overline{STERM}$ may be deleted if the timing shown on Figure 7-11 is met.



MC68030 SPECIFICATION		20 MHz		Required by Bus Adapter 16.67 MHz		20 MHz	
		MIN	MAX	MIN	MAX	MIN	MAX
60	SYNCHRONOUS SETUP TIME	4	-	0	-	0	-
61	SYNCHRONOUS HOLD TIME	12	-	32	-	27	-

Note: If the required timing specification is met for $\overline{STERM}$, latched $\overline{STERM}$ ($\overline{LSTERM}$) is not needed.

Figure 7-11. $\overline{STERM}$ Timing Requirements

When running the bus adapter in an asynchronous environment, the issue of metastability forces a consideration of the maximum frequency that the bus adapter can operate. This design uses 74AS D flip-flops to sample asynchronous inputs $\overline{DSACK0}$, $\overline{DSACK1}$, $\overline{BERR}$, $\overline{HALT}$, $\overline{030\_AVEC}$. It is important that the registered output signals not be used until the metastable outputs are resolved. Given the data supplied by Texas Instruments® 74AS data sheet, a rule-of-thumb 25 ns may be used as a clock-to-output propagation delay.

Given that the clock to output delay for the termination signals RDSACK0, RDSACK1, $\overline{RBERR}$, $\overline{RHALT}$ and $\overline{RAVEC}$, the 25-MHz speed path is defined by the clock-to-ATERM valid, the clock setup time of U2 (MC68040 Termination, $\overline{AS}$/$\overline{DS}$ Generation PAL), and the clock period by the equation:

clock-to-ATERM	+	PAL clock setup	<	clock period
(25 + 7)	+	4.5	<	40

This gives a 3.5 ns margin on a 25 MHz design.

7.5.4 Transfer Attribute Translation

This unit is responsible for generating the lower address lines and size encoding to the MC68030 system. It is also responsible for creating the function codes to the MC68030 system.

A PAL is used to generate the two lower address bits and size encoding for the MC68030 system. The address bits and size encoding are combinatory logic dependent solely on the master state. The logic required in generating these signals is independent of the transfer type of the MC68040. The PAL equations are in the form of a truth table, which simply maps the master state with the appropriate address and size encoding. For the master states I, Z, and DC, the lower addresses and sizes are "don't cares" and are assigned the values equal to what the MC68040 produces.

The system designer must note that the function code translation is merely an approximation of what an actual MC68030 would do. When data is stored in the MC68040 caches, the function code encoding is not stored. Therefore, during cache pushes, the bus adapter assigns them as supervisor accesses.

Another issue relates to the difference in the way PC-relative and MOVES accesses into program space is treated. In the MC68030, these accesses to instruction space reference the data cache, although the function codes still indicate accesses to instruction space. On the other hand, the MC68040 goes further and translates the same instruction space encoding into a data space encoding before reaching the memory management unit, which results in the transfer modifier (TM) indicating a data space reference to these special accesses.

During interrupt acknowledge cycles, the lower address lines must reflect the level of interrupt being serviced. The approach taken in this design is to use an external multiplexer to choose between the MC68040 TM(2:0) lines and the lower address lines. The $\overline{\text{IACK}}$ signal is used to control the external multiplexer. This signal is also used by the internal termination generation.

This implementation of the PAL uses a large 22V10 PAL. If the function codes supplied to the MC68030-based system are used solely to define an interrupt acknowledge cycle, a smaller 16L8 PAL may be used. The simplest way to do this would be to disconnect the TM(2:0) and FC(2:0) from the PAL. This decreases the pin count by six, which is exactly the number needed for the logic to fit into a 16L8 PAL. The only thing left to do would be to run the $\overline{\text{IACK}}$ into an inverting buffer and use this output to drive the FC(2:0) lines.

7.5.5 Bus Arbitration

The bus arbiter unit of the bus adapter performs arbitration between the MC68040 and the MC68030 system. The arbiter starts in an idle state in which neither master has the bus. The bus is given to the first synchronized bus request that is detected.

In case of a tie, the MC68030 system gains control of the bus. Once the MC68030 system gains control of the bus, all MC68040 requests are ignored until the bus

arbiter returns to the idle state. The state diagram that the bus arbiter uses is identical to that found in the MC68020.

If the MC68030 does not request the bus and if the MC68040 asserts its bus request, a bus grant is issued to the MC68040. The bus grant remains asserted as long as the MC68040 maintains the assertion of its $\overline{BR}$ and if no MC68030 system requests for the bus. Otherwise, the MC68040 bus grant negates, and the bus arbiter will sequence toward the idle state. The idle state is not reached until the MC68040 bus busy signal indicates that the MC68040 no longer has the bus. This cycle repeats itself. Figure 7-12 shows the state diagram which governs the bus arbiter unit. Note that the terminology given in the state diagram indicates that signals with an overbar represent a negated state, regardless of whether the actual pin is a logic-high or logic-low signal. The lower portion of the state diagram is identical to that found in an MC68030.

The MC68040 $\overline{TIP}$ signal can be used as the simulated MC68030 bus RMC in that if the $\overline{030\_BR}$ is asserted, as long as $\overline{TIP}$ is asserted, the $\overline{030\_BG}$ is not asserted to the MC68030 system.

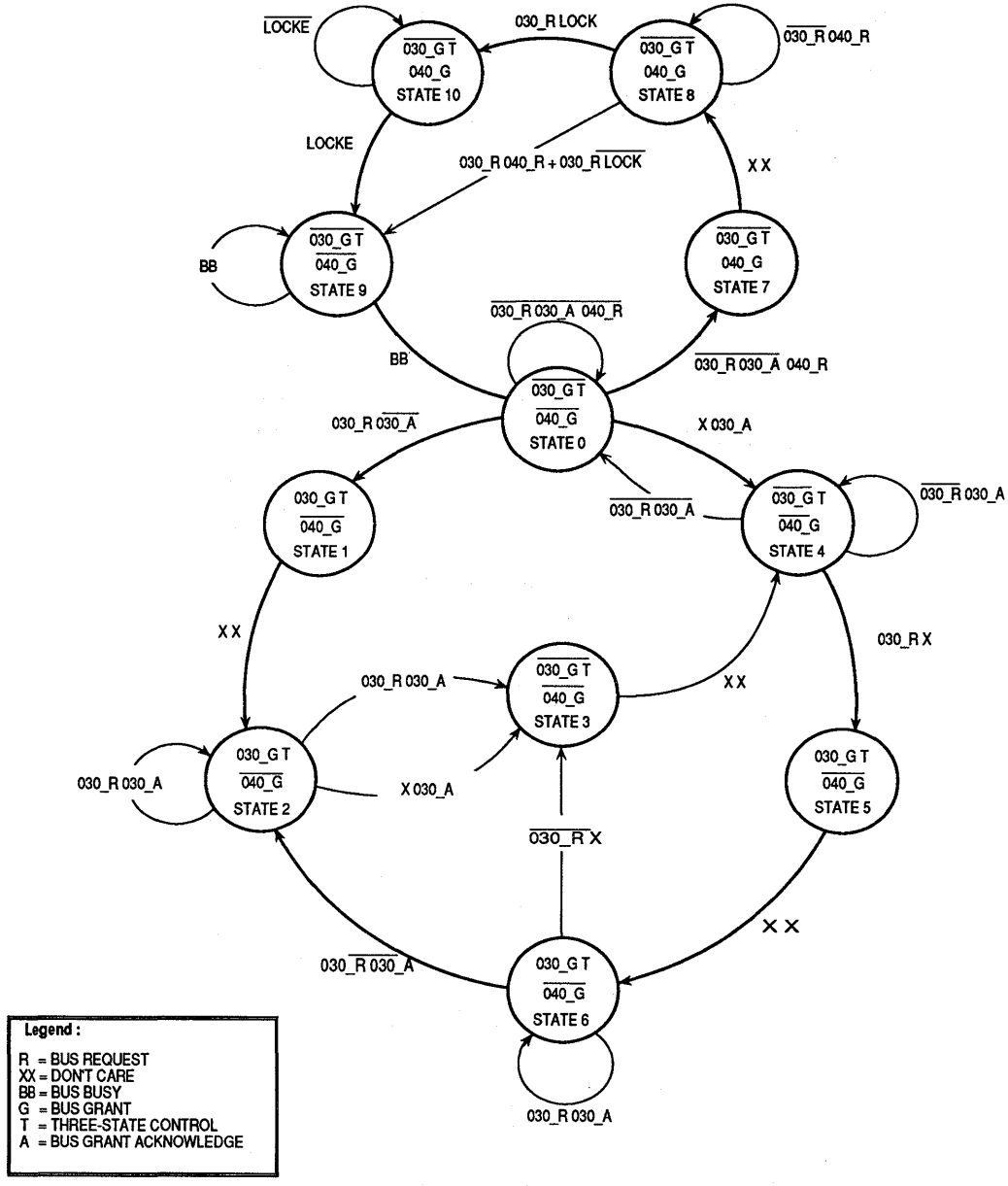


Figure 7-12. Bus Arbitration State Diagram

7.6 SCHEMATICS AND PARTS

Table 7-8 lists the timing for each PAL at 16.67 MHz, 20 MHz, and 25 MHz. Table 7-9 lists different speed grade PALs to meet the MC68030 timing specifications. The speed grade of a PAL is determined by the speed path equations as shown in Table 7-10.

Table 7-8. PAL Speed Grades (tpd)

PAL Function		PAL Type	16.67 MHz	20 MHz	25 MHz
U1	Control	22V10	25 ns	20 ns	12 ns
U2	040 Termination, AS, DS Generation	22V10	25 ns	20 ns	12 ns
U3	$\overline{\text{OEBUS}}$ Generation	20R8	30 ns	25 ns	15 ns
U4	LEBUS	16R8	15 ns	12 ns	7.5 ns
U5	Transfer Attribute Translation	22V10	20 ns	20 ns	15 ns
U6	Bus Arbitration	16R6	30 ns	25 ns	12 ns

Table 7-9. Suggested PALs

Vendor	Part Number	tpd	tco	tsu
AMD	AMPAL22V10A	25	15	20
TI	TIBPAL22VP10-20C	20	12	15
Gazelle	GA22V10-12	12	9.5	4.5
TI	PAL20R8AC	25	15	25
TI	TIBPAL20R8-15C	15	12	15
TI	TIBPAL16R8-12C	12	10	10
Signetics	PLUS16R8D	10	7.5	9
Signetics	PLUS16R8-7	7.5	6.5	7

Table 7-10. Speed Path Equations

U1(tsu) < cycle time – 25 – 5
U2(tsu) < cycle time – 25 – 7
U3(tsu) < cycle time – 25 – 5
U4(tsu) < cycle time – 25 – 7
U5(tsu) < cycle time – 25 – 5
U6(tsu) < cycle time – 25 – 5
U1(tco) + U5(tpd) < cycle time – spec 11 – 8
U2(tco) < spec 9
U3(tco) < cycle time – spec 26 – 10
U4(tco) < spec 47b – 1

Figures 7-13—7-19 show the schematics for the bus adapter. To completely replace the MC68040 Bus Adpater design into an existing MC68030 based system,

the system will need to be synchronized to the system's clock input. The MC88195 performs this function and meets the specifications required by the MC68040 clock inputs. Figure 7-13 (2 of 2) shows a design concept on how a 25 MHz design may be implented. The design has been provided as an example and has not been tested.

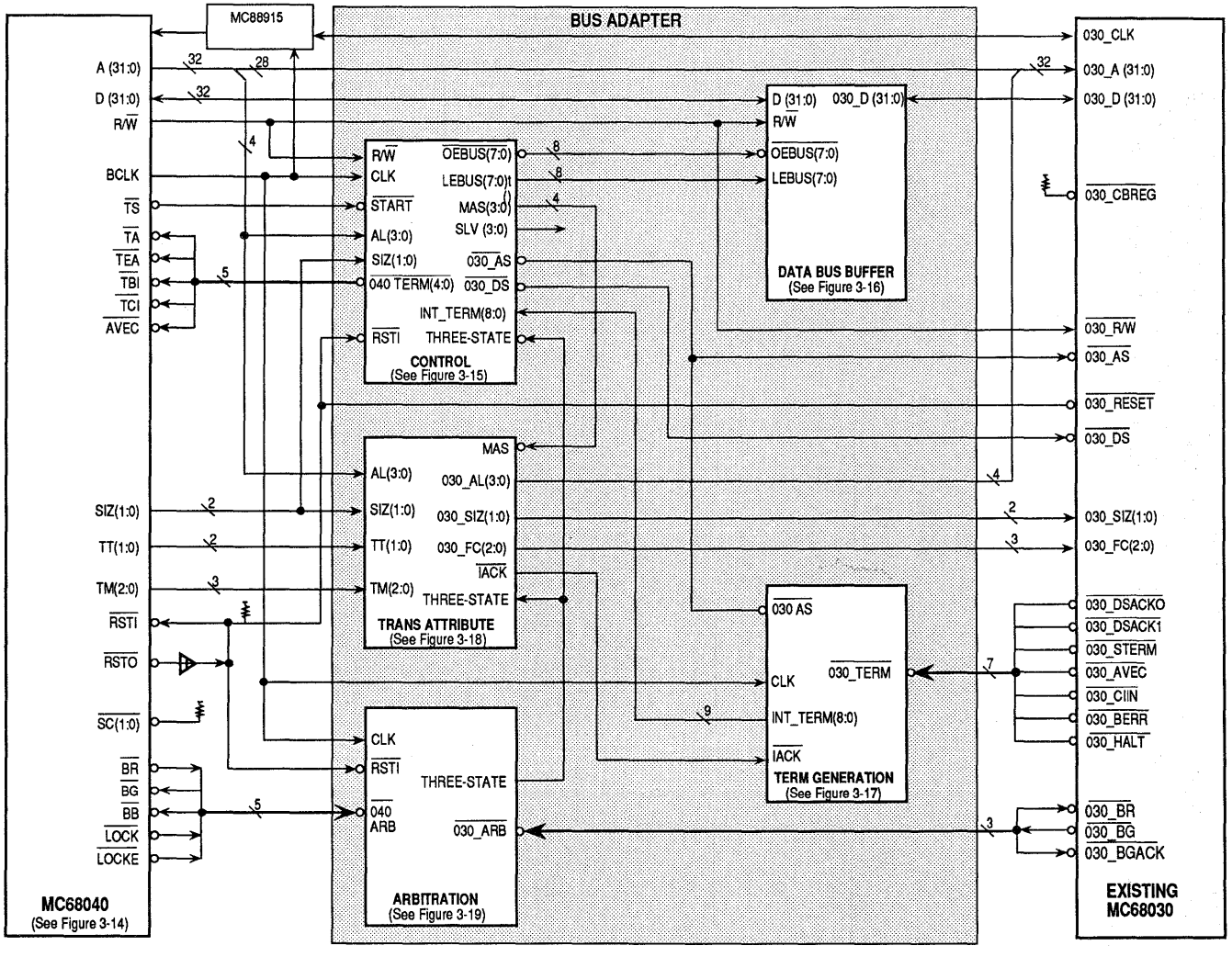


Figure 7-13. Bus Adapter (1 of 2)

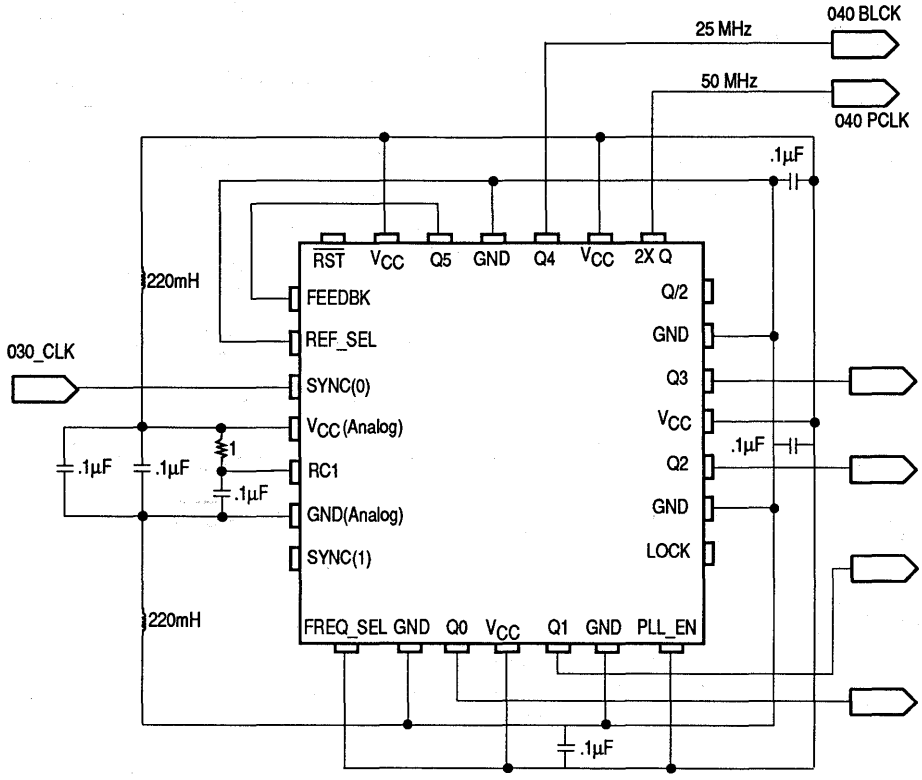


Figure 7-13. Bus Adapter (2 of 2)

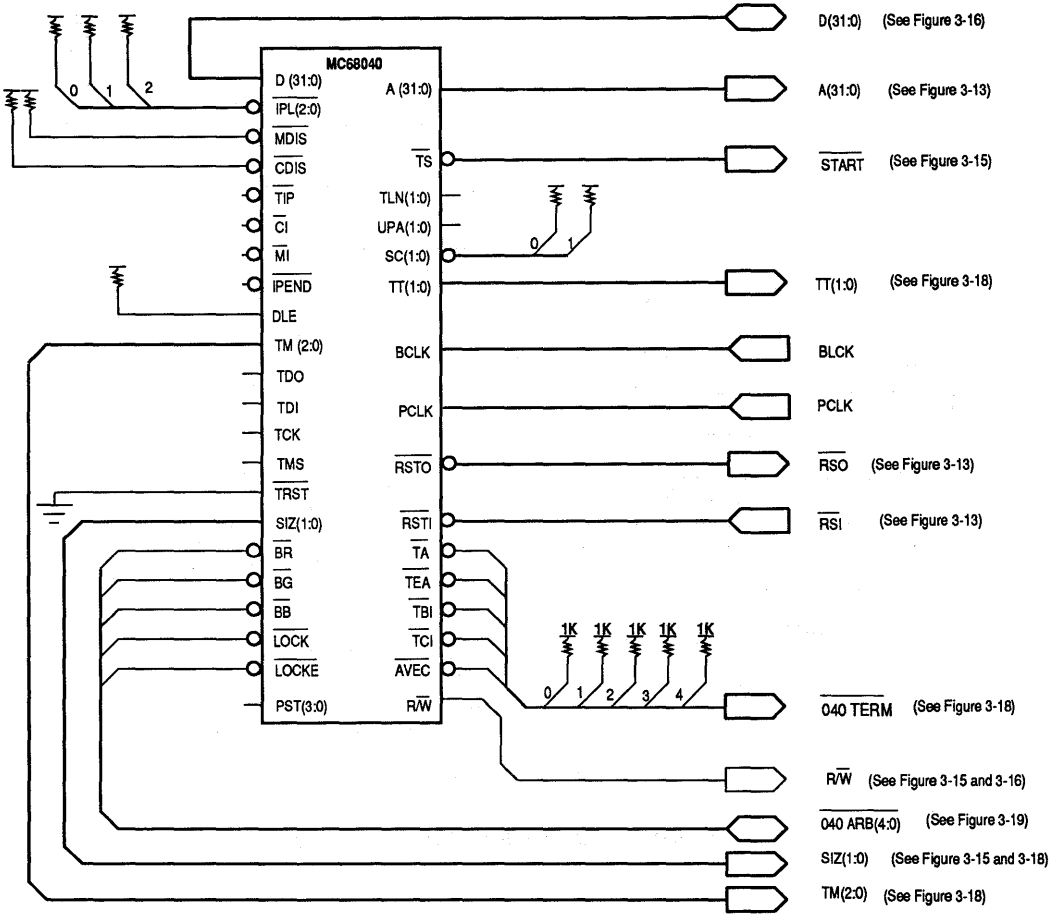


Figure 7-14 MC68040 Processor Connections

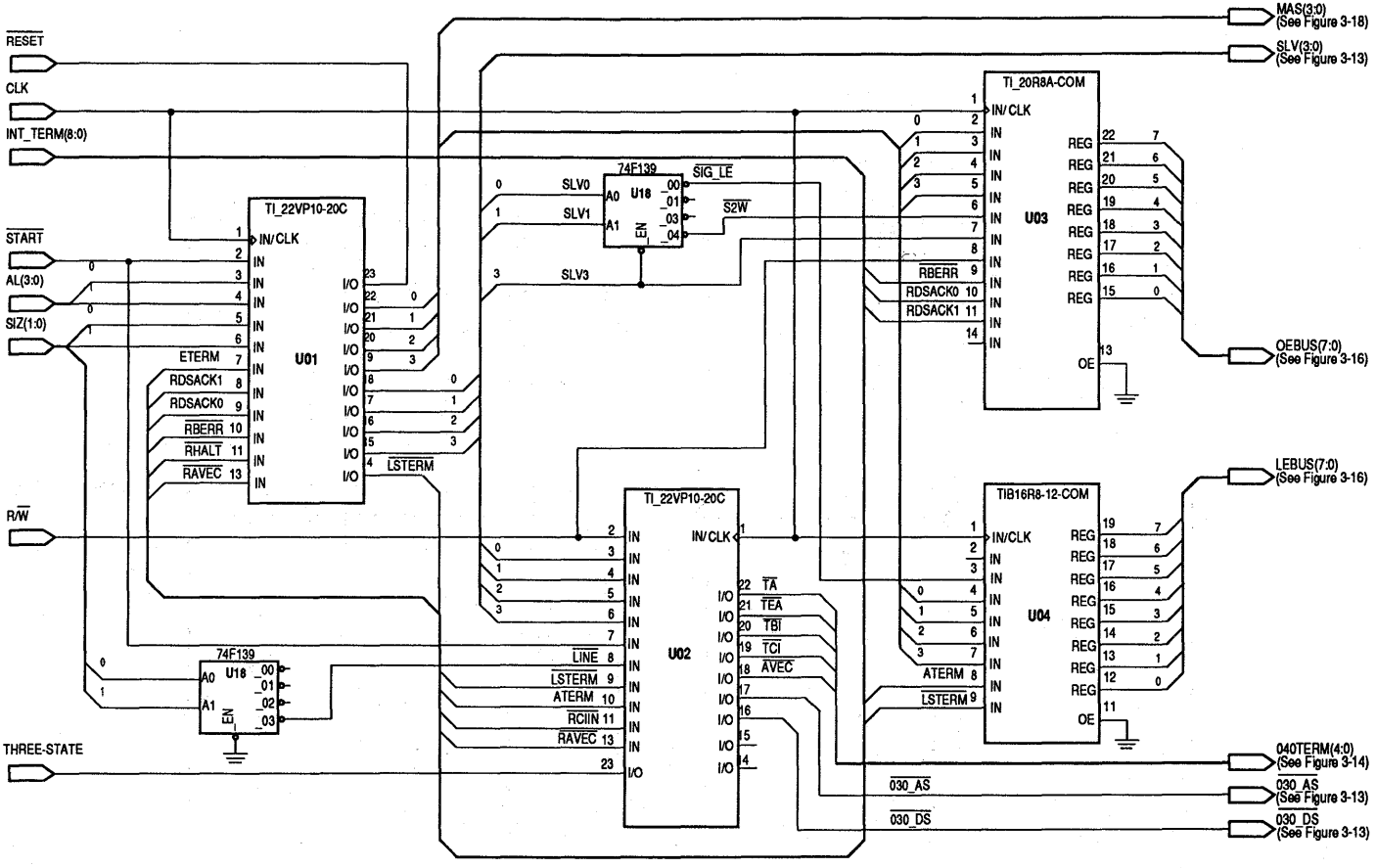


Figure 7-15 Control Unit

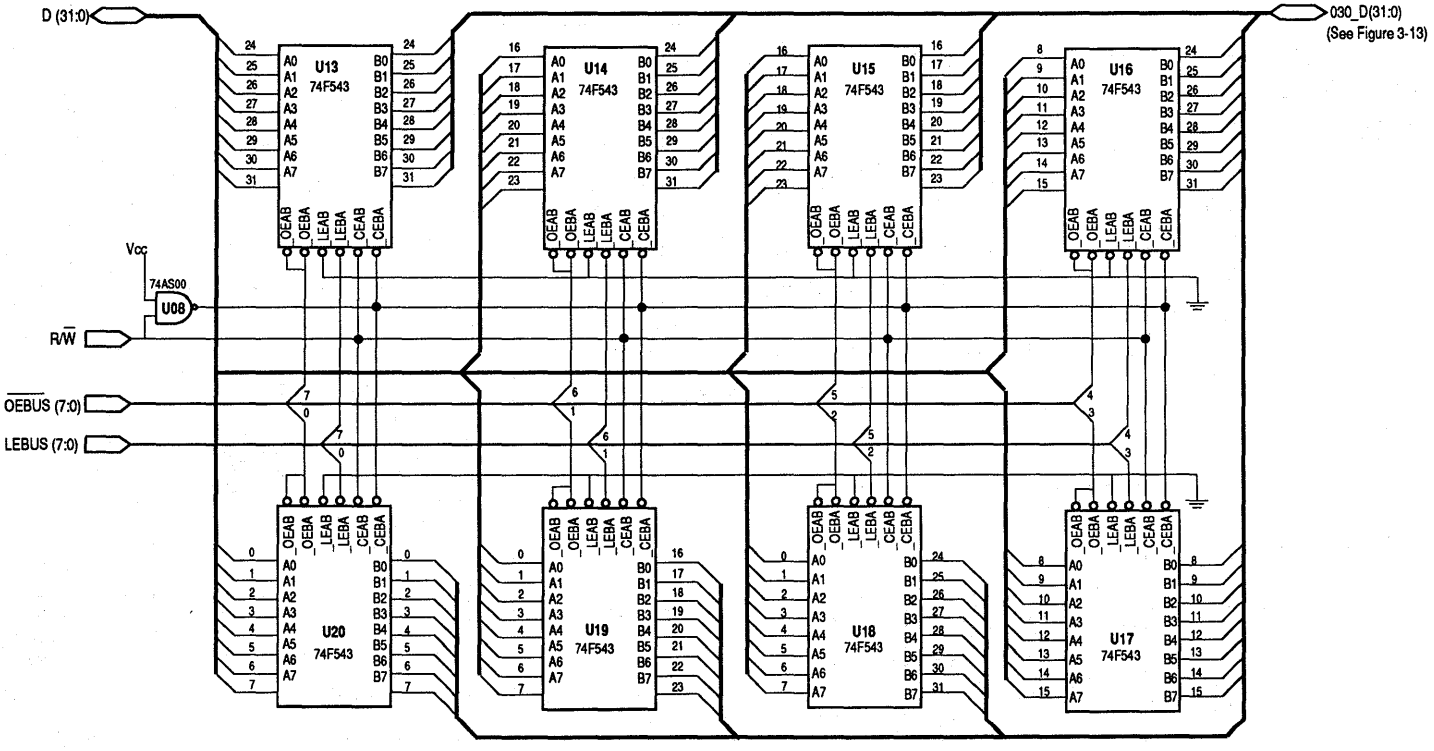


Figure 7-16. Data Bus Buffer Unit

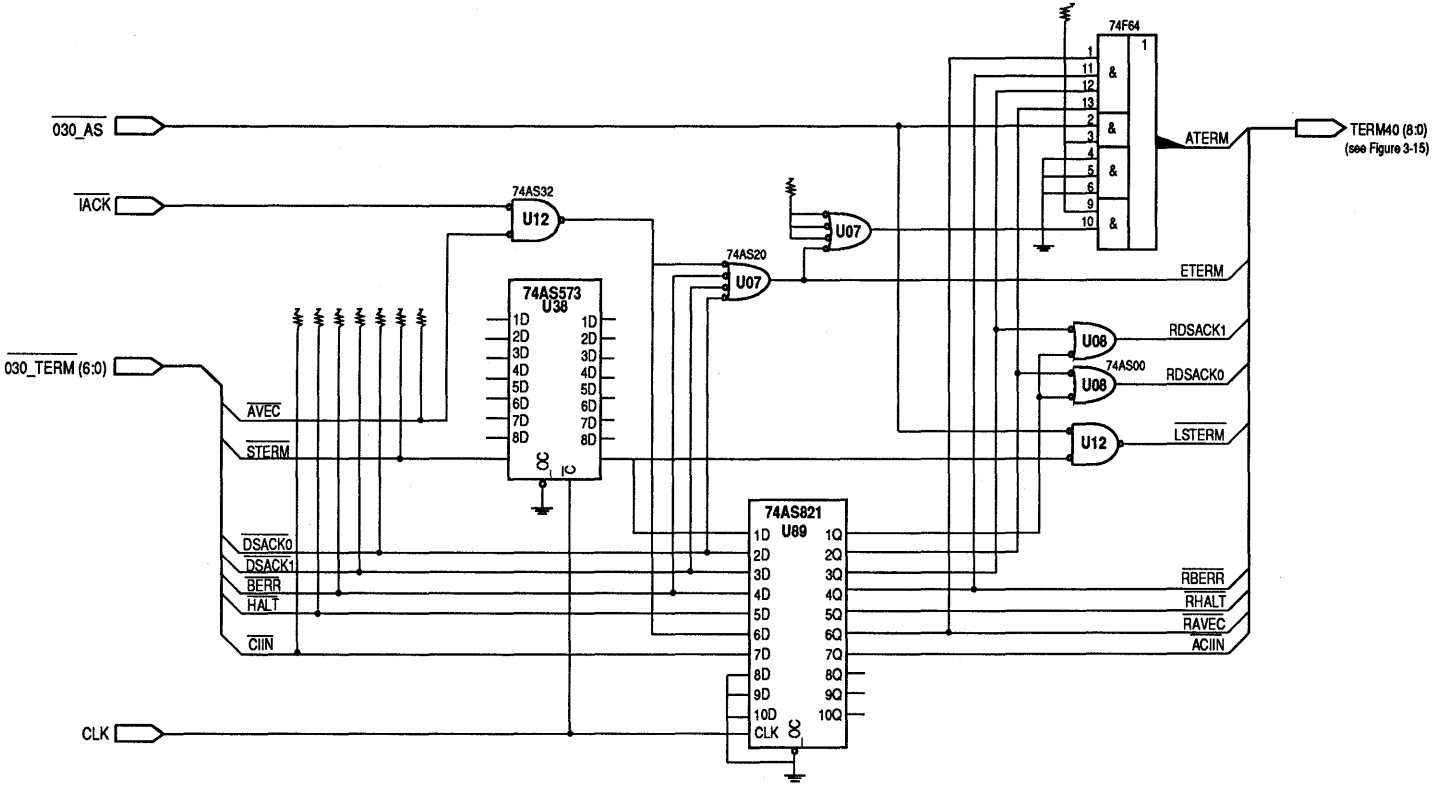


Figure 7-17. Internal Termination Generation

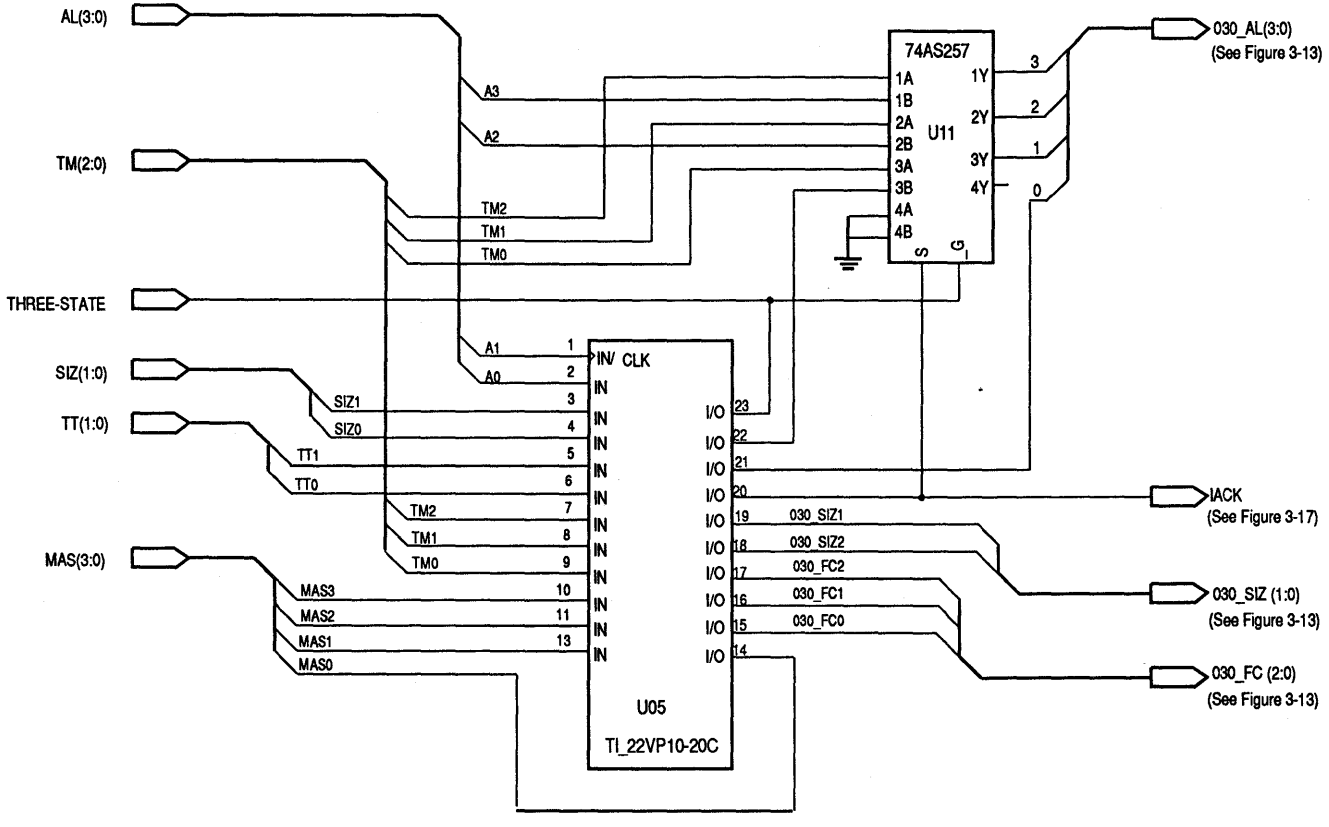


Figure 7-18. Transfer Attribute Translation

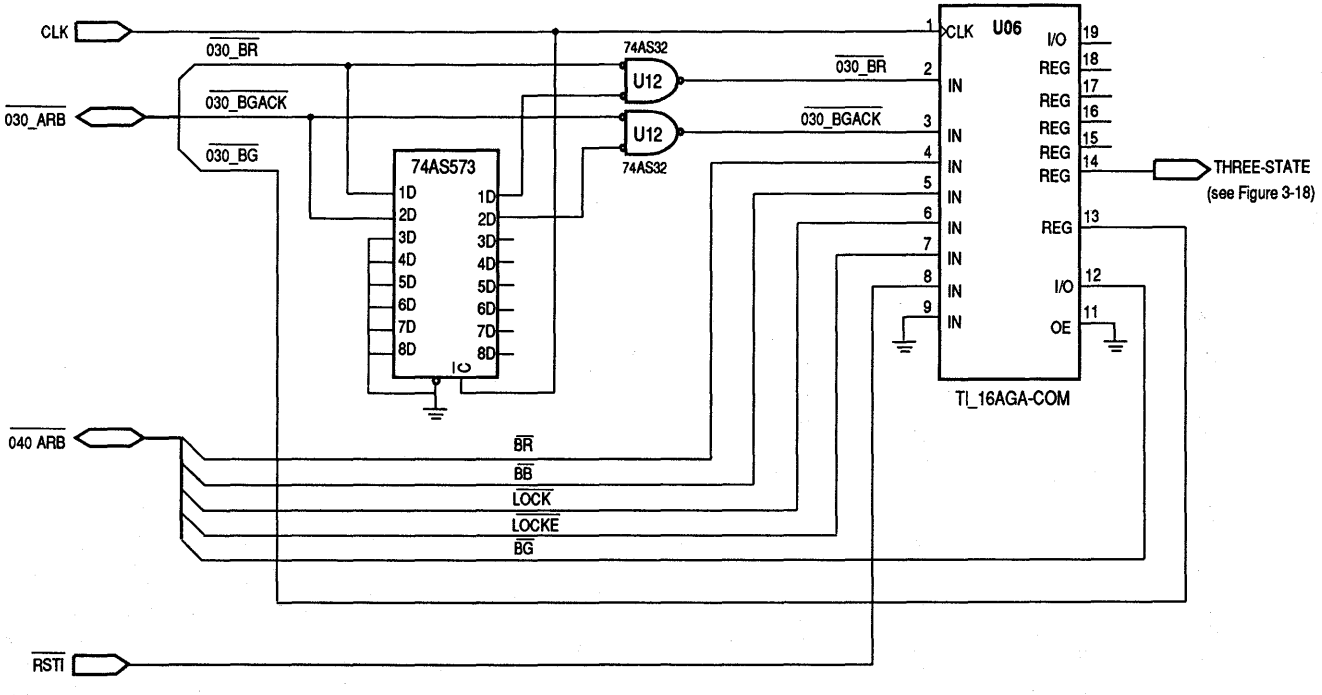


Figure 7-19. Bus Arbritation

7.7 PAL CODING

The following paragraphs provide the code listing for the PALs used in the bus adapter.

7.7.1 MC68040 Bus Translator Control PAL

```
module CONTROL flag '-r3';
title '68040 Bus-Translator Control PAL'

control device 'p22vp10';

" Inputs
bclk                                                    pin 1;
!ts,a0,a1,siz0,siz1                                  pin 2,3,4,5,6;
term, rdsack1, rdsack0,!rberr,!rhalt          pin 7,8,9,10,11;
!ravec,!lsterm                                        pin 13,14;
!reset                                                   pin 23;

" Outputs
mast0,mast1,mast2,mast3                    pin 22,21,20,19;
slave3,slave2,slave1,slave0                pin 15,16,17,18;

mast0,mast1,mast2,mast3 istype 'pos, reg, feed_reg';
slave0,slave1,slave2,slave3 istype 'pos, reg, feed_reg';


" Set assignments for master and slave state machines
mreg  = [mast3,mast2,mast1,mast0];
sreg  = [slave3,slave2,slave1,slave0];
ADDR  = [a1,a0];
SIZE  = [siz1,siz0];
DSACK = [rdsack1,rdsack0];
sreg_sub = [slave2,slave1,slave0]; "subset of sreg

" Constant assignments
x,ck = .x.,.c.;


" Master State Values
I = ^b0000; i = mreg == [0,0,0,0];
A = ^b0001; a = mreg == [0,0,0,1];
B = ^b0101; b = mreg == [0,1,0,1];
C = ^b1101; c = mreg == [1,1,0,1];
D = ^b0100; d = mreg == [0,1,0,0];
E = ^b1100; e = mreg == [1,1,0,0];
F = ^b1001; f = mreg == [1,0,0,1];
G = ^b1011; g = mreg == [1,0,1,1];
H = ^b1111; h = mreg == [1,1,1,1];
J = ^b1110; j = mreg == [1,1,1,0];
K = ^b0011; k = mreg == [0,0,1,1];
L = ^b0010; l = mreg == [0,0,1,0];
M = ^b0111; m = mreg == [0,1,1,1];
N = ^b0110; n = mreg == [0,1,1,0];
Z = ^b1000; z = mreg == [1,0,0,0];
DC= ^b1010;dc = mreg == [1,0,1,0];  " This is a don't care state
```

```
" Slave State Values
S0  = ^b0100; s0  = sreg == [0,1,0,0];
S1  = ^b1000; s1  macro     {slave3};
S2  = ^b0001; s2  = sreg == [0,0,0,1];
S2W = ^b0011; s2w = sreg == [0,0,1,1];
S3W = ^b0010; s3w = sreg == [0,0,1,0];
SB  = ^b0101; sb  = sreg == [0,1,0,1];
SR0 = ^b0111; sr0 = sreg == [0,1,1,1];
SR1 = ^b0110; sr1 = sreg == [0,1,1,0];
S3  = ^b0000; s3  = sreg == [0,0,0,0];


" Abbreviations to aid in understanding state transitions
line = SIZE == [1,1];
long = SIZE == [0,0];
word = SIZE == [1,0];
byte = SIZE == [0,1];

addr0 = ADDR == [0,0];
addr1 = ADDR == [0,1];
addr2 = ADDR == [1,0];
addr3 = ADDR == [1,1];

longport macro {(  rdsack1 &  rdsack0 & !rberr) };
wordport macro {(  rdsack1 & !rdsack0 & !rberr) };
byteport macro {( !rdsack1 &  rdsack0 & !rberr) };
noport   macro {( !rdsack1 & !rdsack0 & !rberr) };

bus_err     macro { (  rberr  &  !rhalt) };
retry       macro { (  rberr  &   rhalt) };
norm_wait   macro { ( !rberr  &   rhalt) };
norm_nowait macro { ( !rberr  &  !rhalt) };


" Signifies the first time a termination signal is seen
first_term_sample macro { (rdsack0 #  rdsack1 #  rberr #  ravec) };

" Determines if s2 or s2w is to be entered. Note that TA is asserted
" during the s2.
Final  macro {( (longport & (a # f # g # k # l # m # n))
             #  (wordport & (b # f # g # k # l # m # n))
             #  (byteport & (e # h # j # k # l # m # n))
             #  (ravec & !rberr                       ) )};

equations
mast0.oe = 1 ;
mast1.oe = 1 ;
mast2.oe = 1 ;
mast3.oe = 1 ;
slave0.oe =1;
slave1.oe =1;
slave2.oe =1;
slave3.oe =1;


state_diagram mreg

   State I :
     case (ts & line) & !reset                          :A ;
```

```
        (ts & long) & !reset                    :A ;
         (ts & word & addr0) & !reset           :F ;
         (ts & word & addr2) & !reset           :G ;
         (ts & byte & addr0) & !reset           :K ;
         (ts & byte & addr1) & !reset           :L ;
         (ts & byte & addr2) & !reset           :M ;
         (ts & byte & addr3) & !reset           :N ;
        (!ts) & !reset                :I ;
         (ts & word & addr1) & !reset           :I ; " Impossible case
         (ts & word & addr3) & !reset           :I ; " Impossible case
       reset                                    :I ;
endcase;

State A :
   case (!s1) & !reset                          :A ;
        (s1 & retry  ) & !reset                 :A ;
        (s1 & bus_err) & !reset                 :Z ;
         (s1 & byteport) & !reset               :C ;
         (s1 & wordport) & !reset                :B ;
         (s1 & longport) & !reset               :I ;
         (s1 & noport) & !reset                 :I ; " Impossible case
       reset                                    :I ;
endcase;

State B :
   case (!s1) & !reset                          :B ;
        (s1 & retry ) & !reset                  :B ;
        (s1 & bus_err) & !reset                 :Z ;
         (s1 & byteport) & !reset               :Z ;
         (s1 & wordport) & !reset               :I ;
         (s1 & longport) & !reset               :Z ;
         (s1 & noport) & !reset                 :I ; " Impossible case
       reset                                    :I ;
endcase;

State C :
   case (!s1)          & !reset                 :C ;
        (s1 & retry )   & !reset                :C ;
        (s1 & bus_err)  & !reset                 :Z ;
         (s1 & byteport) & !reset               :D ;
         (s1 & wordport) & !reset               :Z ;
         (s1 & longport) & !reset               :Z ;
         (s1 & noport)   & !reset               :I ; " Impossible case
          reset                                 :I ;
endcase;

State D :
   case (!s1) & !reset                          :D ;
        (s1 & retry ) & !reset                  :D ;
        (s1 & bus_err) & !reset                 :Z ;
         (s1 & byteport) & !reset               :E ;
         (s1 & wordport) & !reset               :Z ;
         (s1 & longport) & !reset               :Z ;
         (s1 & noport) & !reset                 :I ; " Impossible case
          reset                                 :I ;
endcase;

State E:
   case (!s1) & !reset                          :E ;
        (s1 & retry ) & !reset                  :E ;
```

```
        (s1 & bus_err ) & !reset                :Z ;
        (s1 & byteport) & !reset                :I ;
        (s1 & wordport) & !reset                :Z ;
        (s1 & longport) & !reset                :Z ;
        (s1 & noport) & !reset                  :I ; " Impossible case
         reset                                  :I ;
endcase;

State F:
   case (!s1) & !reset                          :F ;
        (s1 & retry ) & !reset                  :F ;
        (s1 & bus_err ) & !reset                :Z ;
        (s1 & byteport) & !reset                :H ;
        (s1 & wordport) & !reset                :I ;
        (s1 & longport) & !reset                :I ;
        (s1 & noport) & !reset                  :I ; " Impossible case
         reset                                  :I ;
endcase;

 State G:
   case (!s1)  & !reset                         :G ;
        (s1 & retry ) & !reset                  :G ;
        (s1 & bus_err ) & !reset                :Z ;
        (s1 & byteport) & !reset                :J ;
        (s1 & wordport) & !reset                :I ;
        (s1 & longport) & !reset                :I ;
        (s1 & noport) & !reset                  :I ; " Impossible case
         reset                                  :I ;
endcase;

State H:
   case (!s1) & !reset                          :H ;
        (s1 & retry ) & !reset                  :H ;
        (s1 & bus_err ) & !reset                :Z ;
        (s1 & byteport) & !reset                :I ;
        (s1 & wordport) & !reset                :Z ;
        (s1 & longport) & !reset                :Z ;
        (s1 & noport) & !reset                  :I ; " Impossible case
         reset                                  :I ;
endcase;

State J:
   case (!s1)  & !reset                         :J ;
        (s1 & retry ) & !reset                  :J ;
        (s1 & bus_err ) & !reset                :Z ;
        (s1 & byteport) & !reset                :I ;
        (s1 & wordport) & !reset                :Z ;
        (s1 & longport) & !reset                :Z ;
        (s1 & noport) & !reset                  :I ; " Impossible case
         reset                                  :I ;
endcase;

State K:
   case (!s1) & !reset                          :K ;
        (s1 & retry ) & !reset                  :K ;
        (s1 & bus_err ) & !reset                :Z ;
        (s1 & byteport) & !reset                :I ;
        (s1 & wordport) & !reset                :I ;
        (s1 & longport) & !reset                :I ;
        (s1 & noport) & !reset                  :I ; " Impossible case
```

```
        reset                                          :I ;
    endcase;

    State L:
      case (!s1) & !reset                              :L ;
           (s1 & retry ) & !reset                      :L ;
           (s1 & bus_err ) & !reset                    :Z ;
           (s1 & byteport) & !reset                    :I ;
           (s1 & wordport) & !reset                    :I ;
           (s1 & longport) & !reset                    :I ;
           (s1 & noport) & !reset                      :I ; " Impossible case
           reset                                       :I ;
    endcase;

    State M:
      case (!s1) & !reset                              :M ;
           (s1 & retry ) & !reset                      :M ;
           (s1 & bus_err ) & !reset                    :Z ;
           (s1 & byteport) & !reset                    :I ;
           (s1 & wordport)  & !reset                   :I ;
           (s1 & longport) & !reset                    :I ;
           (s1 & noport)  & !reset                     :I ; " Impossible case
           reset                                       :I ;
    endcase;

    State N:
      case (!s1) & !reset                              :N ;
           (s1 & retry ) & !reset                      :N ;
           (s1 & bus_err ) & !reset                    :Z ;
           (s1 & byteport) & !reset                    :I ;
           (s1 & wordport) & !reset                    :I ;
           (s1 & longport) & !reset                    :I ;
           (s1 & noport) & !reset                      :I ; " Impossible case
           reset                                       :I ;
      endcase;

    State DC:
      goto I;

state_diagram sreg

    State S3:
        if   reset then S3 else
        if    i   then S3 else
        if    z   then SB else S0;

    State S0:
      case   first_term_sample &   term & !lsterm & !reset :S1;
           !first_term_sample &   term & !lsterm & !reset  :S0;
           !term & !lsterm & !reset                        :S0;
            lsterm & !reset                                :S1;
            reset                                          :S3;
     endcase;


    State S1:
      case  bus_err & !reset                               :SB ;
           retry & !reset                                  :SR0;
           (norm_wait   & !Final) & !reset                 :S3W;
           (norm_wait   &  Final) & !reset                 :S2W;
```

```
        (norm_nowait & !Final) & !reset          :S3 ;
        (norm_nowait &  Final) & !reset          :S2 ;
         reset                                   :S3 ;
    endcase;

  State S2:
     goto S3;

  State S2W:
    case   !rhalt  & !reset                      :S2 ;
           rhalt  & !reset                       :S2W;
           reset                                 :S3 ;
    endcase;

  State S3W:
    case  ( z # dc ) & !reset                    :SB ;
         !( z # dc ) &   rhalt & !reset          :S3W;
         !( z # dc ) &  !rhalt & !reset          :S3 ;
          reset                                  :S3 ;
    endcase;

  State SR0:
    case  ( z # dc ) & !reset                    :SB ;
         !( z # dc ) &   rhalt & !reset          :SR0;
         !( z # dc ) &  !rhalt & !reset          :SR1;
          reset                                  :S3 ;
    endcase;


  State SR1:
     goto S0;

  State SB:
     goto S3;


" These states are a derivative of the s1 case. They are mere duplicates
" of the s1 state. If the State S1 is modified, these should be modified
" also to ensure correct operation.
    State ^b1001:
      case  bus_err & !reset                     :SB ;
           retry & !reset                        :SR0;
           (norm_wait   & !Final) & !reset       :S3W;
           (norm_wait   &  Final) & !reset       :S2W;
           (norm_nowait & !Final) & !reset       :S3 ;
           (norm_nowait &  Final) & !reset       :S2 ;
            reset                                :S3 ;
      endcase;
    State ^b1010:
      case  bus_err & !reset                     :SB ;
           retry & !reset                        :SR0;
           (norm_wait   & !Final) & !reset       :S3W;
           (norm_wait   &  Final) & !reset       :S2W;
           (norm_nowait & !Final) & !reset       :S3 ;
           (norm_nowait &  Final) & !reset       :S2 ;
            reset                                :S3 ;
      endcase;

    State ^b1011:
      case  bus_err & !reset                     :SB ;
```

```
        retry & !reset                          :SR0;
        (norm_wait  & !Final) & !reset          :S3W;
        (norm_wait  &  Final) & !reset          :S2W;
        (norm_nowait & !Final) & !reset         :S3 ;
        (norm_nowait &  Final) & !reset         :S2 ;
         reset                                  :S3 ;
    endcase;

  State ^b1100:
    case  bus_err & !reset                      :SB ;
        retry & !reset                          :SR0;
        (norm_wait  & !Final) & !reset          :S3W;
        (norm_wait  &  Final) & !reset          :S2W;
        (norm_nowait & !Final) & !reset         :S3 ;
        (norm_nowait &  Final) & !reset         :S2 ;
         reset                                  :S3 ;
    endcase;

  State ^b1101:
    case  bus_err & !reset                      :SB ;
        retry & !reset                          :SR0;
        (norm_wait  & !Final) & !reset          :S3W;
        (norm_wait  &  Final) & !reset          :S2W;
        (norm_nowait & !Final) & !reset         :S3 ;
        (norm_nowait &  Final) & !reset         :S2 ;
         reset                                  :S3 ;
    endcase;

  State ^b1110:
    case  bus_err & !reset                      :SB ;
        retry & !reset                          :SR0;
        (norm_wait  & !Final) & !reset          :S3W;
        (norm_wait  &  Final) & !reset          :S2W;
        (norm_nowait & !Final) & !reset         :S3 ;
        (norm_nowait &  Final) & !reset         :S2 ;
         reset                                  :S3 ;
    endcase;

  State ^b1111:
    case  bus_err & !reset                      :SB ;
        retry & !reset                          :SR0;
        (norm_wait  & !Final) & !reset          :S3W;
        (norm_wait  &  Final) & !reset          :S2W;
        (norm_nowait & !Final) & !reset         :S3 ;
        (norm_nowait &  Final) & !reset         :S2 ;
         reset                                  :S3 ;
    endcase;



test_vectors
([bclk,!ts,ADDR,SIZE,DSACK,!rberr,!rhalt,!!sterm,!term,!ravec,!reset] -> [mreg,slave3,sreg_sub])
 [.p. , 1 , 0  , 0  , 0  , 1  , 1  , 1  , 1  , 1  , 1  , 1  ] -> [ 15  , 1  ,  7 ];
 [ ck , x , x  , x  , x  , x  , x  , x  , x  , 1  , 1  , 0  ] -> [  I  , 0  , S3 ];
 [ ck , 1 , x  , x  , x  , x  , x  , x  , x  , 1  , 1  , 1  ] -> [  I  , 0  , S3 ];
 [ ck , 0 , 0  , 0  , x  , x  , x  , x  , x  , 1  , 1  , 1  ] -> [  A  , 0  , S3 ];
 [ ck , 1 , 0  , 0  , x  , x  , x  , 1  , 1  , 1  , 1  , 1  ] -> [  A  , 0  , S0 ];
 [ ck , 1 , 0  , 0  , x  , 1  , 1  , 0  , 1  , 1  , 1  , 1  ] -> [  A  , 1  , x ];"sterm
 [ ck , 1 , 0  , 0  ,^b11 ,  1  ,  1  , x  , x  , 1  ,  1  , 1  ] -> [  I  , 0  , S2 ];"ta ass.
```

```
[ ck , 1 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"long read
[ ck , 1 , 0 , 0 , x , x , x , 1 , 1 , 1 , 1 ] -> [ A , 0 , S0 ];"10

[ ck , 1 , 0 , 0 , x , 1 , 1 , 0 , 1 , 1 , 1 ] -> [ A , 1 , x ];"sterm with
[ ck , 1 , 0 , 0 ,^b11 , 0 , 1 , x , x , 1 , 1 ] -> [ Z , 0 , SB ];"late berr
[ ck , 1 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"line read
[ ck , 1 , 0 , 0 , x , x , x , 1 , 1 , 1 , 1 ] -> [ A , 0 , S0 ];
[ ck , 1 , 0 , 0 , x , 1 , 1 , 0 , 1 , 1 , 1 ] -> [ A , 1 , x ];"sterm with
[ ck , 1 , 0 , 0 ,^b11 , 0 , 0 , x , x , 1 , 1 ] -> [ A , 0 , SR0];"late retry
[ ck , 1 , 0 , 0 , x , x , 0 , x , x , 1 , 1 ] -> [ A , 0 , SR0];
[ ck , 1 , 0 , 0 , x , x , 1 , x , x , 1 , 1 ] -> [ A , 0 , SR1];
[ ck , 1 , 0 , 0 , x , x , x , 1 , 1 , 1 , 1 ] -> [ A , 0 , S0 ];"20

[ ck , 1 , 0 , 0 , x , x , x , 1 , 1 , 1 , 1 ] -> [ A , 0 , S0 ];
[ ck , 1 , 0 , 0 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ A , 1 , x ];
[ ck , 1 , 0 , 0 ,^b11 , 1 , 1 , x , x , 1 , 1 ] -> [ I , 0 , S2 ];"ta ass.
[ ck , 1 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"line read
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , 1 , 0 , 1 , 1 ] -> [ A , 1 , x ];"byte dsack
[ ck , 1 , 0 , 3 ,^b01 , 1 , 0 , x , x , 1 , 1 ] -> [ C , 0 , S3W];
[ ck , 1 , 0 , 3 , x , x , 1 , x , x , 1 , 1 ] -> [ C , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ C , 0 , S0 ];"30

[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ C , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ D , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ D , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ D , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ E , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ E , 0 , S0 ];
[ ck , 1 , 0 , 3 , x , 0 , x , 1 , 0 , 1 , 1 ] -> [ E , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 0 , 0 , x , x , 1 , 1 ] -> [ E , 0 , SR0];
[ ck , 1 , 0 , 3 , x , x , 1 , x , x , 1 , 1 ] -> [ E , 0 , SR1];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ E , 0 , S0 ];"40

[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ E , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ I , 0 , S2 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"line read
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S0 ];"word port
[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ A , 1 , x ];
[ ck , 1 , 0 , 3 ,^b10 , 1 , 1 , x , x , 1 , 1 ] -> [ B , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ B , 0 , S0 ];
[ ck , 1 , 0 , 3 , x , 0 , x , 1 , 0 , 1 , 1 ] -> [ B , 1 , x ];
[ ck , 1 , 0 , 3 ,^b10 , 0 , 0 , x , x , 1 , 1 ] -> [ B , 0 , SR0];"50

[ ck , 1 , 0 , 3 , x , x , 1 , x , x , 1 , 1 ] -> [ B , 0 , SR1];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ B , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b10 , x , x , 1 , 0 , 1 , 1 ] -> [ B , 1 , x ];
[ ck , 1 , 0 , 3 ,^b10 , 1 , 1 , x , x , 1 , 1 ] -> [ I , 0 , S2 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"long read
[ ck , 1 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S0 ];"word port
[ ck , 1 , 0 , 0 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ A , 1 , x ];
[ ck , 1 , 0 , 0 ,^b10 , 1 , 1 , x , x , 1 , 1 ] -> [ B , 0 , S3 ];
[ ck , 1 , 0 , 0 , x , x , x , x , x , 1 , 1 ] -> [ B , 0 , S0 ];"60

[ ck , 1 , 0 , 0 ,^b01 , 1 , 1 , 1 , 0 , 1 , 1 ] -> [ B , 1 , x ];
```

```
[ck,1,0,0,[x,1], 1 , 1 , x , x , 1 , 1 ]->[ Z , 0 , S3 ];
[ck,1,0,0, x , x , x , x , x , 1 , 1 ]->[ I , 0 , SB ];"port violation
[ck,1,0,0, x , x , x , x , x , 1 , 1 ]->[ I , 0 , S3 ];
[ck,0,0,2, x , x , x , x , x , 1 , 1 ]->[ F , 0 , S3 ];"word,addr0
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ F , 0 , S0 ];"byte port
[ck,1,0,2,^b01, x , x , 1 , 0 , 1 , 1 ]->[ F , 1 , x ];
[ck,1,0,2,^b01, 1 , 1 , x , x , 1 , 1 ]->[ H , 0 , S3 ];
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ H , 0 , S0 ];
[ck,1,0,2, x , 0 , x , 1 , 0 , 1 , 1 ]->[ H , 1 , x ];"70

[ck,1,0,2,^b01, 0 , 0 , x , x , 1 , 1 ]->[ H , 0 , SR0];
[ck,1,0,2, x , x , 1 , x , x , 1 , 1 ]->[ H , 0 , SR1];
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ H , 0 , S0 ];
[ck,1,0,2,^b01, x , x , 1 , 0 , 1 , 1 ]->[ H , 1 , x ];
[ck,1,0,2,^b01, 1 , 1 , x , x , 1 , 1 ]->[ I , 0 , S2 ];
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ I , 0 , S3 ];
[ck,0,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S3 ];"word,addr2
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S0 ];"byte port
[ck,1,2,2,^b01, x , x , 1 , 0 , 1 , 1 ]->[ G , 1 , x ];
[ck,1,2,2,^b01, 1 , 1 , x , x , 1 , 1 ]->[ J , 0 , S3 ];"80

[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ J , 0 , S0 ];
[ck,1,2,2, x , 0 , x , 1 , 0 , 1 , 1 ]->[ J , 1 , x ];
[ck,1,2,2,^b01, 0 , 0 , x , x , 1 , 1 ]->[ J , 0 , SR0];
[ck,1,2,2, x , x , 1 , x , x , 1 , 1 ]->[ J , 0 , SR1];
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ J , 0 , S0 ];
[ck,1,2,2,^b01, x , x , 1 , 0 , 1 , 1 ]->[ J , 1 , x ];
[ck,1,2,2,^b01, 1 , 1 , x , x , 1 , 1 ]->[ I , 0 , S2 ];
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ I , 0 , S3 ];
[ck,0,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S3 ];"word,addr2
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S0 ];"word port  90

[ck,1,2,2, x , 0 , x , 1 , 0 , 1 , 1 ]->[ G , 1 , x ];
[ck,1,2,2,^b10, 0 , 0 , x , x , 1 , 1 ]->[ G , 0 , SR0];
[ck,1,2,2, x , x , 1 , x , x , 1 , 1 ]->[ G , 0 , SR1];
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S0 ];
[ck,1,2,2,^b10, x , x , 1 , 0 , 1 , 1 ]->[ G , 1 , x ];
[ck,1,2,2,^b10, 1 , 1 , x , x , 1 , 1 ]->[ I , 0 , S2 ];
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ I , 0 , S3 ];
[ck,0,0,2, x , x , x , x , x , 1 , 1 ]->[ F , 0 , S3 ];"word,addr0
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ F , 0 , S0 ];"word port
[ck,1,0,2, x , 0 , x , 1 , 0 , 1 , 1 ]->[ F , 1 , x ];" 100

[ck,1,0,2,^b10, 0 , 0 , x , x , 1 , 1 ]->[ F , 0 , SR0];
[ck,1,0,2, x , x , 1 , x , x , 1 , 1 ]->[ F , 0 , SR1];
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ F , 0 , S0 ];
[ck,1,0,2,^b10, x , x , 1 , 0 , 1 , 1 ]->[ F , 1 , x ];
[ck,1,0,2,^b10, 1 , 1 , x , x , 1 , 1 ]->[ I , 0 , S2 ];
[ck,1,0,2, x , x , x , x , x , 1 , 1 ]->[ I , 0 , S3 ];
[ck,0,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S3 ];"word,addr2
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S0 ];"long port
[ck,1,2,2, x , 0 , x , 1 , 0 , 1 , 1 ]->[ G , 1 , x ];
[ck,1,2,2,^b10, 0 , 0 , x , x , 1 , 1 ]->[ G , 0 , SR0];"110

[ck,1,2,2, x , x , 1 , x , x , 1 , 1 ]->[ G , 0 , SR1];
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ G , 0 , S0 ];
[ck,1,2,2,^b11, x , x , 1 , 0 , 1 , 1 ]->[ G , 1 , x ];
[ck,1,2,2,^b11, 1 , 1 , x , x , 1 , 1 ]->[ I , 0 , S2 ];
[ck,1,2,2, x , x , x , x , x , 1 , 1 ]->[ I , 0 , S3 ];
[ck,0,0,2, x , x , x , x , x , 1 , 1 ]->[ F , 0 , S3 ];"word,addr0
```

```
[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ F , 0 , S0 ];"word port
[ ck , 1 , 0 , 2 , x , 0 , x , 1 , 0 , 1 , 1 ] -> [ F , 1 , x ];
[ ck , 1 , 0 , 2 ,^b10 , 0 , 0 , x , x , 1 , 1 ] -> [ F , 0 , SR0];
[ ck , 1 , 0 , 2 , x , x , 1 , x , x , 1 , 1 ] -> [ F , 0 , SR1];"120

[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ F , 0 , S0 ];
[ ck , 1 , 0 , 2 ,^b11 , x , x , 1 , 0 , 1 , 1 ] -> [ F , 1 , x ];
[ ck , 1 , 0 , 2 ,^b11 , 1 , 1 , x , x , 1 , 1 ] -> [ I , 0 , S2 ];
[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 2 , 2 , x , x , x , x , x , 1 , 1 ] -> [ G , 0 , S3 ];"word,addr2
[ ck , 1 , 2 , 2 , x , x , x , x , x , 1 , 1 ] -> [ G , 0 , S0 ];"byte port
[ ck , 1 , 2 , 2 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ G , 1 , x ];
[ ck , 1 , 2 , 2 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ J , 0 , S3 ];
[ ck , 1 , 2 , 2 , x , x , x , x , x , 1 , 1 ] -> [ J , 0 , S0 ];
[ ck , 1 , 2 , 2 , x , 0 , x , 1 , 0 , 1 , 1 ] -> [ J , 1 , x ];"130

[ ck , 1 , 2 , 2 ,[1,x], 1 , 1 , x , x , 1 , 1 ] -> [ Z , 0 , S3 ];"port violation
[ ck , 1 , 2 , 2 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , SB ];
[ ck , 1 , 2 , 2 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ F , 0 , S3 ];"word,addr0
[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ F , 0 , S0 ];"byte port
[ ck , 1 , 0 , 2 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ F , 1 , x ];
[ ck , 1 , 0 , 2 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ H , 0 , S3 ];
[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ H , 0 , S0 ];
[ ck , 1 , 0 , 2 , x , 0 , x , 1 , 0 , 1 , 1 ] -> [ H , 1 , x ];
[ ck , 1 , 0 , 2 ,[1,x], 1 , 1 , x , x , 1 , 1 ] -> [ Z , 0 , S3 ];"port violation 140

[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , SB ];
[ ck , 1 , 0 , 2 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"line read
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , 1 , 0 , 1 , 1 ] -> [ A , 1 , x ];"byte dsack
[ ck , 1 , 0 , 3 ,^b01 , 1 , 0 , x , x , 1 , 1 ] -> [ C , 0 , S3W];
[ ck , 1 , 0 , 3 , x , x , 1 , x , x , 1 , 1 ] -> [ C , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ C , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ C , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ D , 0 , S3 ];"150

[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ D , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ D , 1 , x ];
[ ck , 1 , 0 , 3 ,[1,x], 1 , 1 , x , x , 1 , 1 ] -> [ Z , 0 , S3 ];"port violation
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , SB ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , S3 ];
[ ck , 0 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S3 ];"line read
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ A , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , 1 , 0 , 1 , 1 ] -> [ A , 1 , x ];"byte dsack
[ ck , 1 , 0 , 3 ,^b01 , 1 , 0 , x , x , 1 , 1 ] -> [ C , 0 , S3W];
[ ck , 1 , 0 , 3 , x , x , 1 , x , x , 1 , 1 ] -> [ C , 0 , S3 ];"160

[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ C , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ C , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ D , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ D , 0 , S0 ];
[ ck , 1 , 0 , 3 ,^b01 , x , x , 1 , 0 , 1 , 1 ] -> [ D , 1 , x ];
[ ck , 1 , 0 , 3 ,^b01 , 1 , 1 , x , x , 1 , 1 ] -> [ E , 0 , S3 ];
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ E , 0 , S0 ];
[ ck , 1 , 0 , 3 ,[1,x], 1 , 0 , 1 , 0 , 1 , 1 ] -> [ E , 1 , x ];
[ ck , 1 , 0 , 3 ,[1,x], 1 , 0 , x , x , 1 , 1 ] -> [ Z , 0 , S3W];"port violation
[ ck , 1 , 0 , 3 , x , x , x , x , x , 1 , 1 ] -> [ I , 0 , SB ]; "170
```

```
[ck,1,0,3,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,0,1,x,x,x,x,x,1,1]->[K,0,S3];"byte,addr0
[ck,1,0,1,x,x,x,x,x,1,1]->[K,0,S0];"byte port
[ck,1,0,1,x,0,x,1,0,1,1]->[K,1,x];
[ck,1,0,1,^b01,0,0,x,x,1,1]->[K,0,SR0];
[ck,1,0,1,x,x,1,x,x,1,1]->[K,0,SR1];
[ck,1,0,1,x,x,x,x,x,1,1]->[K,0,S0];
[ck,1,0,1,^b01,x,x,1,0,1,1]->[K,1,x];
[ck,1,0,1,^b01,1,1,x,x,1,1]->[I,0,S2];
[ck,1,0,1,x,x,x,x,x,1,1]->[I,0,S3];"180

[ck,0,0,1,x,x,x,x,x,1,1]->[K,0,S3];"byte,addr0
[ck,1,0,1,x,x,x,x,x,1,1]->[K,0,S0];"long port
[ck,1,0,1,x,0,x,1,0,1,1]->[K,1,x];
[ck,1,0,1,^b11,0,0,x,x,1,1]->[K,0,SR0];
[ck,1,0,1,x,x,1,x,x,1,1]->[K,0,SR1];
[ck,1,0,1,x,x,x,x,x,1,1]->[K,0,S0];
[ck,1,0,1,^b11,x,x,1,0,1,1]->[K,1,x];
[ck,1,0,1,^b11,1,1,x,x,1,1]->[I,0,S2];
[ck,1,0,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,0,1,x,x,x,x,x,1,1]->[K,0,S3];"byte,addr0  190

[ck,1,0,1,x,x,x,x,x,1,1]->[K,0,S0];"word port
[ck,1,0,1,x,0,x,1,0,1,1]->[K,1,x];
[ck,1,0,1,^b10,0,0,x,x,1,1]->[K,0,SR0];
[ck,1,0,1,x,x,1,x,x,1,1]->[K,0,SR1];
[ck,1,0,1,x,x,x,x,x,1,1]->[K,0,S0];
[ck,1,0,1,^b10,x,x,1,0,1,1]->[K,1,x];
[ck,1,0,1,^b10,1,1,x,x,1,1]->[I,0,S2];
[ck,1,0,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,1,1,x,x,x,x,x,1,1]->[L,0,S3];"byte,addr1
[ck,1,1,1,x,x,x,x,x,1,1]->[L,0,S0];"byte port  200

[ck,1,1,1,x,0,x,1,0,1,1]->[L,1,x];
[ck,1,1,1,^b01,0,0,x,x,1,1]->[L,0,SR0];
[ck,1,1,1,x,x,1,x,x,1,1]->[L,0,SR1];
[ck,1,1,1,x,x,x,x,x,1,1]->[L,0,S0];
[ck,1,1,1,^b01,x,x,1,0,1,1]->[L,1,x];
[ck,1,1,1,^b01,1,1,x,x,1,1]->[I,0,S2];
[ck,1,1,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,1,1,x,x,x,x,x,1,1]->[L,0,S3];"byte,addr1
[ck,1,1,1,x,x,x,x,x,1,1]->[L,0,S0];"long port
[ck,1,1,1,x,0,x,1,0,1,1]->[L,1,x];"210

[ck,1,1,1,^b11,0,0,x,x,1,1]->[L,0,SR0];
[ck,1,1,1,x,x,1,x,x,1,1]->[L,0,SR1];
[ck,1,1,1,x,x,x,x,x,1,1]->[L,0,S0];
[ck,1,1,1,^b11,x,x,1,0,1,1]->[L,1,x];
[ck,1,1,1,^b11,1,1,x,x,1,1]->[I,0,S2];
[ck,1,1,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,1,1,x,x,x,x,x,1,1]->[L,0,S3];"byte,addr1
[ck,1,1,1,x,x,x,x,x,1,1]->[L,0,S0];"word port
[ck,1,1,1,x,0,x,1,0,1,1]->[L,1,x];
[ck,1,1,1,^b10,0,0,x,x,1,1]->[L,0,SR0];"220

[ck,1,1,1,x,x,1,x,x,1,1]->[L,0,SR1];
[ck,1,1,1,x,x,x,x,x,1,1]->[L,0,S0];
[ck,1,1,1,^b10,x,x,1,0,1,1]->[L,1,x];
[ck,1,1,1,^b10,1,1,x,x,1,1]->[I,0,S2];
[ck,1,1,1,x,x,x,x,x,1,1]->[I,0,S3];
```

```
[ck,0,2,1,x,x,x,x,x,1,1]->[M,0,S3];"byte,addr2
[ck,1,2,1,x,x,x,x,x,1,1]->[M,0,S0];"byte port
[ck,1,2,1,x,0,x,1,0,1,1]->[M,1,x];
[ck,1,2,1,^b01,0,0,x,x,1,1]->[M,0,SR0];
[ck,1,2,1,x,x,1,x,x,1,1]->[M,0,SR1];"230

[ck,1,2,1,x,x,x,x,x,1,1]->[M,0,S0];
[ck,1,2,1,^b01,x,x,1,0,1,1]->[M,1,x];
[ck,1,2,1,^b01,1,1,x,x,1,1]->[I,0,S2];
[ck,1,2,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,2,1,x,x,x,x,x,1,1]->[M,0,S3];"byte,addr2
[ck,1,2,1,x,x,x,x,x,1,1]->[M,0,S0];"long port
[ck,1,2,1,x,0,x,1,0,1,1]->[M,1,x];
[ck,1,2,1,^b11,0,0,x,x,1,1]->[M,0,SR0];
[ck,1,2,1,x,x,1,x,x,1,1]->[M,0,SR1];
[ck,1,2,1,x,x,x,x,x,1,1]->[M,0,S0];"240

[ck,1,2,1,^b11,x,x,1,0,1,1]->[M,1,x];
[ck,1,2,1,^b11,1,1,x,x,1,1]->[I,0,S2];
[ck,1,2,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,2,1,x,x,x,x,x,1,1]->[M,0,S3];"byte,addr2
[ck,1,2,1,x,x,x,x,x,1,1]->[M,0,S0];"word port
[ck,1,2,1,x,0,x,1,0,1,1]->[M,1,x];
[ck,1,2,1,^b10,0,0,x,x,1,1]->[M,0,SR0];
[ck,1,2,1,x,x,1,x,x,1,1]->[M,0,SR1];
[ck,1,2,1,x,x,x,x,x,1,1]->[M,0,S0];
[ck,1,2,1,^b10,x,x,1,0,1,1]->[M,1,x];"250

[ck,1,2,1,^b10,1,1,x,x,1,1]->[I,0,S2];
[ck,1,2,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,3,1,x,x,x,x,x,1,1]->[N,0,S3];"byte,addr3
[ck,1,3,1,x,x,x,x,x,1,1]->[N,0,S0];"byte port
[ck,1,3,1,x,0,x,1,0,1,1]->[N,1,x];
[ck,1,3,1,^b01,0,0,x,x,1,1]->[N,0,SR0];
[ck,1,3,1,x,x,1,x,x,1,1]->[N,0,SR1];
[ck,1,3,1,x,x,x,x,x,1,1]->[N,0,S0];
[ck,1,3,1,^b01,x,x,1,0,1,1]->[N,1,x];
[ck,1,3,1,^b01,1,1,x,x,1,1]->[I,0,S2];"260

[ck,1,3,1,x,x,x,x,x,1,1]->[I,0,S3];
[ck,0,3,1,x,x,x,x,x,1,1]->[N,0,S3];"byte,addr3
[ck,1,3,1,x,x,x,x,x,1,1]->[N,0,S0];"long port
[ck,1,3,1,x,0,x,1,0,1,1]->[N,1,x];
[ck,1,3,1,^b11,0,0,x,x,1,1]->[N,0,SR0];
[ck,1,3,1,x,x,1,x,x,1,1]->[N,0,SR1];
[ck,1,3,1,x,x,x,x,x,1,1]->[N,0,S0];
[ck,1,3,1,^b11,x,x,1,0,1,1]->[N,1,x];
[ck,1,3,1,^b11,1,1,x,x,1,1]->[I,0,S2];
[ck,1,3,1,x,x,x,x,x,1,1]->[I,0,S3];"270

[ck,0,3,1,x,x,x,x,x,1,1]->[N,0,S3];"byte,addr3
[ck,1,3,1,x,x,x,x,x,1,1]->[N,0,S0];"word port
[ck,1,3,1,x,0,x,1,0,1,1]->[N,1,x];
[ck,1,3,1,^b10,0,0,x,x,1,1]->[N,0,SR0];
[ck,1,3,1,x,x,1,x,x,1,1]->[N,0,SR1];
[ck,1,3,1,x,x,x,x,x,1,1]->[N,0,S0];
[ck,1,3,1,^b10,x,x,1,0,1,1]->[N,1,x];
[ck,1,3,1,^b10,1,1,x,x,1,1]->[I,0,S2];
[ck,1,3,1,x,x,x,x,x,1,1]->[I,0,S3];"279
```

```
end CONTROL;
```

7.7.2 MC68040 Bus Translator Termination Generation PAL

```
module TERM flag '-r3';
title '68040 Bus-Translator Termination Generation PAL'

term device 'p22vp10';

" Inputs
bclk                                    pin 1;
rd,slave0,slave1,slave2,slave3          pin 2,3,4,5,6;
!lsterm,aterm                           pin 9,10;
!ts,!line                               pin 7,8;
!ciin                                   pin 11;
!_030_avec                              pin 13;
tristate                                pin 23;


" Outputs
!ta,!tea                                pin 22,21;
!tbi,!tci,!avec                         pin 20,19,18;
!as,!ds,!nc0,!nc1                       pin 17,16,15,14;


strob =  [as,ds,nc1,nc0];
sreg  =  [slave3,slave2,slave1,slave0];
x,ck  =  .x.,.c.;

as,ds,nc0,nc1  istype  'neg, reg, feed_reg';
tci,avec       istype  'pos, reg, feed_reg';

" Slave State Values
S0  = ^b0100; s0  = sreg == [0,1,0,0];
S1  = ^b1000; s1  macro     {slave3};
S2  = ^b0001; s2  = sreg == [0,0,0,1];
S2W = ^b0011; s2w = sreg == [0,0,1,1];
S3W = ^b0010; s3w = sreg == [0,0,1,0];
SB  = ^b0101; sb  = sreg == [0,1,0,1];
SR0 = ^b0111; sr0 = sreg == [0,1,1,1];
SR1 = ^b0110; sr1 = sreg == [0,1,1,0];
S3  = ^b0000; s3  = sreg == [0,0,0,0];


" State Machine for generating ~AS and ~DS.
" bits nc1 and nc0 are used internally and must
" not be connected externally.
sidle      = ^b0000; " Idle state, wait for TS assertion
ts_rec     = ^b0010; " Transfer start recognized
rd_state   = ^b1100; " Assert Address Strobe and Data Strobe
wr_state   = ^b1000; " Assert Address Strobe only
wait_term  = ^b1101; " Wait for termination to occur
cycle_end  = ^b0001; " End of cycle go back to start

wr  macro { !rd };
```

```
equations

tbi      = s2 & line;
ta       = s2 ;
tea      = sb;

tci      := (!as & ciin & !tci)  #  ( tci & !( ta # tea # as)) ;
avec     := (!as & _030_avec & !avec) # ( avec & !(ta # tea # as));

nc0.oe    = 1;
nc1.oe    = 1;
tci.oe    = 1;
avec.oe   = 1;

as.oe     = !tristate;
ds.oe     = !tristate;
nc0.oe    = !tristate;
nc1.oe    = !tristate;
tci.oe    = !tristate;
avec.oe   = !tristate;


" This state diagram is for generating AS and DS.
state_diagram strob

State sidle:
  case  ts              :ts_rec;
       !ts              :sidle;
  endcase;

State ts_rec:
  case  rd              :rd_state;    " Assert both AS and DS
       !rd              :wr_state;    " Only AS is asserted
  endcase;

State rd_state:
  case  !lsterm         :wait_term;
        lsterm          :cycle_end;   " Synchronous termination detected.
  endcase;

State wr_state:
  case  !lsterm         :wait_term;
        lsterm          :cycle_end;   " Synchronous termination detected.

  endcase;

State wait_term:
  case  (aterm # lsterm)    :cycle_end;   " Termination detected
       !(aterm # lsterm)    :wait_term;   " wait
  endcase;

State cycle_end:
  case   (s2 # sb) & !(sr1 # s3)        :sidle;
         !(s2 # sb) &  (sr1 # s3) &  rd     :rd_state;
         !(s2 # sb) &  (sr1 # s3) & !rd     :wr_state;
         !(s2 # sb) & !(sr1 # s3)           :cycle_end;
         (s2 # sb) &  (sr1 # s3)            :sidle;   " impossible case
```

```
  endcase;
test_vectors
([bclk,!ts,rd,sreg,!sterm,aterm,!tristate] -> [strob     ]) "!as,!ds
[ ck , 1 ,1 , S3 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 0 ,1 , S3 , x  , x  ,  1    ] -> [ts_rec   ]; " 1 , 1
[ ck , 1 ,1 , S3 , x  , x  ,  1    ] -> [rd_state ]; " 0 , 0
[ ck , 1 ,1 , S0 , 0  , 0  ,  1    ] -> [wait_term]; " 0 , 0
[ ck , 1 ,1 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S2 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 0 ,x , S3 , x  , x  ,  1    ] -> [ts_rec   ]; " 1 , 1
[ ck , 1 ,0 , S3 , x  , x  ,  1    ] -> [wr_state ]; " 0 , 1
[ ck , 1 ,0 , S0 , 0  , 0  ,  1    ] -> [wait_term]; " 0 , 0
[ ck , 1 ,0 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S2 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 0 ,x , S3 , x  , x  ,  1    ] -> [ts_rec   ]; " 1 , 1
[ ck , 1 ,1 , S3 , x  , x  ,  1    ] -> [rd_state ]; " 0 , 0
[ ck , 1 ,1 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S2 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 0 ,x , S3 , x  , x  ,  1    ] -> [ts_rec   ]; " 1 , 1
[ ck , 1 ,0 , S3 , x  , x  ,  1    ] -> [wr_state ]; " 0 , 1
[ ck , 1 ,0 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S2 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 0 ,x , S3 , x  , x  ,  1    ] -> [ts_rec   ]; " 1 , 1
[ ck , 1 ,0 , S3 , x  , x  ,  1    ] -> [wr_state ]; " 0 , 1
[ ck , 1 ,0 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , SR0, x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , SR1, x  , x  ,  1    ] -> [wr_state ]; " 0 , 1
[ ck , 1 ,0 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , S3W, x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,0 , SB , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 1 ,0 , S3 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 0 ,x , S3 , x  , x  ,  1    ] -> [ts_rec   ]; " 1 , 1
[ ck , 1 ,1 , S3 , x  , x  ,  1    ] -> [rd_state ]; " 0 , 0
[ ck , 1 ,1 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , SR0, x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , SR1, x  , x  ,  1    ] -> [rd_state ]; " 0 , 0
[ ck , 1 ,1 , S0 , 1  , 0  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S1 , x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , S3W, x  , x  ,  1    ] -> [cycle_end]; " 1 , 1
[ ck , 1 ,1 , SB , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1
[ ck , 1 ,1 , S3 , x  , x  ,  1    ] -> [sidle    ]; " 1 , 1

end TERM;
```

7.7.3 MC68040 Bus-Translator Output Enable PAL

```
module TRANSFER flag '-r3';
title '68040 Bus-Translator Transfer Attributes'

transfer device 'p22vp10';

a1,a0,siz1,siz0                    pin 1,2,3,4;
tt1,tt0,tm2,tm1,tm0                pin 5,6,7,8,9;
```

```
mast3,mast2,mast1,mast0                    pin 10,11,13,14;
tristate                                      pin 23;

_030_a1,_030_a0,_030_siz1,_030_siz0           pin 22,21,19,18;
fc2,fc1,fc0                                   pin 17,16,15;
!iack                                          pin 20;

x=.x.;
address  = [a1,a0] ;
size     = [siz1,siz0];
mreg     = [mast3..mast0];
type     = [tt1,tt0];
mod      = [tm2..tm0];
fcode    = [fc2..fc0];
_030_size = [_030_siz1,_030_siz0];
_030_addr = [_030_a1,_030_a0];

I = ^b0000; i = mreg == [0,0,0,0];
A = ^b0001; a = mreg == [0,0,0,1];
B = ^b0101; b = mreg == [0,1,0,1];
C = ^b1101; c = mreg == [1,1,0,1];
D = ^b0100; d = mreg == [0,1,0,0];
E = ^b1100; e = mreg == [1,1,0,0];
F = ^b1001; f = mreg == [1,0,0,1];
G = ^b1011; g = mreg == [1,0,1,1];
H = ^b1111; h = mreg == [1,1,1,1];
J = ^b1110; j = mreg == [1,1,1,0];
K = ^b0011; k = mreg == [0,0,1,1];
L = ^b0010; l = mreg == [0,0,1,0];
M = ^b0111; m = mreg == [0,1,1,1];
N = ^b0110; n = mreg == [0,1,1,0];
Z = ^b1000; z = mreg == [1,0,0,0];
DC= ^b1010;dc = mreg == [1,0,1,0];

norm   = ^b00;
alt    = ^b10;
intack = ^b11;
mov16  = ^b01;

line = ^b11;
long = ^b00;
word = ^b10;
byte = ^b01;

sup_dat = ^b100;
sup_cod = ^b110;
usr_dat = ^b000;
usr_cod = ^b010;
cpu_spc = ^b111;

truth_table
([mreg,address] -> [_030_addr])
 [ I ,   0    ] -> [  0  ];
 [ I ,   1    ] -> [  1  ];
 [ I ,   2    ] -> [  2  ];
 [ I ,   3    ] -> [  3  ];
 [ Z ,   0    ] -> [  0  ];
 [ Z ,   1    ] -> [  1  ];
 [ Z ,   2    ] -> [  2  ];
 [ Z ,   3    ] -> [  3  ];
```

```
[ DC ,  0   ] -> [   0   ];
[ DC ,  1   ] -> [   1   ];
[ DC ,  2   ] -> [   2   ];
[ DC ,  3   ] -> [   3   ];
[ A  ,   x  ] -> [   0   ];
[ B  ,   x  ] -> [   2   ];
[ C  ,   x  ] -> [   1   ];
[ D  ,   x  ] -> [   2   ];
[ E  ,   x  ] -> [   3   ];
[ F  ,   x  ] -> [   0   ];
[ G  ,   x  ] -> [   2   ];
[ H  ,   x  ] -> [   1   ];
[ J  ,   x  ] -> [   3   ];
[ K  ,   x  ] -> [   0   ];
[ L  ,   x  ] -> [   1   ];
[ M  ,   x  ] -> [   2   ];
[ N  ,   x  ] -> [   3   ];

truth_table
([mreg,size ] -> [_030_size])
[ I  ,byte ] ->  [ byte  ] ;
[ I  ,word ] ->  [ word  ] ;
[ I  ,long ] ->  [ long  ] ;
[ I  ,line ] ->  [ line  ] ;
[ Z  ,byte ] ->  [ byte  ] ;
[ Z  ,word ] ->  [ word  ] ;
[ Z  ,long ] ->  [ long  ] ;
[ Z  ,line ] ->  [ line  ] ;
[ DC ,byte ] ->  [ byte  ] ;
[ DC ,word ] ->  [ word  ] ;
[ DC ,long ] ->  [ long  ] ;
[ DC ,line ] ->  [ line  ] ;
[ A  ,  x  ] ->  [ long  ] ;
[ B  ,  x  ] ->  [ word  ] ;
[ C  ,  x  ] ->  [ byte  ] ;
[ D  ,  x  ] ->  [ word  ] ;
[ E  ,  x  ] ->  [ byte  ] ;
[ F  ,  x  ] ->  [ word  ] ;
[ G  ,  x  ] ->  [ word  ] ;
[ H  ,  x  ] ->  [ byte  ] ;
[ J  ,  x  ] ->  [ byte  ] ;
[ K  ,  x  ] ->  [ byte  ] ;
[ L  ,  x  ] ->  [ byte  ] ;
[ M  ,  x  ] ->  [ byte  ] ;
[ N  ,  x  ] ->  [ byte  ] ;

truth_table
([ type ,  mod  ,address] -> [ fcode ])
 [intack,   x   ,  x    ] -> [cpu_spc];
 [mov16 ,[0,x,x],  x    ] -> [usr_dat];
 [mov16 ,[1,x,x],  x    ] -> [sup_dat];
 [alt   ,  0    ,  x    ] -> [  0    ];
 [alt   ,  1    ,  x    ] -> [  1    ];
 [alt   ,  2    ,  x    ] -> [  2    ];
 [alt   ,  3    ,  x    ] -> [  3    ];
 [alt   ,  4    ,  x    ] -> [  4    ];
 [alt   ,  5    ,  x    ] -> [  5    ];
 [alt   ,  6    ,  x    ] -> [  6    ];
 [alt   ,  7    ,  x    ] -> [  7    ];
 [norm  ,  0    ,  x    ] -> [sup_dat];
```

```
[norm  ,  1   ,  x    ] -> [usr_dat];
[norm  ,  2   ,  x    ] -> [usr_cod];
[norm  ,  3   ,  x    ] -> [sup_dat];
[norm  ,  4   ,  x    ] -> [sup_dat];
[norm  ,  5   ,  x    ] -> [sup_dat];
[norm  ,  6   ,  x    ] -> [sup_cod];
[norm  ,  7   ,  x    ] -> [sup_dat];

equations

iack = (type == intack) & (address == 3);
_030_a0.oe = !tristate;
fc2.oe     = !tristate;
fc1.oe     = !tristate;
fc0.oe     = !tristate;
_030_siz0.oe = !tristate;
_030_siz1.oe = !tristate;

test_vectors
([mreg,address,size, type , mod,tristate] -> [ fcode  ,_030_size,_030_addr ,!iack ])
[ N  ,  3   ,byte,intack,  x ,   0    ] -> [cpu_spc ,  byte   ,   3     ,   0   ];
[ K  ,  0   ,byte,intack,  x ,   0    ] -> [cpu_spc ,  byte   ,   0     ,   1   ];
[ x  ,  0   ,long, alt   ,  5 ,  0    ] -> [   5    ,  long   ,   0     ,   1   ];
[ x  ,  2   ,word, alt   ,  6 ,  0    ] -> [   6    ,  word   ,   2     ,   1   ];

end TRANSFER;
```

7.7.4 MC68040 Bus-Translator Master PAL

```
module LATCH_EN flag '-r3';
title '68040 Bus-Translator Master PAL'

latch_en device 'P16R8';

"Inputs
bclk                                                    pin 1;
!s3_or_sr1                                              pin 3;
at,!st                                                  pin 8,9;
mast0,mast1,mast2,mast3                                                 pin 4,5,6,7;
!oe                                                     pin 11;

"Outputs
a_le,b2_le,b1_le,c2_le                                  pin 19,18,17,16;
c1_le,d3_le,d2_le,d1_le                                 pin 15,14,13,12;
mreg = [mast3..mast0];


x,ck = .x.,.c.;
neg_le = s3_or_sr1;  "These states cause the negation of the appropriate le signal
                     "once that le signal is asserted.
" Master State Values
I = ^b0000; i = mreg == [0,0,0,0];
A = ^b0001; a = mreg == [0,0,0,1];
B = ^b0101; b = mreg == [0,1,0,1];
C = ^b1101; c = mreg == [1,1,0,1];
D = ^b0100; d = mreg == [0,1,0,0];
E = ^b1100; e = mreg == [1,1,0,0];
```

```
F = ^b1001; f = mreg == [1,0,0,1];
G = ^b1011; g = mreg == [1,0,1,1];
H = ^b1111; h = mreg == [1,1,1,1];
J = ^b1110; j = mreg == [1,1,1,0];
K = ^b0011; k = mreg == [0,0,1,1];
L = ^b0010; l = mreg == [0,0,1,0];
M = ^b0111; m = mreg == [0,1,1,1];
N = ^b0110; n = mreg == [0,1,1,0];
Z = ^b1000; z = mreg == [1,0,0,0];
DC= ^b1010;dc = mreg == [1,0,1,0];


equations

 a_le := at # st # ( a_le & !neg_le & ( i # z # a # f # k )) #  b # c # d # e # h ;
b1_le := at # st # (b1_le & !neg_le & ( i # z # a # f # l )) #  b ;
b2_le := at # st # (b2_le & !neg_le & ( i # z # c # h # l )) #  d # e ;
c1_le := at # st # (c1_le & !neg_le & ( i # z # a # g # m ));
c2_le := at # st # (c2_le & !neg_le & ( i # z # b # g # m # d # dc )) # e # j # c # h;
d1_le := at # st # (d1_le & !neg_le & ( i # z # a # g # n ));
d2_le := at # st # (d2_le & !neg_le & ( i # z # b # g # n # dc));
d3_le := at # st # (d3_le & !neg_le & ( i # z # j # n # e ));


test_vectors
([bclk,!st,!at,neg_le,mreg] -> [!a_le,!b1_le,!b2_le,!c1_le,!c2_le,!d1_le,!d2_le,!d3_le])
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3
 [ ck ,1 ,  1,  0   ,A ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s0
 [ ck ,1 ,  0,  0   ,A ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
 [ ck ,1 ,  1,  0   ,A ] -> [  0  ,  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ];"s1
 [ ck ,1 ,  1,  1   ,A ] -> [  1  ,  1  ,  x  ,  1  ,  x  ,  1  ,  x  ,  x  ];"sr0
 [ ck ,1 ,  1,  0   ,A ] -> [  1  ,  1  ,  x  ,  1  ,  x  ,  1  ,  x  ,  x  ];"sr1
 [ ck ,1 ,  1,  0   ,A ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s0
 [ ck ,1 ,  0,  0   ,A ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
 [ ck ,1 ,  1,  0   ,A ] -> [  0  ,  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ];"s1
 [ ck ,1 ,  1,  0   ,A ] -> [  0  ,  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ];"s2
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3  vector 12
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3
 [ ck ,1 ,  1,  0   ,F ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s0
 [ ck ,1 ,  0,  0   ,F ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
 [ ck ,1 ,  1,  0   ,F ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s1
 [ ck ,1 ,  1,  1   ,F ] -> [  1  ,  1  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"sr0
 [ ck ,1 ,  1,  0   ,F ] -> [  1  ,  1  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"sr1
 [ ck ,1 ,  1,  0   ,F ] -> [  1  ,  1  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s0
 [ ck ,1 ,  0,  0   ,F ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
 [ ck ,1 ,  1,  0   ,F ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s1
 [ ck ,1 ,  1,  0   ,F ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s2
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3  vector 23
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3
 [ ck ,1 ,  1,  0   ,G ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s0
 [ ck ,1 ,  0,  0   ,G ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
 [ ck ,1 ,  1,  0   ,G ] -> [  x  ,  x  ,  x  ,  0  ,  0  ,  0  ,  0  ,  x  ];"s1
 [ ck ,1 ,  1,  1   ,G ] -> [  x  ,  x  ,  x  ,  1  ,  1  ,  1  ,  1  ,  x  ];"sr0
 [ ck ,1 ,  1,  0   ,G ] -> [  x  ,  x  ,  x  ,  1  ,  1  ,  1  ,  1  ,  x  ];"sr1
 [ ck ,1 ,  1,  0   ,G ] -> [  x  ,  x  ,  x  ,  1  ,  1  ,  1  ,  1  ,  x  ];"s0
 [ ck ,1 ,  0,  0   ,G ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
 [ ck ,1 ,  1,  0   ,G ] -> [  x  ,  x  ,  x  ,  0  ,  0  ,  0  ,  0  ,  x  ];"s1
 [ ck ,1 ,  1,  0   ,G ] -> [  x  ,  x  ,  x  ,  0  ,  0  ,  0  ,  0  ,  x  ];"s2
 [ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3 vector 34
```

```
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3
[ ck ,1 , 1, 0  , K ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ ck ,1 , 0, 0  , K ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , K ] -> [ 0 , x , x , x , x , x , x , x ];"s1
[ ck ,1 , 1, 1  , K ] -> [ 1 , x , x , x , x , x , x , x ];"sr0
[ ck ,1 , 1, 0  , K ] -> [ 1 , x , x , x , x , x , x , x ];"sr1
[ ck ,1 , 1, 0  , K ] -> [ 1 , x , x , x , x , x , x , x ];"s0
[ ck ,1 , 0, 0  , K ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , K ] -> [ 0 , x , x , x , x , x , x , x ];"s1
[ ck ,1 , 1, 0  , K ] -> [ 0 , x , x , x , x , x , x , x ];"s2
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3 vector 45
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3
[ ck ,1 , 1, 0  , L ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ ck ,1 , 0, 0  , L ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , L ] -> [ x , 0 , 0 , x , x , x , x , x ];"s1
[ ck ,1 , 1, 1  , L ] -> [ x , 1 , 1 , x , x , x , x , x ];"sr0
[ ck ,1 , 1, 0  , L ] -> [ x , 1 , 1 , x , x , x , x , x ];"sr1
[ ck ,1 , 1, 0  , L ] -> [ x , 1 , 1 , x , x , x , x , x ];"s0
[ ck ,1 , 0, 0  , L ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , L ] -> [ x , 0 , 0 , x , x , x , x , x ];"s1
[ ck ,1 , 1, 0  , L ] -> [ x , 0 , 0 , x , x , x , x , x ];"s2
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3 vector 56
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3
[ ck ,1 , 1, 0  , M ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ ck ,1 , 0, 0  , M ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , M ] -> [ x , x , x , 0 , 0 , x , x , x ];"s1
[ ck ,1 , 1, 1  , M ] -> [ x , x , x , 1 , 1 , x , x , x ];"sr0
[ ck ,1 , 1, 0  , M ] -> [ x , x , x , 1 , 1 , x , x , x ];"sr1
[ ck ,1 , 1, 0  , M ] -> [ x , x , x , 1 , 1 , x , x , x ];"s0
[ ck ,1 , 0, 0  , M ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , M ] -> [ x , x , x , 0 , 0 , x , x , x ];"s1
[ ck ,1 , 1, 0  , M ] -> [ x , x , x , 0 , 0 , x , x , x ];"s2
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3 vector 67
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3
[ ck ,1 , 1, 0  , N ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ ck ,1 , 0, 0  , N ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , N ] -> [ x , x , x , x , x , 0 , 0 , 0 ];"s1
[ ck ,1 , 1, 1  , N ] -> [ x , x , x , x , x , 1 , 1 , 1 ];"sr0
[ ck ,1 , 1, 0  , N ] -> [ x , x , x , x , x , 1 , 1 , 1 ];"sr1
[ ck ,1 , 1, 0  , N ] -> [ x , x , x , x , x , 1 , 1 , 1 ];"s0
[ ck ,1 , 0, 0  , N ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , N ] -> [ x , x , x , x , x , 0 , 0 , 0 ];"s1
[ ck ,1 , 1, 0  , N ] -> [ x , x , x , x , x , 0 , 0 , 0 ];"s2
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3 vector 78
[ ck ,1 , 1, 1  , I ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3
[ ck ,1 , 1, 0  , F ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ ck ,1 , 0, 0  , F ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , F ] -> [ 0 , 0 , x , x , x , x , x , x ];"s1
[ ck ,1 , 1, 1  , F ] -> [ 1 , 1 , x , x , x , x , x , x ];"sr0
[ ck ,1 , 1, 0  , F ] -> [ 1 , 1 , x , x , x , x , x , x ];"sr1
[ ck ,1 , 1, 0  , F ] -> [ 1 , 1 , x , x , x , x , x , x ];"s0
[ ck ,1 , 0, 0  , F ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , F ] -> [ 0 , 0 , x , x , x , x , x , x ];"s1
[ ck ,1 , 1, 0  , F ] -> [ 0 , 0 , x , x , x , x , x , x ];"s3w
[ ck ,1 , 1, 1  , H ] -> [ 0 , x , x , x , x , x , x , x ];"s3
[ ck ,1 , 1, 0  , H ] -> [ 0 , x , 1 , x , x , x , x , x ];"s0  vector 90
[ ck ,1 , 0, 0  , H ] -> [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ ck ,1 , 1, 0  , H ] -> [ 0 , x , 0 , x , x , x , x , x ];"s1
[ ck ,1 , 1, 1  , H ] -> [ 0 , x , 1 , x , x , x , x , x ];"sr0
[ ck ,1 , 1, 0  , H ] -> [ 0 , x , 1 , x , x , x , x , x ];"sr1
```

```
[ck,1, 1, 0 ,H ]->[ 0 , x , 1 , x , x , x , x , x ];"s0
[ck,1, 0, 0 ,H ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,H ]->[ 0 , x , 0 , x , x , x , x , x ];"s1
[ck,1, 1, 0 ,H ]->[ 0 , x , 0 , x , x , x , x , x ];"s2
[ck,1, 1, 1 ,I ]->[ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3
[ck,1, 1, 1 ,I ]->[ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3 vector 100
[ck,1, 1, 0 ,G ]->[ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ck,1, 0, 0 ,G ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,G ]->[ x , x , x , 0 , 0 , 0 , 0 , x ];"s1
[ck,1, 1, 1 ,G ]->[ x , x , x , 1 , 1 , 1 , 1 , x ];"sr0
[ck,1, 1, 0 ,G ]->[ x , x , x , 1 , 1 , 1 , 1 , x ];"sr1
[ck,1, 1, 0 ,G ]->[ x , x , x , 1 , 1 , 1 , 1 , x ];"s0
[ck,1, 0, 0 ,G ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,G ]->[ x , x , x , 0 , 0 , 0 , 0 , x ];"s1
[ck,1, 1, 0 ,G ]->[ x , x , x , 0 , 0 , 0 , 0 , x ];"s3w
[ck,1, 1, 1 ,J ]->[ x , x , x , x , 0 , x , x , x ];"s3
[ck,1, 1, 0 ,J ]->[ x , x , x , x , 0 , x , x , 1 ];"s0
[ck,1, 0, 0 ,J ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,J ]->[ x , x , x , x , 0 , x , x , 0 ];"s1
[ck,1, 1, 1 ,J ]->[ x , x , x , x , 0 , x , x , 1 ];"sr0
[ck,1, 1, 0 ,J ]->[ x , x , x , x , 0 , x , x , 1 ];"sr1
[ck,1, 1, 0 ,J ]->[ x , x , x , x , 0 , x , x , 1 ];"s0
[ck,1, 0, 0 ,J ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,J ]->[ x , x , x , x , 0 , x , x , 0 ];"s1
[ck,1, 1, 0 ,J ]->[ x , x , x , x , 0 , x , x , 0 ];"s2
[ck,1, 1, 1 ,I ]->[ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3  vector 120
[ck,1, 1, 1 ,I ]->[ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s3  121
[ck,1, 1, 0 ,A ]->[ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"s0
[ck,1, 0, 0 ,A ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,A ]->[ 0 , x , x , x , x , x , x , x ];"s1
[ck,1, 1, 1 ,A ]->[ 1 , x , x , x , x , x , x , x ];"sr0
[ck,1, 1, 0 ,A ]->[ 1 , x , x , x , x , x , x , x ];"sr1
[ck,1, 1, 0 ,A ]->[ 1 , x , x , x , x , x , x , x ];"s0
[ck,1, 0, 0 ,A ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,A ]->[ 0 , x , x , x , x , x , x , x ];"s1
[ck,1, 1, 0 ,A ]->[ 0 , x , x , x , x , x , x , x ];"s3w
[ck,1, 1, 1 ,C ]->[ 0 , x , x , x , x , x , x , x ];"s3
[ck,1, 1, 0 ,C ]->[ 0 , x , 1 , x , x , x , x , x ];"s0
[ck,1, 0, 0 ,C ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,C ]->[ 0 , x , 0 , x , x , x , x , x ];"s1
[ck,1, 1, 1 ,C ]->[ 0 , x , 1 , x , x , x , x , x ];"sr0
[ck,1, 1, 0 ,C ]->[ 0 , x , 1 , x , x , x , x , x ];"sr1
[ck,1, 1, 0 ,C ]->[ 0 , x , 1 , x , x , x , x , x ];"s0
[ck,1, 0, 0 ,C ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,C ]->[ 0 , x , 0 , x , x , x , x , x ];"s1
[ck,1, 1, 0 ,C ]->[ 0 , x , 0 , x , x , x , x , x ];"s3W  140
[ck,1, 1, 1 ,D ]->[ 0 , x , 0 , x , x , x , x , x ];"s3
[ck,1, 1, 0 ,D ]->[ 0 , x , 0 , x , 1 , x , x , 1 ];"s0
[ck,1, 0, 0 ,D ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,D ]->[ 0 , x , 0 , x , 0 , x , x , x ];"s1
[ck,1, 1, 1 ,D ]->[ 0 , x , 0 , x , 1 , x , x , x ];"sr0
[ck,1, 1, 0 ,D ]->[ 0 , x , 0 , x , 1 , x , x , x ];"sr1
[ck,1, 1, 0 ,D ]->[ 0 , x , 0 , x , 1 , x , x , x ];"s0
[ck,1, 0, 0 ,D ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,D ]->[ 0 , x , 0 , x , 0 , x , x , x ];"s1
[ck,1, 1, 0 ,D ]->[ 0 , x , 0 , x , 0 , x , x , x ];"s3w  150
[ck,1, 1, 1 ,E ]->[ 0 , x , 0 , x , 0 , x , x , x ];"s3
[ck,1, 1, 0 ,E ]->[ 0 , x , 0 , x , 0 , x , x , 1 ];"s0
[ck,1, 0, 0 ,E ]->[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ];"s0
[ck,1, 1, 0 ,E ]->[ 0 , x , 0 , x , 0 , x , x , 0 ];"s1
```

```
[ ck ,1 ,  1,  1   ,E ] -> [  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ,  1  ];"sr0
[ ck ,1 ,  1,  0   ,E ] -> [  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ,  1  ];"sr1
[ ck ,1 ,  1,  0   ,E ] -> [  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ,  1  ];"s0
[ ck ,1 ,  0,  0   ,E ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
[ ck ,1 ,  1,  0   ,E ] -> [  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ,  0  ];"s1
[ ck ,1 ,  1,  0   ,E ] -> [  0  ,  x  ,  0  ,  x  ,  0  ,  x  ,  x  ,  0  ];"s2
[ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3  161

[ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3
[ ck ,1 ,  1,  0   ,A ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s0
[ ck ,1 ,  0,  0   ,A ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
[ ck ,1 ,  1,  0   ,A ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s1
[ ck ,1 ,  1,  1   ,A ] -> [  1  ,  1  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"sr0
[ ck ,1 ,  1,  0   ,A ] -> [  1  ,  1  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"sr1
[ ck ,1 ,  1,  0   ,A ] -> [  1  ,  1  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s0
[ ck ,1 ,  0,  0   ,A ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
[ ck ,1 ,  1,  0   ,A ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s1
[ ck ,1 ,  1,  0   ,A ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s3w
[ ck ,1 ,  1,  1   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  x  ,  x  ,  x  ,  x  ];"s3
[ ck ,1 ,  1,  0   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  1  ,  x  ,  1  ,  x  ];"s0
[ ck ,1 ,  0,  0   ,B ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
[ ck ,1 ,  1,  0   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  0  ,  x  ,  0  ,  x  ];"s1
[ ck ,1 ,  1,  1   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  1  ,  x  ,  1  ,  x  ];"sr0
[ ck ,1 ,  1,  0   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  1  ,  x  ,  1  ,  x  ];"sr1
[ ck ,1 ,  1,  0   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  1  ,  x  ,  1  ,  x  ];"s0
[ ck ,1 ,  0,  0   ,B ] -> [  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ,  0  ];"s0
[ ck ,1 ,  1,  0   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  0  ,  x  ,  0  ,  x  ];"s1
[ ck ,1 ,  1,  0   ,B ] -> [  0  ,  0  ,  x  ,  x  ,  0  ,  x  ,  0  ,  x  ];"s2
[ ck ,1 ,  1,  1   ,I ] -> [  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ,  1  ];"s3

end LATCH_EN;
```

7.7.5 MC68040 Bus Translator Bus Arbitration PAL

```
module BUSARB flag '-r3';
title '68040 Bus-Translator Bus Arbitration PAL'

busarb device 'P16R6';

" Inputs
bclk                                pin 1;
!br_30,!bgack,!br_40,!bb            pin 2,3,4,5;
!lock,!locke,!reset                 pin 6,7,8;

" Outputs
!bg_40                              pin 12;
!bg_30,tristate                     pin 13,14;


bs0,bs1,bs2,bs3                     pin 18,17,16,15;


" Set assignments state machines
```

```
bus_stat  = [bs0,bs1,bs2,bs3];

" Macro Assignments for each state
STATE0  macro { with bg_30 := 0; tristate := 0; endwith }
STATE1  macro { with bg_30 := 1; tristate := 1; endwith }
STATE2  macro { with bg_30 := 1; tristate := 1; endwith }
STATE3  macro { with bg_30 := 0; tristate := 1; endwith }
STATE4  macro { with bg_30 := 0; tristate := 1; endwith }
STATE5  macro { with bg_30 := 1; tristate := 1; endwith }
STATE6  macro { with bg_30 := 1; tristate := 1; endwith }
STATE7  macro { with bg_30 := 0; tristate := 0; endwith }
STATE8  macro { with bg_30 := 0; tristate := 0; endwith }
STATE9  macro { with bg_30 := 0; tristate := 0; endwith }
STATE10 macro { with bg_30 := 0; tristate := 0; endwith }

" Constant assignments
x,ck = .x.,.c.;


" State Values
B00 = ^b1111; b00 = bus_stat == [1,1,1,1];
B01 = ^b1001; b01 = bus_stat == [1,0,0,1];
B02 = ^b1101; b02 = bus_stat == [1,1,0,1];
B03 = ^b1100; b03 = bus_stat == [1,1,0,0];
B04 = ^b0101; b04 = bus_stat == [0,1,0,1];
B05 = ^b1011; b05 = bus_stat == [1,0,1,1];
B06 = ^b0011; b06 = bus_stat == [0,0,1,1];
B07 = ^b0110; b07 = bus_stat == [0,1,1,0];
B08 = ^b0111; b08 = bus_stat == [0,1,1,1];
B09 = ^b1010; b09 = bus_stat == [1,0,1,0];
B10 = ^b1110; b10 = bus_stat == [1,1,1,0];

state_diagram bus_stat

   State B00:
     case
          bgack & !reset                          :B04 STATE4;
        ( !bgack &  br_30 & !reset)               :B01 STATE1;
         ( !bgack & !br_30 &  br_40 & !reset)     :B07 STATE7;
         ( !bgack & !br_30 &  !br_40 & !reset)    :B00 STATE0;
        reset                                     :B00 STATE0;
     endcase;

   State B01:
     case
          reset                    :B00 STATE0;
          !reset                   :B02 STATE2;
     endcase;

   State B02:
     case
        ( bgack & !reset)                  :B03 STATE3;
         ( !bgack &  br_30 & !reset)       :B02 STATE2;
         ( !bgack & !br_30 & !reset)       :B03 STATE3;
        reset                              :B00 STATE0;
     endcase;

   State B03:
     case
        !reset                   :B04 STATE4;
```

```
      reset                        :B00 STATE0;
    endcase;

  State B04:
    case
       ( br_30 & !reset)                  :B05 STATE5;
       ( !br_30 &  bgack & !reset)        :B04 STATE4;
       ( !br_30 & !bgack & !reset)        :B00 STATE0;
       reset                              :B00 STATE0;
    endcase;

  State B05:
    case
       !reset                       :B06 STATE6;
       reset                        :B00 STATE0;
    endcase;


  State B06:
    case
       ( !br_30  & !reset)                 :B03 STATE3;
       (  br_30 &  bgack & !reset)        :B06 STATE6;
       (  br_30 & !bgack & !reset)        :B02 STATE2;
       reset                              :B00 STATE0;
    endcase;

  State B07:
    case
       !reset                       :B08 STATE8;
       reset                        :B00 STATE0;
    endcase;

  State B08:
    case

       br_30 & !lock & !reset             :B09 STATE9;
       br_30 &  lock & !reset             :B10 STATE10;
       !br_30 & br_40 & !reset            :B08 STATE8;
       !br_30 & !br_40 & !reset           :B09 STATE9;
       reset                              :B00 STATE0;
    endcase;

  State B09:
    case
       bb & !reset                  :B09 STATE9;
       !bb & !reset                 :B00 STATE0;
       reset                        :B00 STATE0;
    endcase;

  State B10:
    case
       locke & !reset               :B09 STATE9;
       !locke & !reset              :B10 STATE10;
       reset                        :B00 STATE0;
    endcase;

equations

bg_40 = b07 # b08 # b10;
```

```
test_vectors
([bclk,!br_30,!br_40,!bgack,!bb,!lock,!locke,!reset] -> [!bg_30,!bg_40,!tristate,bus_stat])
[ ck,    1  ,   1  ,   1  , 1, 1  ,   1  ,   0  ] -> [  1  ,   1  ,   1  ,   B00  ];
[ ck,    1  ,   1  ,   1  , 1, 1  ,   1  ,   1  ] -> [  1  ,   1  ,   1  ,   B00  ];
[ ck,    1  ,   0  ,   1  , 1, 1  ,   1  ,   1  ] -> [  1  ,   0  ,   1  ,   B07  ];
[ ck,    x  ,   x  ,   x  , x, x  ,   x  ,   1  ] -> [  1  ,   0  ,   1  ,   B08  ];
[ ck,    1  ,   0  ,   x  , x, x  ,   x  ,   1  ] -> [  1  ,   0  ,   1  ,   B08  ];
[ ck, 0  ,  x , x  , x, 0  , 1  , 1  ] -> [  1  , 0  ,   1  ,   B10  ];
[ ck, x  ,  x , x  , x, x  , 0  , 1  ] -> [  1  , 1  ,   1  ,   B09  ];
[ ck, x  ,  x , x  , 0, x  , x  , 1  ] -> [  1  , 1  ,   1  ,   B09  ];
[ ck, x  ,  x , x  , 1, x  , x  , 1  ] -> [  1  , 1  ,   1  ,   B00  ];"vector 9
[ ck, 0  ,  x , 1  , x, x  , x  , 1  ] -> [  0  , 1  ,   0  ,   B01  ];
[ ck, 0  ,  x , 1  , x, x  , x  , 1  ] -> [  0  , 1  ,   0  ,   B02  ];
[ ck, 0  ,  x , 0  , x, x  , x  , 1  ] -> [  1  , 1  ,   0  ,   B03  ];
[ ck, 1  ,  x , 0  , x, x  , x  , 1  ] -> [  1  , 1  ,   0  ,   B04  ];
[ ck, 0  ,  x , 0  , x, x  , x  , 1  ] -> [  0  , 1  ,   0  ,   B05  ];
[ ck, 1  ,  x , 0  , x, x  , x  , 1  ] -> [  0  , 1  ,   0  ,   B06  ];"vector 15
[ ck, 0  ,  x , 0  , x, x  , x  , 1  ] -> [  0  , 1  ,   0  ,   B06  ];
[ ck, 1  ,  x , 0  , x, x  , x  , 1  ] -> [  1  , 1  ,   0  ,   B03  ];
[ ck, 1  ,  x , 0  , x, x  , x  , 1  ] -> [  1  , 1  ,   0  ,   B04  ];
[ ck, 1  ,  x , 1  , x, x  , x  , 1  ] -> [  1  , 1  ,   1  ,   B00  ];"vector 19
[ ck, 1  ,  1 , 1  , 1, 1  , x  , 1  ] -> [  1  , 1  ,   1  ,   B00  ];



end BUSARB;
```

7.7.6 MC68040 Bus Translator Output Enable PAL

```
 module OUT_EN flag '-r3';
title '68040 Bus-Translator Output Enable PAL'

out_en device 'P20R8';

" Inputs
bclk                                    pin 1;
mast0,mast1,mast2,mast3                 pin 2,3,4,5;
!s2w,s1,rd                               pin 6,7,8;
!rberr,rdsack0,rdsack1                        pin 9,10,11;

" Outputs
!aoe,!b2oe,!b1oe,!c2oe,!c1oe,!d3oe,!d2oe,!d1oe          pin 22,21,20,19,18,17,16,15;

" Set assignments
mreg = [mast3..mast0];
DSACK  = [rdsack1,rdsack0];

" Constant assignments
x,ck = .x.,.c.;

" Master State Values
I = ^b0000; i = mreg == [0,0,0,0];
A = ^b0001; a = mreg == [0,0,0,1];
B = ^b0101; b = mreg == [0,1,0,1];
C = ^b1101; c = mreg == [1,1,0,1];
D = ^b0100; d = mreg == [0,1,0,0];
E = ^b1100; e = mreg == [1,1,0,0];
F = ^b1001; f = mreg == [1,0,0,1];
```

```
G = ^b1011; g = mreg == [1,0,1,1];
H = ^b1111; h = mreg == [1,1,1,1];
J = ^b1110; j = mreg == [1,1,1,0];
K = ^b0011; k = mreg == [0,0,1,1];
L = ^b0010; l = mreg == [0,0,1,0];
M = ^b0111; m = mreg == [0,1,1,1];
N = ^b0110; n = mreg == [0,1,1,0];
Z = ^b1000; z = mreg == [1,0,0,0];
DC= ^b1010;dc = mreg == [1,0,1,0];


wr  macro { !rd };
byte macro {  rdsack0 & !rdsack1 & !rberr };
word macro { !rdsack0 &  rdsack1 & !rberr };
long macro {  rdsack0 &  rdsack1 & !rberr };



equations


aoe   :=  (
            wr & ( a # f # k ) & !s1
          # rd & (
                   ( byte # word # long ) & s1
                 # (  aoe & s2w )
               )
         );


b2oe :=    (
            wr & ( c # h # l ) & !s1
          # rd & (
                     byte & s1
                 # (  b2oe & s2w )
               )
         );

b1oe :=    (
            wr & ( a # f # h # k # l # m) & !s1
          # rd & (
                   ( word # long ) & s1
                 # (  b1oe & s2w )
               )
         );

c2oe :=    (
            wr & ( d # g # m # b ) & !s1
          # rd & (
                   ( byte # word ) & s1
                 # (  c2oe & s2w )
               )
         );

c1oe :=    (
            wr &  !(i # z) & !s1
          # rd & (
                     long & s1
```

```
                # ( c1oe & s2w )
            )
        );

d3oe :=   (
            wr & ( e # j # n ) & !s1
          # rd & (
                    byte & s1
                # ( d3oe & s2w )
            )
        );

d2oe :=   (
            wr & (c # b # d # e # g # j # n ) & !s1
          # rd & (
                    word & s1
                # ( d2oe & s2w )
            )
        );

d1oe :=   (
            wr & !( i # z ) & !s1
          # rd & (
                    long & s1
                # ( d1oe & s2w )
            )
        );

test_vectors
([bclk, rd,mreg, s1, s2w,DSACK,rberr] -> [!aoe,!b1oe,!b2oe,!c1oe,!c2oe,!d1oe,!d2oe,!d3oe])
"read test
[.p. , 1 , I , 0, 0 , x , x ] -> [ x , x , x , x , x , x , x , x ];
[ ck , 1 , I , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , A , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , A , 1, 0 , 3 , 0 ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 1 , A , 0, 1 , x , x ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 1 , I , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];

[ ck , 1 , A , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , A , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , A , 1, 0 , 2 , 0 ] -> [ 0 , 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ ck , 1 , A , 0, 1 , x , x ] -> [ 0 , 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ ck , 1 , B , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"11

[ ck , 1 , B , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , B , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , B , 1, 0 , 2 , 0 ] -> [ 0 , 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ ck , 1 , B , 0, 1 , x , x ] -> [ 0 , 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ ck , 1 , I , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"16

[ ck , 1 , A , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , A , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , A , 1, 0 , 1 , 0 ] -> [ 0 , 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ ck , 1 , A , 0, 1 , x , x ] -> [ 0 , 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ ck , 1 , C , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"21

[ ck , 1 , C , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , C , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , C , 1, 0 , 1 , 0 ] -> [ 0 , 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ ck , 1 , C , 0, 1 , x , x ] -> [ 0 , 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
```

```
[ck,1,D , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"26

[ck,1,D , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,D , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,D , 1, 0, 1 , 0 ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,D , 0, 1, x , x ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,E , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"31

[ck,1,E , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,E , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,E , 1, 0, 1 , 0 ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,E , 0, 1, x , x ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"36

[ck,1,F , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,F , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,F , 1, 0, 3 , 0 ]->[ 0, 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ck,1,F , 0, 1, x , x ]->[ 0, 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"41

[ck,1,F , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,F , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,F , 1, 0, 2 , 0 ]->[ 0, 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ck,1,F , 0, 1, x , x ]->[ 0, 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"46

[ck,1,F , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,F , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,F , 1, 0, 1 , 0 ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,F , 0, 1, x , x ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,H , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,H , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,H , 1, 0, 1 , 0 ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,H , 0, 1, x , x ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"55

[ck,1,G , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,G , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,G , 1, 0, 3 , 0 ]->[ 0, 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ck,1,G , 0, 1, x , x ]->[ 0, 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"60

[ck,1,G , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,G , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,G , 1, 0, 2 , 0 ]->[ 0, 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ck,1,G , 0, 1, x , x ]->[ 0, 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"65

[ck,1,G , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,G , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,G , 1, 0, 1 , 0 ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,G , 0, 1, x , x ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,J , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"70
[ck,1,J , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ck,1,J , 1, 0, 1 , 0 ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,J , 0, 1, x , x ]->[ 0, 1 , 0 , 1 , 0 , 1 , 1 , 0 ];
[ck,1,I , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];

[ck,1,K , 0, 0, x , x ]->[ 1, 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"75
```

```
[ck,1,K, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,K, 1, 0, 1, 0]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,K, 0, 1, x, x]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,K, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"80
[ck,1,K, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,K, 1, 0, 2, 0]->[ 0, 0, 1, 1, 0, 1, 0, 1];
[ck,1,K, 0, 1, x, x]->[ 0, 0, 1, 1, 0, 1, 0, 1];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,K, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"85
[ck,1,K, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,K, 1, 0, 3, 0]->[ 0, 0, 1, 0, 1, 0, 1, 1];
[ck,1,K, 0, 1, x, x]->[ 0, 0, 1, 0, 1, 0, 1, 1];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,L, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"90
[ck,1,L, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,L, 1, 0, 1, 0]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,L, 0, 1, x, x]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,L, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"95
[ck,1,L, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,L, 1, 0, 2, 0]->[ 0, 0, 1, 1, 0, 1, 0, 1];
[ck,1,L, 0, 1, x, x]->[ 0, 0, 1, 1, 0, 1, 0, 1];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,L, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"100
[ck,1,L, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,L, 1, 0, 3, 0]->[ 0, 0, 1, 0, 1, 0, 1, 1];
[ck,1,L, 0, 1, x, x]->[ 0, 0, 1, 0, 1, 0, 1, 1];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,M, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"105
[ck,1,M, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,M, 1, 0, 1, 0]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,M, 0, 1, x, x]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,M, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"110
[ck,1,M, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,M, 1, 0, 2, 0]->[ 0, 0, 1, 1, 0, 1, 0, 1];
[ck,1,M, 0, 1, x, x]->[ 0, 0, 1, 1, 0, 1, 0, 1];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,M, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"115
[ck,1,M, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,M, 1, 0, 3, 0]->[ 0, 0, 1, 0, 1, 0, 1, 1];
[ck,1,M, 0, 1, x, x]->[ 0, 0, 1, 0, 1, 0, 1, 1];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,N, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"120
[ck,1,N, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];
[ck,1,N, 1, 0, 1, 0]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,N, 0, 1, x, x]->[ 0, 1, 0, 1, 0, 1, 1, 0];
[ck,1,I, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];

[ck,1,N, 0, 0, x, x]->[ 1, 1, 1, 1, 1, 1, 1, 1];"125
```

```
[ ck , 1 , N , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , N , 1, 0 , 2 , 0 ] -> [ 0 , 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ ck , 1 , N , 0, 1 , x , x ] -> [ 0 , 0 , 1 , 1 , 0 , 1 , 0 , 1 ];
[ ck , 1 , I , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];

[ ck , 1 , N , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"130
[ ck , 1 , N , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];
[ ck , 1 , N , 1, 0 , 3 , 0 ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 1 , N , 0, 1 , x , x ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 1 , I , 0, 0 , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];

"write vectors
[ ck , 0 , I , 0, x , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];"135
[ ck , 0 , A , 0, x , x , x ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 0 , B , 0, x , x , x ] -> [ 1 , 1 , 1 , 0 , 0 , 0 , 0 , 1 ];
[ ck , 0 , C , 0, x , x , x ] -> [ 1 , 1 , 0 , 0 , 1 , 0 , 0 , 1 ];
[ ck , 0 , D , 0, x , x , x ] -> [ 1 , 1 , 1 , 0 , 0 , 0 , 0 , 1 ];
[ ck , 0 , E , 0, x , x , x ] -> [ 1 , 1 , 1 , 0 , 1 , 0 , 0 , 0 ];
[ ck , 0 , F , 0, x , x , x ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 0 , G , 0, x , x , x ] -> [ 1 , 1 , 1 , 0 , 0 , 0 , 0 , 1 ];
[ ck , 0 , H , 0, x , x , x ] -> [ 1 , 0 , 0 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 0 , J , 0, x , x , x ] -> [ 1 , 1 , 1 , 0 , 1 , 0 , 0 , 0 ];
[ ck , 0 , K , 0, x , x , x ] -> [ 0 , 0 , 1 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 0 , L , 0, x , x , x ] -> [ 1 , 0 , 0 , 0 , 1 , 0 , 1 , 1 ];
[ ck , 0 , M , 0, x , x , x ] -> [ 1 , 0 , 1 , 0 , 0 , 0 , 1 , 1 ];
[ ck , 0 , N , 0, x , x , x ] -> [ 1 , 1 , 1 , 0 , 1 , 0 , 0 , 0 ];
[ ck , 0 , Z , 0, x , x , x ] -> [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 ];

end OUT_EN;
```

7.8 TIMING EXAMPLES

Figures 7-20—7-26 show example timing diagrams illustrating the relative timing among various signals. Figure 7-20 shows a retry example of a long-word read to a word port with no wait states. Figure 7-21 shows a halt example of a long word read to a word port with no wait states. Figure 7-22 shows a bus error timing example of a bus read at address 0 to a long word port with no wait states. Figures 7-23—26 show different read/write timing examples.

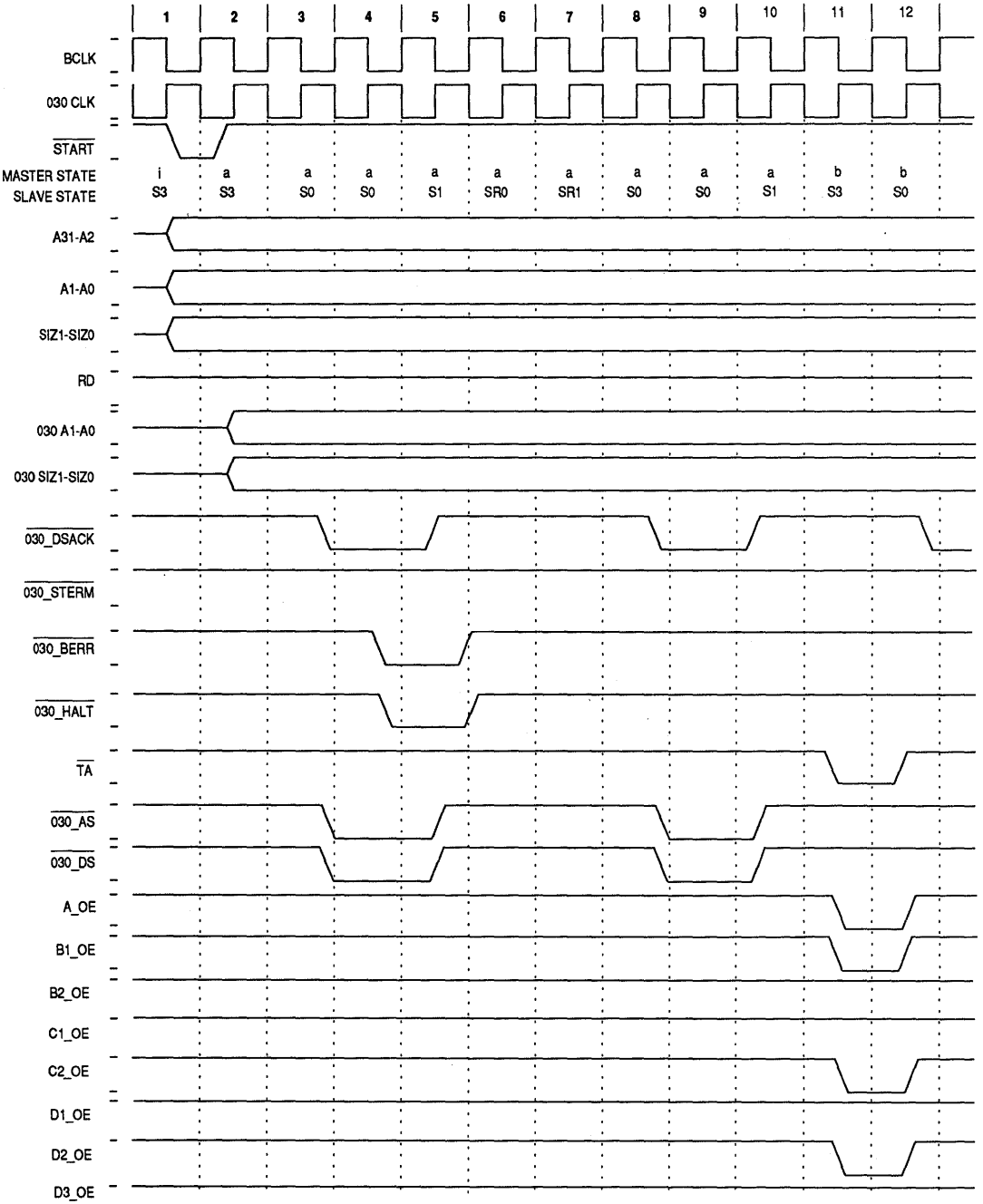


Figure 7-20. Retry Timing Example (Sheet 1 of 2)

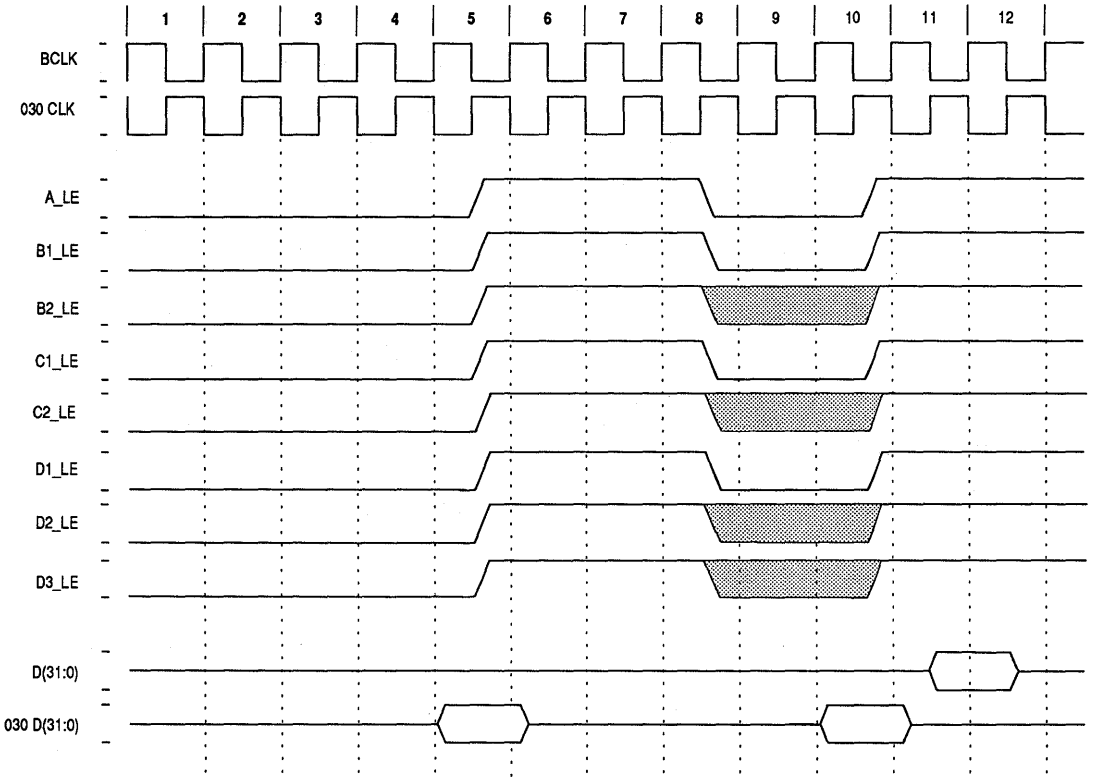


Figure 7-20. Retry Timing Example (Sheet 2 of 2)

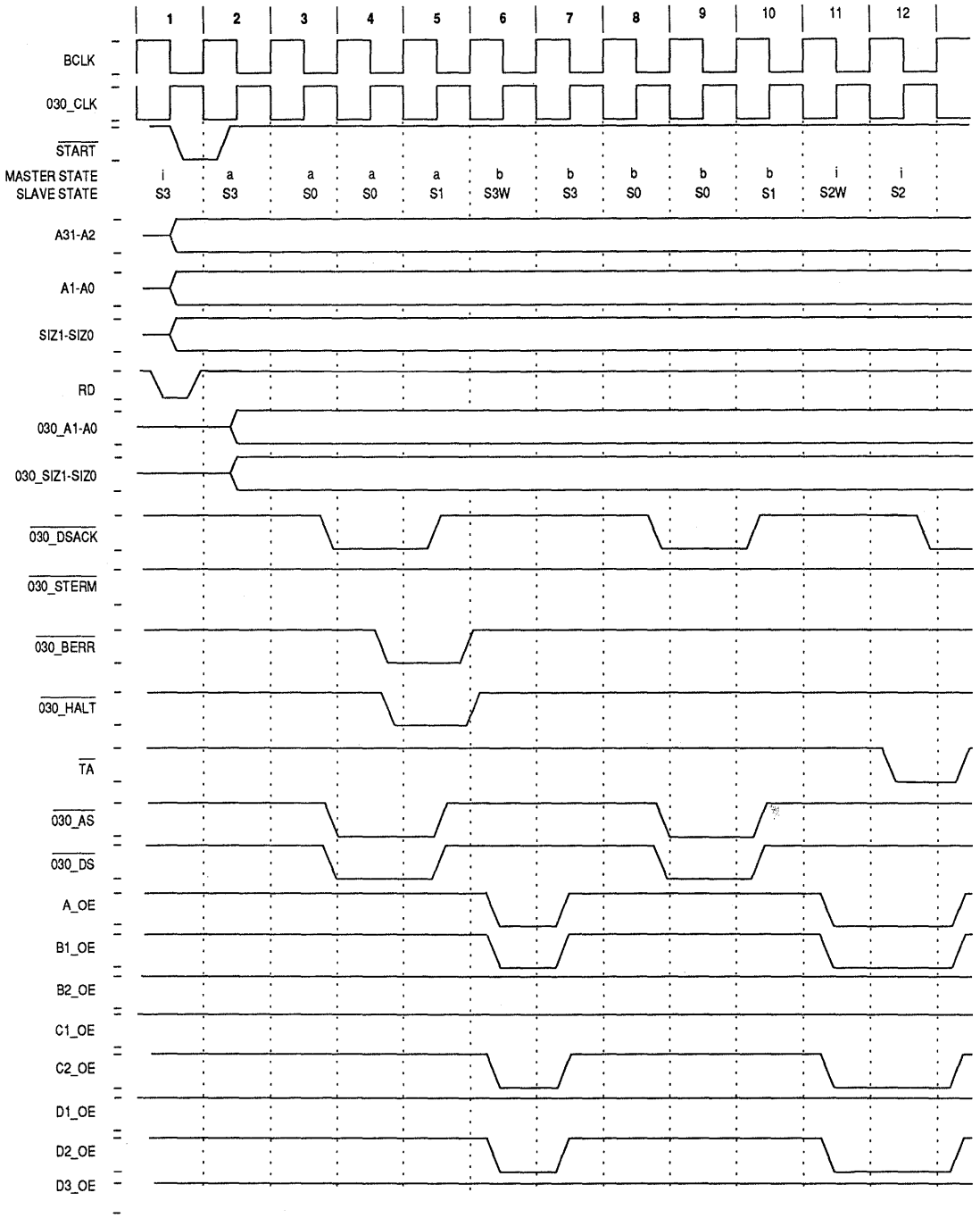


Figure 7-21. HALT Timing Example (Sheet 1 of 2)

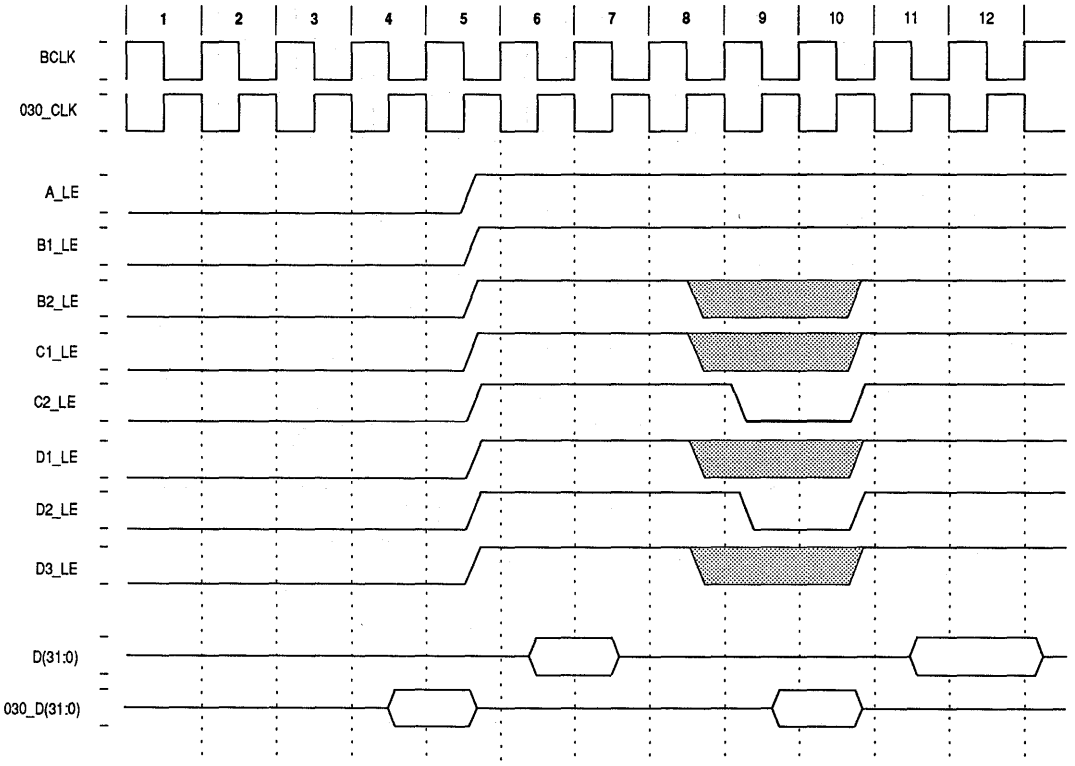


Figure 7-21. HALT Timing Example (Sheet 2 of 2)

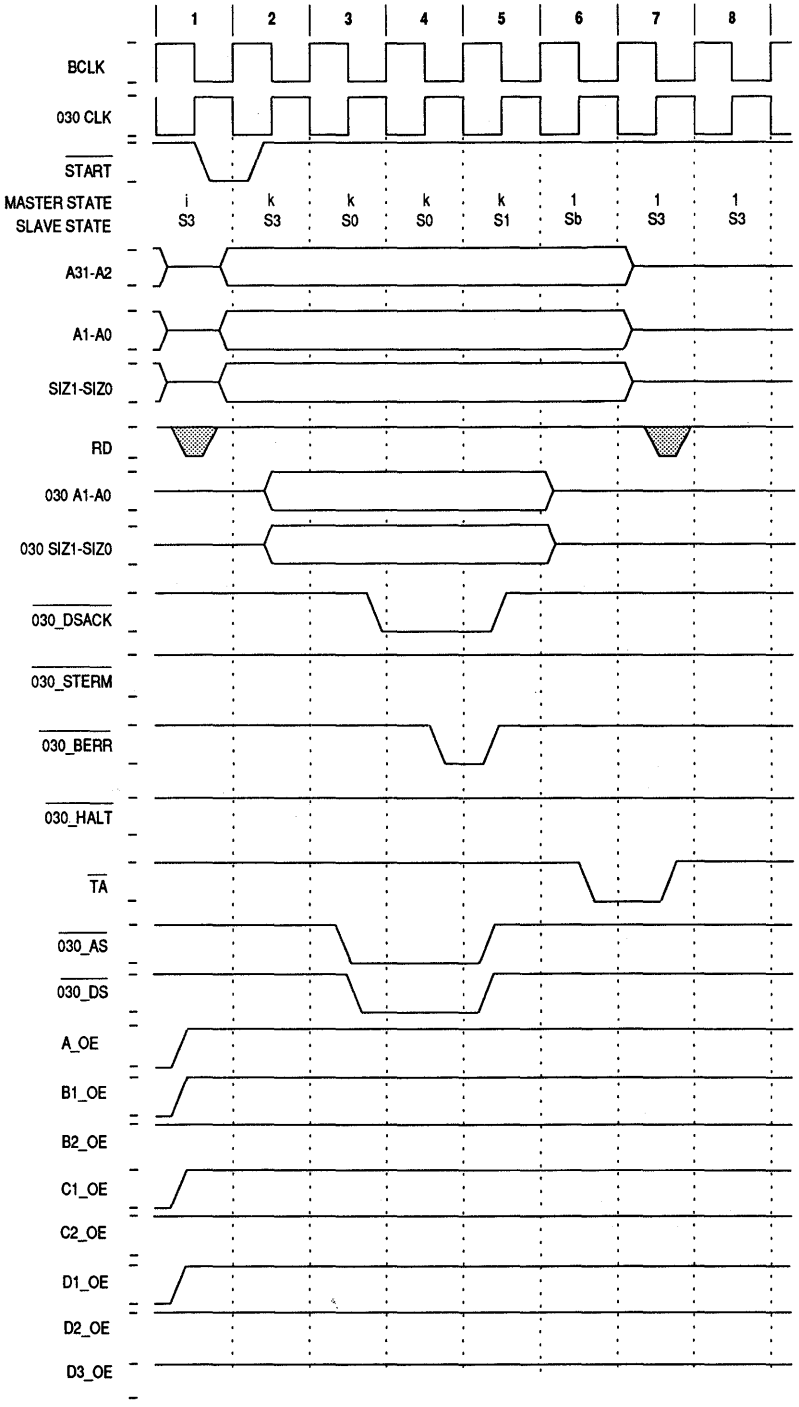


Figure 7-22. Bus Error Timing Example (Sheet 1 of 2)

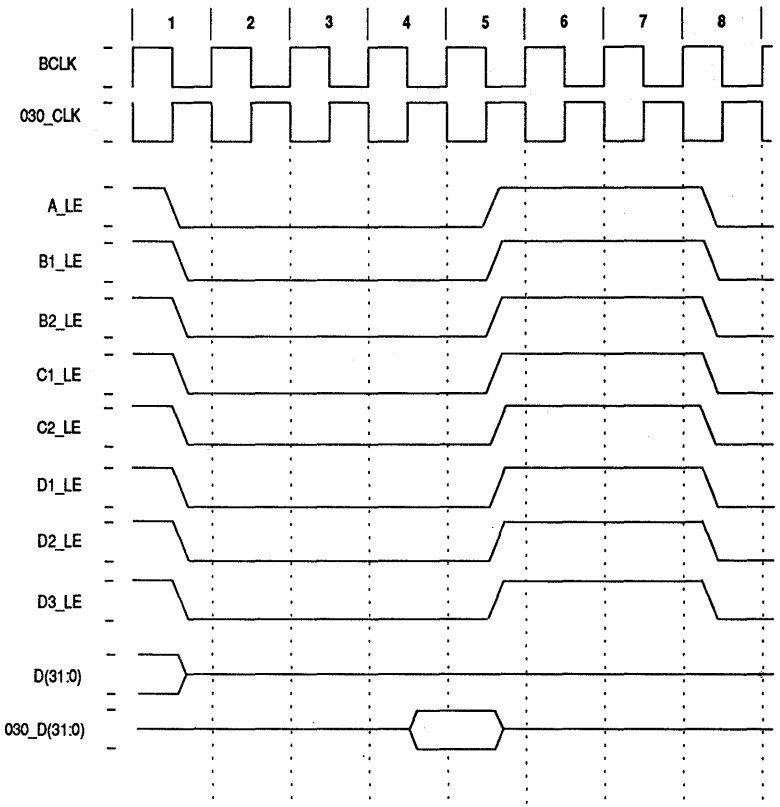


Figure 7-22. Bus Error Timing Example (Sheet 2 of 2)

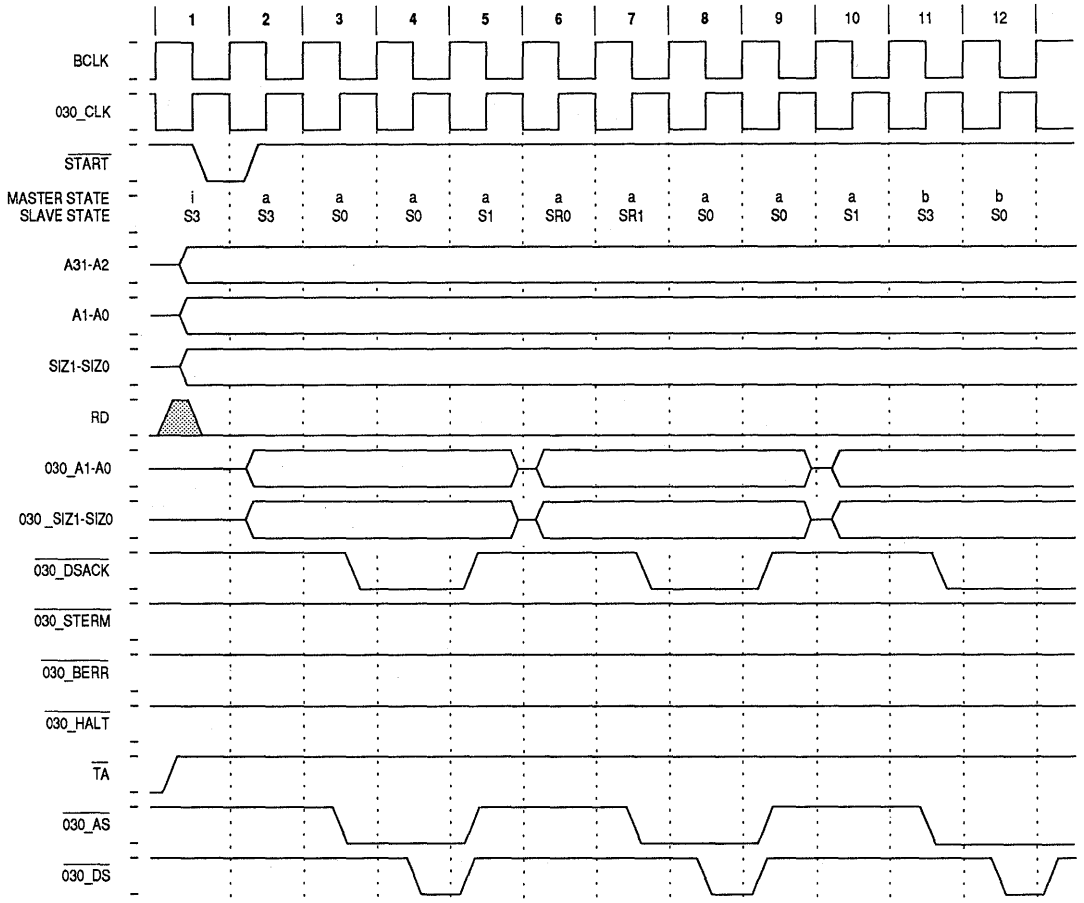


Figure 7-23. Long-Word Write to Word Port Example (Sheet 1 of 2)

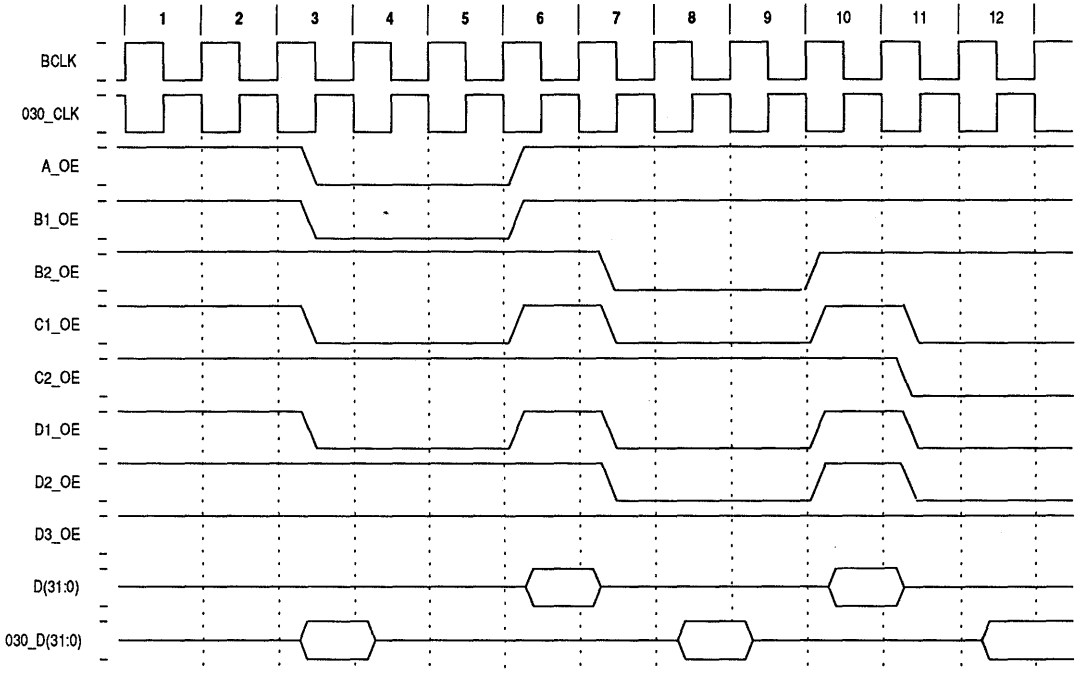


Figure 7-23. Long-Word Write to Word Port Example (Sheet 2 of 2)

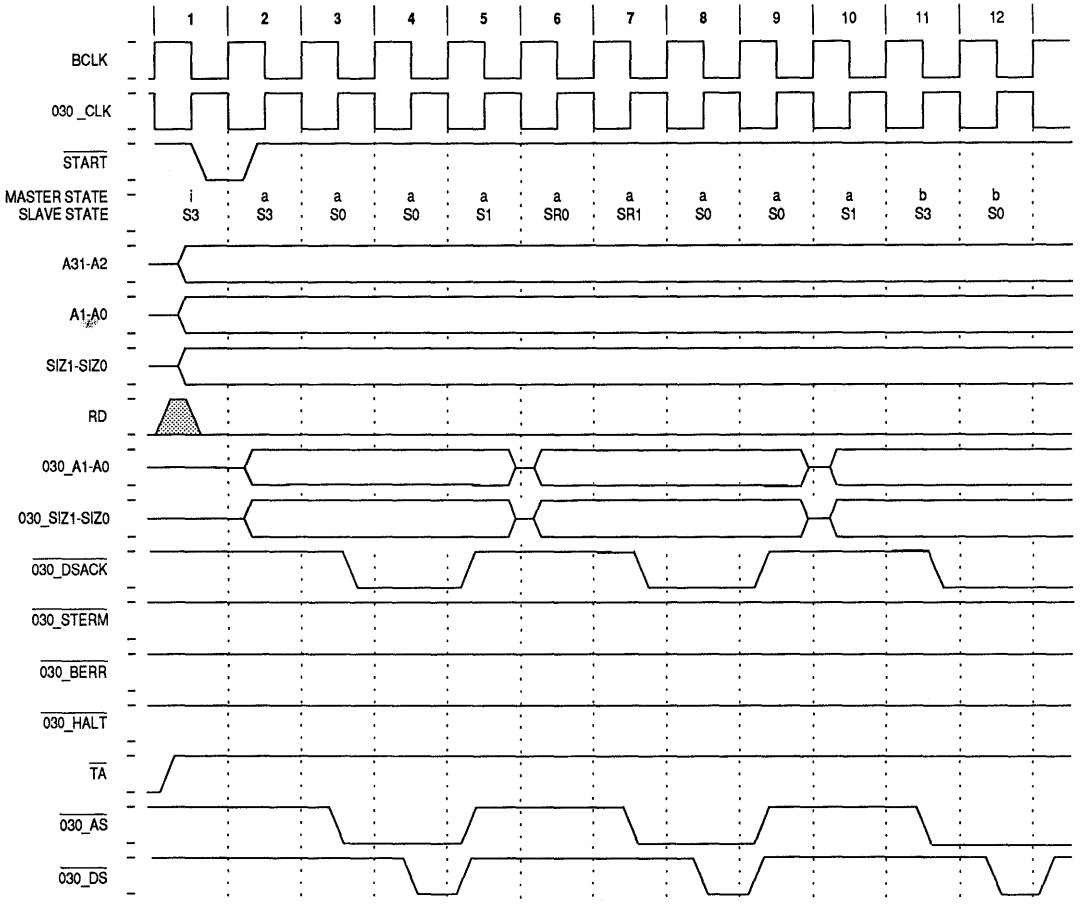


Figure 7-24. Word Write to Byte Port Example (Sheet 1 of 2)

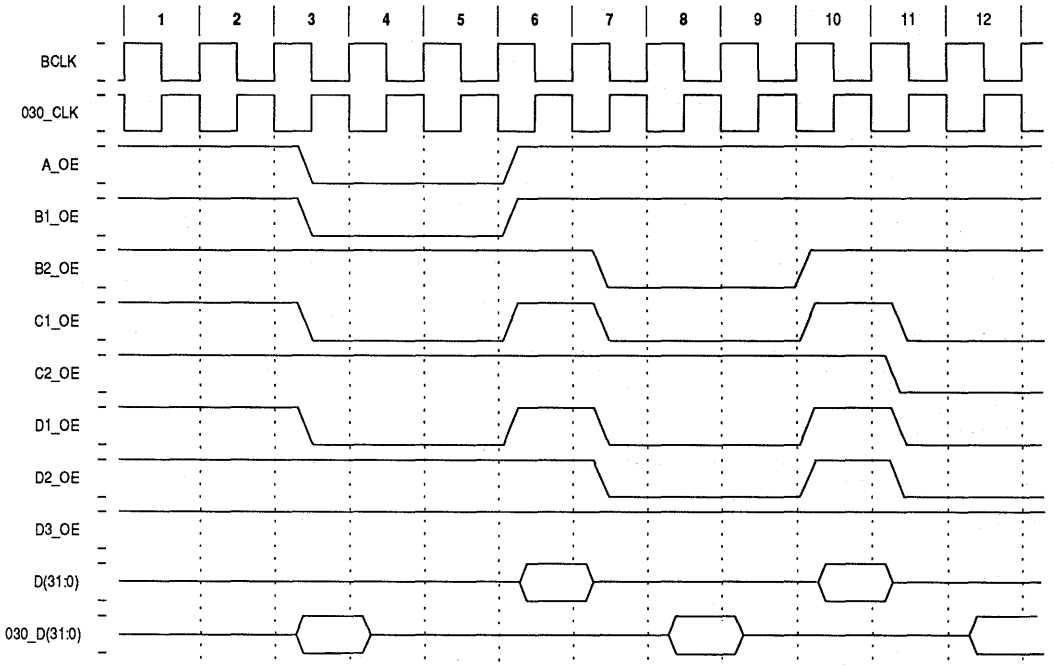


Figure 7-24. Word Write to Byte Port Example (Sheet 2 of 2)

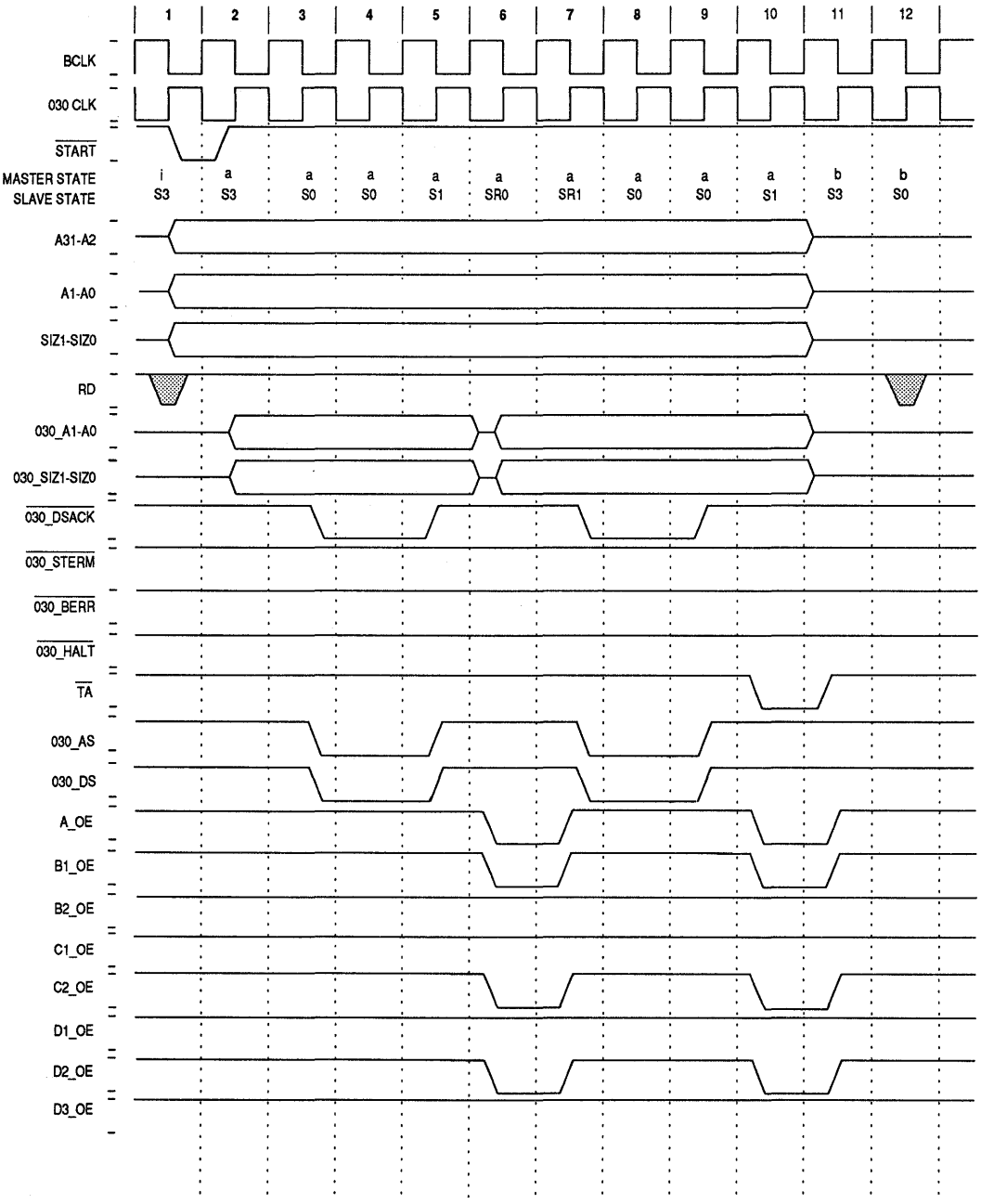


Figure 7-25. Long-Word Read, to Word Port Example (Sheet 1 of 2)

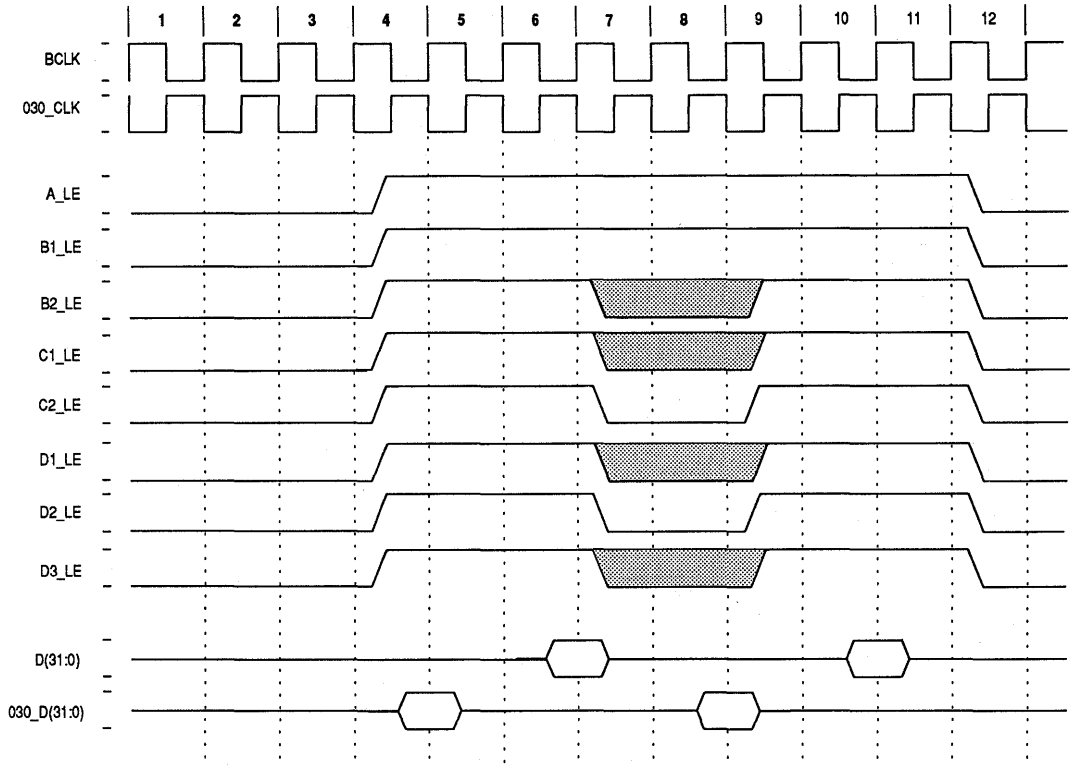


Figure 7-25. Long-Word Read, to Word Port Example (Sheet 2 of 2)

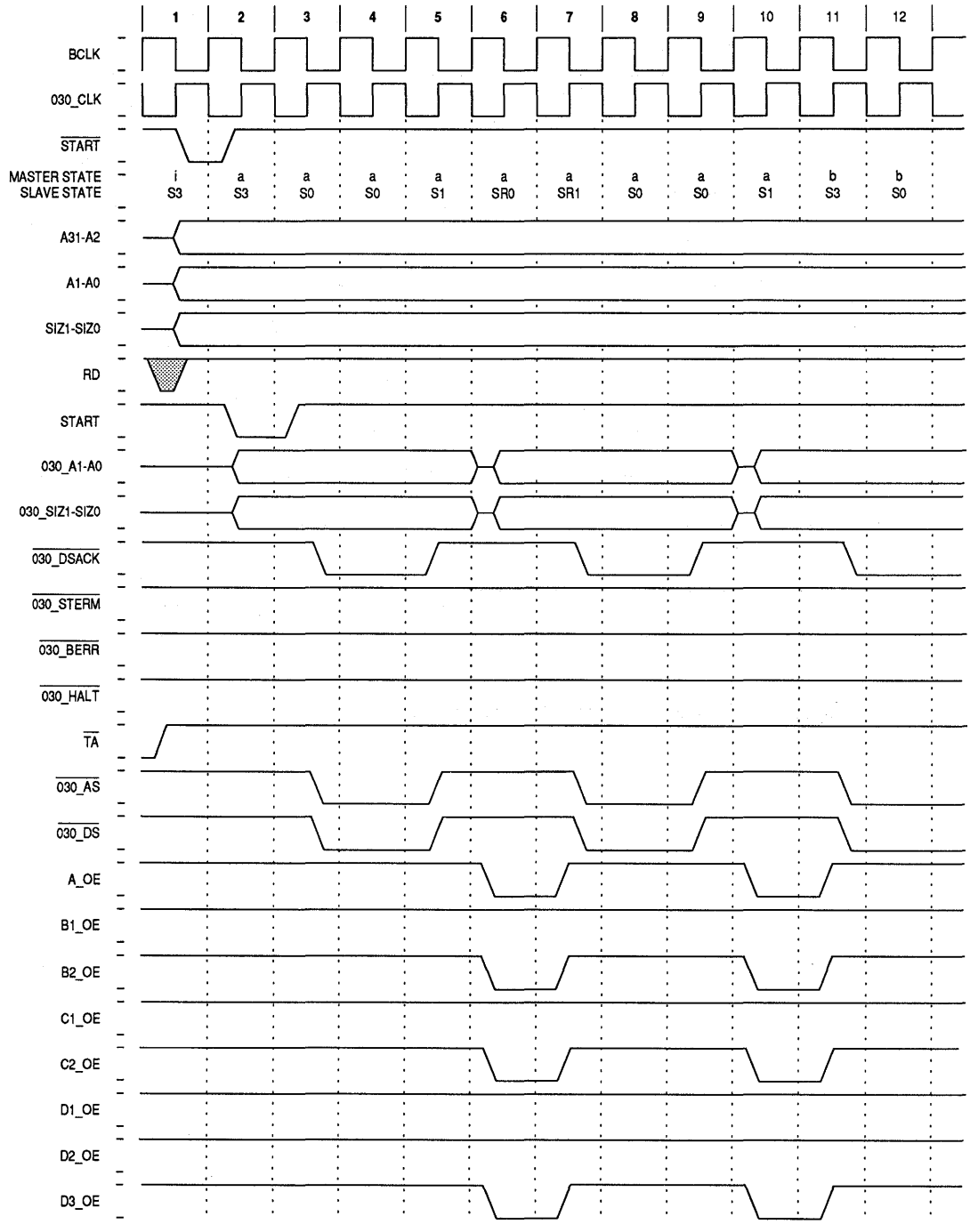


Figure 7-26. Long-Word Read, to Byte Port Example (Sheet 1 of 2)

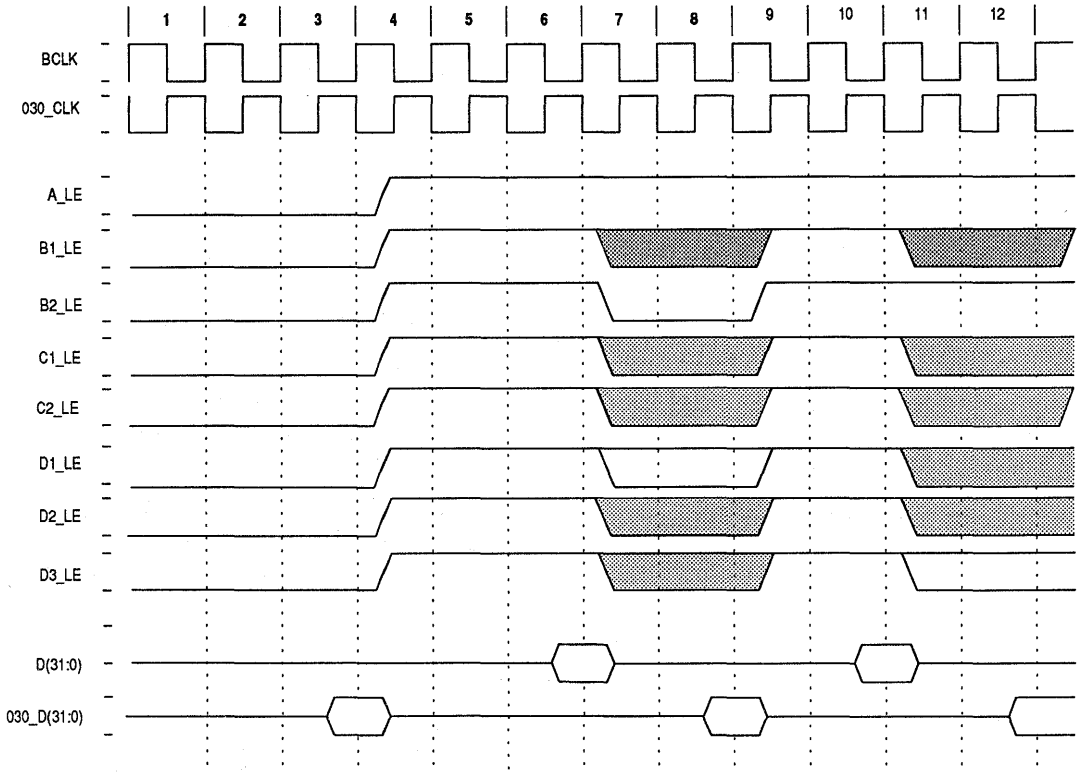


Figure 7-26. Long-Word Read, to Byte Port Example (Sheet 2 of 2)

SECTION 8
IEEE 1149.1 TEST ACCESS PORT

The MC68040 includes dedicated user-accessible test logic that is fully compatible with the IEEE 1149.1 *Standard Test Access Port and Boundary Scan Architecture.* Problems associated with testing high-density circuit boards have led to development of this standard under the sponsorship of the Test Technology Committee of the IEEE and the Joint Test Action Group (JTAG). The MC68040 implementation supports circuit board test strategies based on this standard.

The test logic includes a test access port (TAP) consisting of five dedicated signal pins, a 16-state controller, and two test data registers. A boundary scan register links all device signal pins into a single shift register. The test logic is implemented utilizing static logic design and is independent of the system logic of the device. The MC68040 implementation provides capabilities to

a. perform boundary scan operations to test circuit board electrical continuity,
b. bypass the MC68040 by reducing the shift register path of the device to a single cell,
c. sample the MC68040 system pins during operation and to transparently shift out the result in the boundary scan register, and
d. disable the output drive to output-only pins during circuit board testing.

NOTE

The IEEE 1149.1 test logic cannot be considered completely benign to those planning not to use this capability. Certain precautions must be observed to insure that this logic does not interfere with nontest operation. See paragraph **8.4 NON-IEEE 1149.1 OPERATION** for details.

8.1 OVERVIEW

This document includes those aspects of the IEEE 1149.1 implementation that are specific to the MC68040 and is intended to be used with the supporting IEEE document. The scope of this description includes those items required by the standard to be defined and, in certain cases, provides additional information specific to the MC68040 implementation. For internal details and applications of the standard, the reader is referred to the IEEE 1149.1 document.

An overview of the MC68040 implementation of IEEE 1149.1 is shown in Figure 8-1. The implementation includes a dedicated TAP consisting of the following signals:

- TCK – a test clock input to synchronize the test logic,
- TMS – a test mode select input with an internal pull-up resistor sampled on the rising edge of TCK to sequence the test controller's state machine,
- TDI – a test data input with an internal pull-up resistor sampled on the rising edge of TCK,
- TDO – a three-stateable test data output actively driven only in the Shift-IR and *Shift-DR* controller states which changes on the falling edge of TCK, and
- $\overline{\text{TRST}}$ – an active low asynchronous reset with an internal pull-up resistor which forces the TAP controller into the *Test-Logic-Reset* state.

The MC68040 implementation includes a 3-bit instruction register. Test data registers include a 1-bit bypass register and a 184-bit boundary scan register.

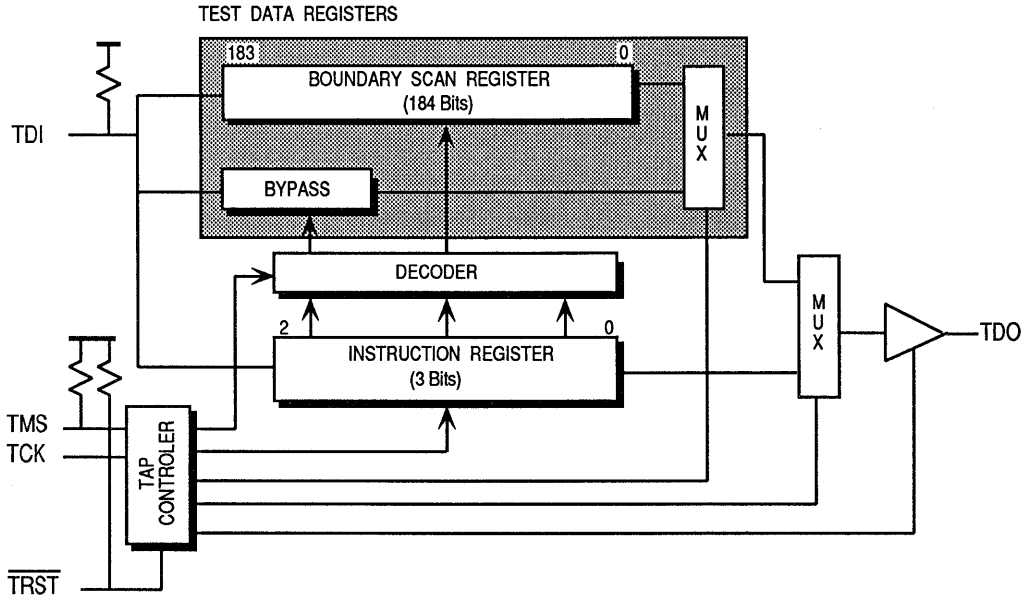


Figure 8-1. MC68040 Test Logic Block Diagram

8.2 INSTRUCTION REGISTER

The MC68040 IEEE 1149.1 implementation includes the three mandatory public instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) but does not support any of the optional standard public instructions. Two additional public instructions provide a capability to disable all device output drivers and to operate the device in a BYPASS configuration without a system clocking requirement. The MC68040 includes a 3-bit instruction register without parity consisting of a shift register with

three parallel outputs. Data is transferred from the shift register to the parallel outputs during the *Update-IR* controller state. Three bits are used to decode the instructions shown in Table 8-1. Note that the least significant bit of the instruction (b0) is the first bit to be shifted into the instruction register.

The instruction parallel output register is reset to all ones in the *Test-Logic-Reset* controller state. Note that this preset state is equivalent to selecting the BYPASS instruction.

Table 8-1. Instructions

Code			Instruction	Test Data Register
b2	b1	b0		
0	0	0	EXTEST	BOUNDARY SCAN
0	0	1	HI-Z	BYPASS
0	1	X	SAMPLE/PRELOAD	BOUNDARY SCAN
1	0	X	SHUTDOWN	BYPASS
1	1	0	RESERVED	—
1	1	1	BYPASS	BYPASS

During the *Capture-IR* controller state, the 3-bit binary value "001" is loaded into the parallel inputs of the instruction shift register.

8.2.1 EXTEST (000)

The external test instruction (EXTEST) selects the 184-bit boundary scan register which includes cells for all device signal pins and clock pins along with associated control signals. This instruction activates two internal functions which are intended to protect the device from damage while performing boundary scan operations. Finally, the instruction asserts an internal reset for the MC68040 system logic to force a predictable benign internal state and the instruction activates an internal low frequency clock to protect the device from internal damage. The internal clock obviates the requirement to keep the processor clock (PCLK) running and frees the clock pins for boundary scan testing.

If the EXTEST instruction is **not** immediately preceded by either the HI-Z or SHUTDOWN instruction, then two clocking restrictions apply. First, the processor clocks must be kept running for at least two (2) BCLK periods after the *Update-IR* state which invokes the EXTEST instruction. This restriction is necessary to allow time for the internal reset to propagate through a BCLK synchronizer. After the two (2) BCLK periods expire, there are no time domain processor clock pin requirements while the EXTEST instruction is active. The processor clocks (PCLK and BCLK) must be re-started at least two (2) BCLK periods prior to invoking another instruction other than SHUTDOWN or HI-Z. Failure to do so will turn off the processor low frequency clock since it is invoked by an EXTEST, SHUTDOWN, or HI-Z instruction decode and, therefore, will re-instate the processor system clocking requirements.

By using the TAP controller, the register is capable of scanning user-defined values into the output buffers, capturing values presented to input pins, and controlling the direction of bi-directional pins and the output drive of three-statable output pins. All MC68040 bi-directional pins include two boundary scan data cells (input and output) and are controlled by one of five associated boundary scan control cells.

Boundary scan bit definitions are shown in Table 8-2. The first column in Table 8-2 defines the bit's position in the boundary scan register. The shift register cell nearest TDO (i.e., first to be shifted out) is defined as bit zero. The last bit to be shifted out is bit 183. The second column in Table 8-2 references one of the three MC68040 cell types depicted in Figures 8-3–8-5, which describe the cell structure for that bit.

The third column in Table 8-2 lists the pin name for all pin-related cells or defines the name of bi-directional control register cells. The five bi-directional and three-state control bits and their ordinal bit positions are as follows:

1. io.db (150)
2. io.ab (151)
3. io.2 (154)
4. io.1 (155)
5. io.0 (156)

If these cells contain a logic one, the associated bi-directional or three-state pin will be configured as an output and enabled.

The fourth column lists the system pin type for convenience where "TS-output" indicates a three-stateable output pin and "Inp/Out" indicates a bi-directional pin. The last column of Table 8-2 lists the name of the associated boundary scan register control bit for bi-directional and three-state output pins.

Table 8-2. Boundary Scan Bit Definitions

BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL	BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
0	O.Latch	RSTO	Output	(Note 1)	50	I.Pin	A30	Inp/Out	io.ab
1	O.Latch	IPEND	Output	(Note 1)	51	O.Latch	A31	Inp/Out	io.ab
2	O.Latch	CIOUT	TS-Output	io.0	52	I.Pin	A31	Inp/Out	io.ab
3	O.Latch	UPA0	TS-Output	io.0	53	O.Latch	D0	Inp/Out	io.db
4	O.Latch	UPA1	TS-Output	io.0	54	O.Latch	D1	Inp/Out	io.db
5	O.Latch	TT0	Inp/Out	io.0	55	O.Latch	D2	Inp/Out	io.db
6	I.Pin	TT0	Inp/Out	io.0	56	O.Latch	D3	Inp/Out	io.db
7	O.Latch	TT1	Inp/Out	io.0	57	O.Latch	D4	Inp/Out	io.db
8	I.Pin	TT1	Inp/Out	io.0	58	O.Latch	D5	Inp/Out	io.db
9	O.Latch	A10	Inp/Out	io.ab	59	O.Latch	D6	Inp/Out	io.db
10	I.Pin	A10	Inp/Out	io.ab	60	O.Latch	D7	Inp/Out	io.db
11	O.Latch	A11	Inp/Out	io.ab	61	O.Latch	D8	Inp/Out	io.db
12	I.Pin	A11	Inp/Out	io.ab	62	O.Latch	D9	Inp/Out	io.db
13	O.Latch	A12	Inp/Out	io.ab	63	O.Latch	D10	Inp/Out	io.db
14	I.Pin	A12	Inp/Out	io.ab	64	O.Latch	D11	Inp/Out	io.db
15	O.Latch	A13	Inp/Out	io.ab	65	O.Latch	D12	Inp/Out	io.db
16	I.Pin	A13	Inp/Out	io.ab	66	O.Latch	D13	Inp/Out	io.db
17	O.Latch	A14	Inp/Out	io.ab	67	O.Latch	D14	Inp/Out	io.db
18	I.Pin	A14	Inp/Out	io.ab	68	O.Latch	D15	Inp/Out	io.db
19	O.Latch	A15	Inp/Out	io.ab	69	O.Latch	D16	Inp/Out	io.db
20	I.Pin	A15	Inp/Out	io.ab	70	O.Latch	D17	Inp/Out	io.db
21	O.Latch	A16	Inp/Out	io.ab	71	O.Latch	D18	Inp/Out	io.db
22	I.Pin	A16	Inp/Out	io.ab	72	O.Latch	D19	Inp/Out	io.db
23	O.Latch	A17	Inp/Out	io.ab	73	O.Latch	D20	Inp/Out	io.db
24	I.Pin	A17	Inp/Out	io.ab	74	O.Latch	D21	Inp/Out	io.db
25	O.Latch	A18	Inp/Out	io.ab	75	O.Latch	D22	Inp/Out	io.db
26	I.Pin	A18	Inp/Out	io.ab	76	O.Latch	D23	Inp/Out	io.db
27	O.Latch	A19	Inp/Out	io.ab	77	O.Latch	D24	Inp/Out	io.db
28	I.Pin	A19	Inp/Out	io.ab	78	O.Latch	D25	Inp/Out	io.db
29	O.Latch	A20	Inp/Out	io.ab	79	O.Latch	D26	Inp/Out	io.db
30	I.Pin	A20	Inp/Out	io.ab	80	O.Latch	D27	Inp/Out	io.db
31	O.Latch	A21	Inp/Out	io.ab	81	O.Latch	D28	Inp/Out	io.db
32	I.Pin	A21	Inp/Out	io.ab	82	O.Latch	D29	Inp/Out	io.db
33	O.Latch	A22	Inp/Out	io.ab	83	O.Latch	D30	Inp/Out	io.db
34	I.Pin	A22	Inp/Out	io.ab	84	O.Latch	D31	Inp/Out	io.db
35	O.Latch	A23	Inp/Out	io.ab	85	I.Pin	D0	Inp/Out	io.db
36	I.Pin	A23	Inp/Out	io.ab	86	I.Pin	D1	Inp/Out	io.db
37	O.Latch	A24	Inp/Out	io.ab	87	I.Pin	D2	Inp/Out	io.db
38	I.Pin	A24	Inp/Out	io.ab	88	I.Pin	D3	Inp/Out	io.db
39	O.Latch	A25	Inp/Out	io.ab	89	I.Pin	D4	Inp/Out	io.db
40	I.Pin	A25	Inp/Out	io.ab	90	I.Pin	D5	Inp/Out	io.db
41	O.Latch	A26	Inp/Out	io.ab	91	I.Pin	D6	Inp/Out	io.db
42	I.Pin	A26	Inp/Out	io.ab	92	I.Pin	D7	Inp/Out	io.db
43	O.Latch	A27	Inp/Out	io.ab	93	I.Pin	D8	Inp/Out	io.db
44	I.Pin	A27	Inp/Out	io.ab	94	I.Pin	D9	Inp/Out	io.db
45	O.Latch	A28	Inp/Out	io.ab	95	I.Pin	D10	Inp/Out	io.db
46	I.Pin	A28	Inp/Out	io.ab	96	I.Pin	D11	Inp/Out	io.db
47	O.Latch	A29	Inp/Out	io.ab	97	I.Pin	D12	Inp/Out	io.db
48	I.Pin	A29	Inp/Out	io.ab	98	I.Pin	D13	Inp/Out	io.db
49	O.Latch	A30	Inp/Out	io.ab	99	I.Pin	D14	Inp/Out	io.db

Table 8-2. Boundary Scan Bit Definitions - Continued

BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL	BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
100	I.Pin	D15	Inp/Out	io.db	142	O.Latch	SIZ0	Inp/Out	io.0
101	I.Pin	D16	Inp/Out	io.db	143	I.Pin	SIZ0	Inp/Out	io.0
102	I.Pin	D17	Inp/Out	io.db	144	O.Latch	R/W	Inp/Out	io.0
103	I.Pin	D18	Inp/Out	io.db	145	I.Pin	R/W	Inp/Out	io.0
104	I.Pin	D19	Inp/Out	io.db	146	O.Latch	LOCKE	TS-Output	io.0
105	I.Pin	D20	Inp/Out	io.db	147	O.Latch	SIZ1	Inp/Out	io.0
106	I.Pin	D21	Inp/Out	io.db	148	I.Pin	SIZ1	Inp/Out	io.0
107	I.Pin	D22	Inp/Out	io.db	149	O.Latch	LOCK	TS-Output	io.0
108	I.Pin	D23	Inp/Out	io.db	150	IO.Ctl	io.ab	—	(Note 2)
109	I.Pin	D24	Inp/Out	io.db	151	IO.Ctl	io.db	—	(Note 2)
110	I.Pin	D25	Inp/Out	io.db	152	O.Latch	MI	Output	(Note 1)
111	I.Pin	D26	Inp/Out	io.db	153	O.Latch	BR	Output	(Note 1)
112	I.Pin	D27	Inp/Out	io.db	154	IO.Ctl	io.2	—	(Note 2)
113	I.Pin	D28	Inp/Out	io.db	155	IO.Ctl	io.1	—	(Note 2)
114	I.Pin	D29	Inp/Out	io.db	156	IO.Ctl	io.0	—	(Note 2)
115	I.Pin	D30	Inp/Out	io.db	157	O.Latch	TS	Inp/Out	io.ab
116	I.Pin	D31	Inp/Out	io.db	158	I.Pin	TS	Inp/Out	io.ab
117	O.Latch	A9	Inp/Out	io.ab	159	O.Latch	BB	Inp/Out	io.1
118	I.Pin	A9	Inp/Out	io.ab	160	I.Pin	BB	Inp/Out	io.1
119	O.Latch	A8	Inp/Out	io.ab	161	O.Latch	TIP	TS-Output	io.1
120	I.Pin	A8	Inp/Out	io.ab	162	O.Latch	PST3	Output	(Note 1)
121	O.Latch	A7	Inp/Out	io.ab	163	O.Latch	PST2	Output	(Note 1)
122	I.Pin	A7	Inp/Out	io.ab	164	O.Latch	PST1	Output	(Note 1)
123	O.Latch	A6	Inp/Out	io.ab	165	O.Latch	PST0	Output	(Note 1)
124	I.Pin	A6	Inp/Out	io.ab	166	O.Latch	TA	Inp/Out	io.2
125	O.Latch	A5	Inp/Out	io.ab	167	I.Pin	TA	Inp/Out	io.2
126	I.Pin	A5	Inp/Out	io.ab	168	I.Pin	TEA	Input	--
127	O.Latch	A4	Inp/Out	io.ab	169	I.Pin	BG	Input	--
128	I.Pin	A4	Inp/Out	io.ab	170	I.Pin	SC1	Input	--
129	O.Latch	A3	Inp/Out	io.ab	171	I.Pin	SC0	Input	--
130	I.Pin	A3	Inp/Out	io.ab	172	I.Pin	TBI	Input	--
131	O.Latch	A2	Inp/Out	io.ab	173	I.Pin	AVEC	Input	--
132	I.Pin	A2	Inp/Out	io.ab	174	I.Pin	TCI	Input	--
133	O.Latch	A1	Inp/Out	io.ab	175	I.Pin	DLE	Input	--
134	I.Pin	A1	Inp/Out	io.ab	176	I.Pin	PCLK	Input	--
135	O.Latch	A0	Inp/Out	io.ab	177	I.Pin	BCLK	Input	--
136	I.Pin	A0	Inp/Out	io.ab	178	I.Pin	IPL0	Input	--
137	O.Latch	TM2	TS-Output	io.0	179	I.Pin	IPL1	Input	--
138	O.Latch	TM1	TS-Output	io.0	180	I.Pin	IPL2	Input	--
139	O.Latch	TM0	TS-Output	io.0	181	I.Pin	RSTI	Input	--
140	O.Latch	TLN1	TS-Output	io.0	182	I.Pin	CDIS	Input	--
141	O.Latch	TLN0	TS-Output	io.0	183	I.Pin	MDIS	Input	--

NOTES:

1. These output-only cells can be turned "off" (high impedance) by using the HI-Z instruction.
2. All of the control signals (IO.Ctl) are cleared in the *Test-Logic-Reset* controller state.

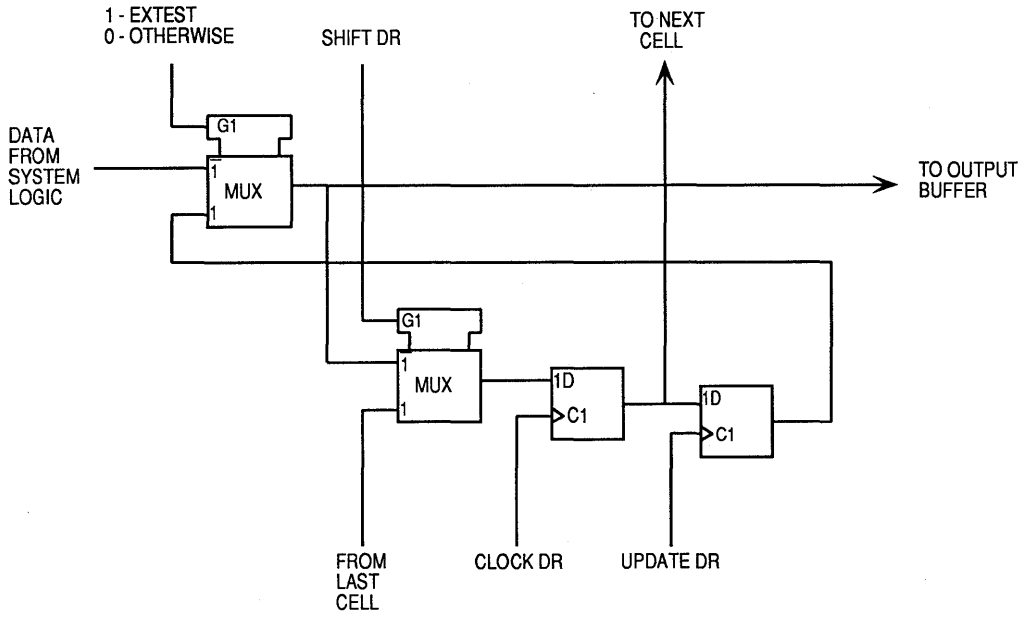


Figure 8-2. Output Latch Cell (O.Latch)

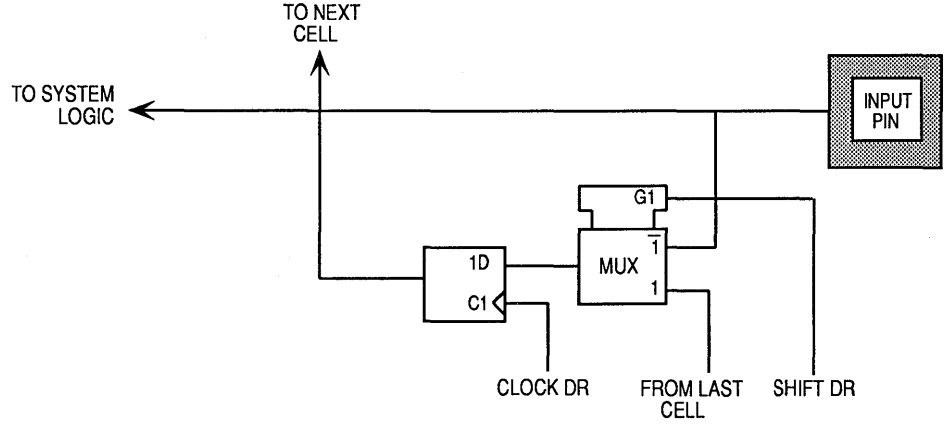
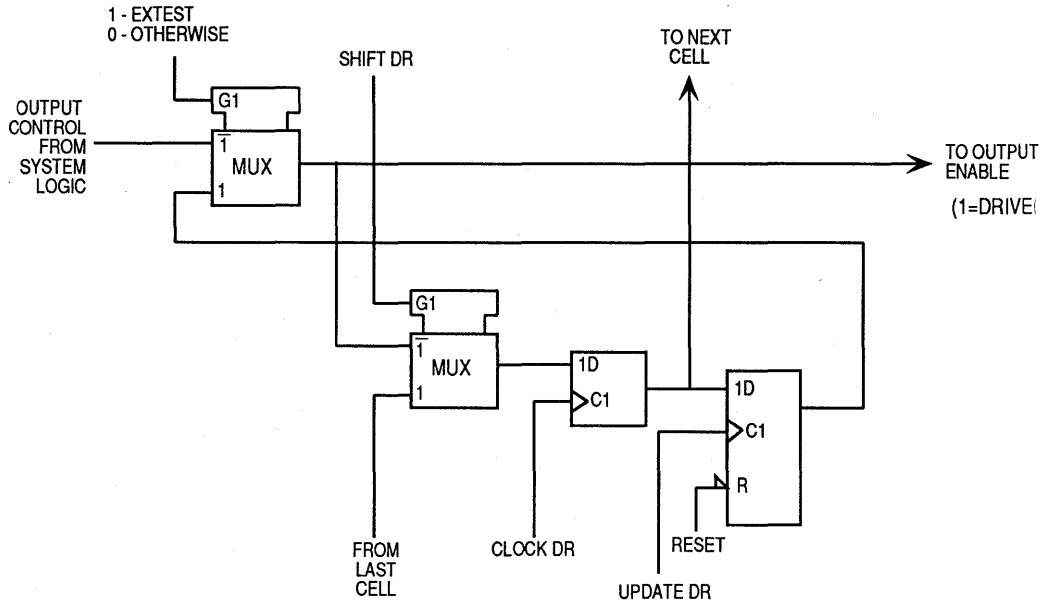
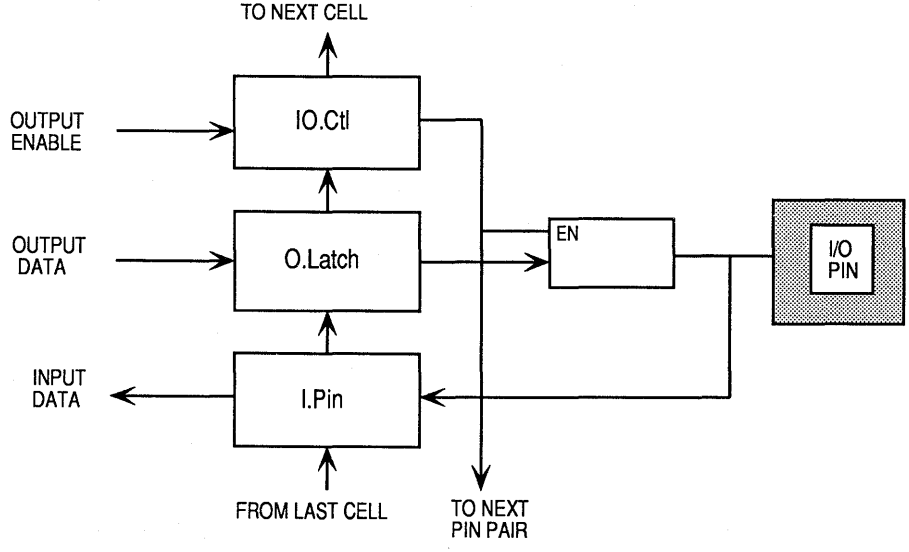


Figure 8-3. Input Pin Cell (I.Pin)



(a) Output Control Cell (IO.Ctl)



(b) General Arrangement of Bi-Directional Pin

Figure 8-4. Output Control Cells (IO.Ctl)

8.2.2 HI-Z (001)

The HI-Z instruction is not included in the IEEE 1149.1 standard. It is provided as a manufacturer's optional public instruction to obviate the need to backdrive output pins during circuit board testing. The instruction implementation is identical to the SHUTDOWN instruction with one exception: the output drivers of output-only pins are disabled (i.e. high impedance). System clocking restrictions for the HI-Z instruction are identical to those of the SHUTDOWN and EXTEST instruction.

The HI-Z instruction can be invoked by using only the TMS and TCK pins and the following TAP controller sequence. The *Test-Logic-Reset* state is entered by asserting $\overline{\text{TRST}}$ or by holding TMS high and clocking TCK for at least five rising edges. A *Capture-IR - Exit1-IR - Update-IR - Run-Test/Idle* TAP controller sequence can then be used to invoke the HI-Z instruction. This scheme works because the value captured by the instruction register during the *Capture-IR* operation is identical to the opcode (001).

The Hi-Z instruction activates the internal low frequency clock, asserts internal system Reset, forces the output-only pins to the high impedance state, and selects the bypass register.

8.2.3 SAMPLE/PRELOAD (01X)

The SAMPLE/PRELOAD instruction provides two separate functions. First, it provides a means to obtain a "snapshot" of system data and control signals. The snapshot occurs on the rising edge of TCK in the *Capture-DR* controller state. The data can be observed by shifting it transparently through the boundary scan register to the output TDO.

NOTE

Since there is no internal synchronization between the IEEE 1149.1 clock (TCK) and the system clock (BCLK), the user must provide some form of external synchronization to achieve meaningful results.

The second function of the SAMPLE/PRELOAD instruction is to initialize the boundary scan register output cells prior to selection of the EXTEST instruction. This insures that known data will appear on the outputs when entering the EXTEST instruction.

8.2.4 SHUTDOWN (10X)

The purpose of this instruction is to provide a means to operate the device in the BYPASS configuration with output-only pins enabled but without a requirement to keep the MC68040 processor clock (PCLK) running. This instruction asserts internal system Reset, activates the internal low-frequency clock, and selects the bypass register. When this instruction has been invoked, the processor clock pins (PCLK and BCLK) are not operative and the user has complete time domain

freedom with these pins while testing other board-level devices coupled to the clock lines.

The system clocking restrictions which apply to the EXTEST and HI-Z instruction also apply to SHUTDOWN. If the instruction preceding SHUTDOWN was not EXTEST or HI-Z, then the processor clocks must be kept running for at least two (2) BCLK periods after the *Update-IR* state which invokes the SHUTDOWN instruction. This restriction is necessary to allow time for the internal reset to propagate through a BCLK synchronizer. After the two (2) BCLK periods expire, there are no time domain processor clock pin requirements while the SHUT-DOWN instruction is active. The processor clocks (PCLK and BCLK) must be re-started at least two (2) BCLK periods prior to invoking another instruction (other than EXTEST or HI-Z). Failure to do so will turn off the processor low frequency clock since it is invoked by an instruction decode of EXTEST, SHUTDOWN, or HI-Z and will re-instate the processor system clocking requirements. Either EXTEST or HI-Z can be entered following the SHUTDOWN instruction without a clocking restriction.

8.2.5 BYPASS (11X)

The BYPASS instruction selects the single-bit bypass register as shown in Figure 8-5. This creates a single-bit shift-register path from TDI to the bypass register to TDO and circumvents the 184-bit boundary scan register. This instruction can be used to enhance test efficiency when a component other than the MC68040 becomes the device under test.

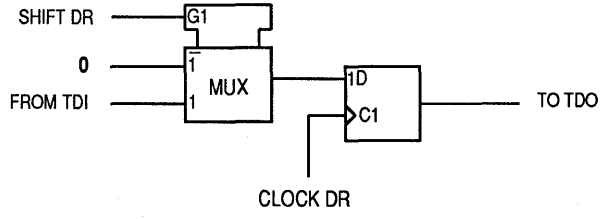


Figure 8-5. Bypass Register

When the bypass register is initially selected, the shift-register stage is set to a logic zero on the rising edge of TCK following entry into the *Capture-DR* controller state. Therefore, the first bit to be shifted out after selecting the bypass register will always be a logic zero.

8.3 MC68040 RESTRICTIONS

The M68040 provides two types of output drivers which are selected by sampling the three IPL pins while $\overline{\text{RESI}}$ is low and then latching the levels on a $\overline{\text{RESI}}$ rising edge. When either the SHUTDOWN or EXTEST instructions are selected, the initial instruction decode effectively latches the value on the IPL pins (if not previously latched by an $\overline{\text{RESI}}$ rising edge) and "freezes" the driver selection

latches while either instruction is being executed. If SHUTDOWN followed by EXTEST is executed, the IPL pins will not be re-sampled between the two instructions. The SHUTDOWN instruction should be used for driver selection if the IPL pins are actively driven by another IEEE 1149.1-compatible device to avoid a race condition when globally selecting the EXTEST instruction. As a consequence of this functionality, the desired driver selection should be previously latched by an RESI rising edge prior to conducting boundary scan operations or levels which select the desired drivers should be present at the IPL pins (with RESI low) at the time either the SHUTDOWN or EXTEST instruction are initially invoked.

The MC68040 IEEE 1149.1 test logic is implemented using static logic design and the test clock (TCK) can be stopped in either a high or low state without loss of data. The system logic, however, includes considerable dynamic logic. For this reason, the system clock (PCLK) cannot be stopped or run slower than the specified frequency except when the EXTEST, HI-Z, and/or SHUTDOWN instructions have been invoked. These three instructions utilize an internal low frequency clock and internal system reset to protect the device from damage. When any of these three instructions have been invoked, there is no system clocking requirement. The clock pins should not be allowed to float.

Control over the output enable signals using the boundary scan register and the EXTEST and HI-Z instructions requires a compatible circuit board test environment to avoid destructive configurations. Avoidance of situations in which the MC68040 output drivers are enabled into actively driven networks is the responsibility of the user.

The $\overline{\text{TRST}}$ signal provides a means to accomplish an asynchronous reset of the IEEE 1149.1 test logic and requires no internal clocking to force the TAP controller into the Test-Logic-Reset state. Negation of the $\overline{\text{TRST}}$ signal, however, requires certain precautions to achieve predictable results. The TMS signal is sampled on the rising edge of TCK and is used to sequence the TAP controller. If TMS is low and $\overline{\text{TRST}}$ is negated simultaneously with a rising edge of TCK, the resultant state is unpredictable but will be either Test-Logic-Reset or *Run-Test/Idle*. To avoid this problem, either the negation of $\overline{\text{TRST}}$ should be synchronized with the falling edge of TCK, or TMS should remain high until after $\overline{\text{TRST}}$ negation. Alternatively, a second TCK period with TMS low can be used following the $\overline{\text{TRST}}$ negation to insure the *Run-Test/Idle* con-troller state.

8.4 NON-IEEE 1149.1 OPERATION

In non-IEEE 1149.1 operation, there are two constraints. First, the test clock input (TCK) does **not** include an internal pull-up resistor and should not be left unconnected to preclude mid-level inputs. The second constraint is to insure that the IEEE 1149.1 test logic is kept transparent to the system logic by providing a means to force the Test-Logic-Reset controller state.

These constraints can be accomplished in several ways. First, $\overline{TRST}$ can be asserted either by connecting it to ground (directly or through a resistor) or by means of a suitable logic network. Connecting $\overline{TRST}$ to the system reset ($\overline{RESI}$) while TCK is tied either high or low meets both requirements. If $\overline{TRST}$ is asserted by a pulse, the TAP controller will be forced to the *Test-Logic-Reset* state and will remain in this state providing a rising edge on the TCK pin does not occur when TMS is low.

As an alternative, $\overline{TRST}$ and TMS can remain unconnected and the TAP controller sequencer utilized to force the Test-Logic-Reset controller state. Since TMS has an internal pull-up resistor, applying a suitable clock to the TCK pin will force the TAP controller into the *Test-Logic-Reset* state within five rising edges of TCK. If this approach is used, the five TCK periods must occur before the MC68040 leaves the system reset state to avoid interference with the first instruction fetch.

8.5 IEEE 1149.1 ELECTRICAL SPECIFICATIONS

The following paragraphs describe the DC electrical characteristics and preliminary timing specifications. The timing specifications are given for reference only.

8.5.1 DC Electrical Characteristics

Table 8-3 list the DC electrical characteristics for the IEEE 1149.1 Test Access Port.

Table 8-3. DC Electrical Characteristics

Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V_{IH}	2	VCC	V
Input Low Voltage	V_{IL}	GND	0.8	V
Undershoot	—	—	0.8	V
TCK Input Leakage Current @ 0.5 2.4 V	I_{in}	20	20	µA
TDO Hi-Z (Off-State) Leakage Current @ 0.5 2.4 V	I_{TST}	20	20	µA
Signal Low Input Current TMS,TDI, $\overline{TRST}$ V_{IL} = 0.8 V	I_L	-1.1	-0.18	mA
Signal High Input Current TMS,TDI, $\overline{TRST}$ V_{IL} = 2.0 V	I_H	-0.94	-0.16	mA
TDO Output High Voltage	V_{OH}	2.4	—	V
TDO Output Low Voltage	V_{OL}	—	0.5	V
Capacitance (see note) V_{in} = 0 V, f= 1 MHz	C_{in}	—	20	pF

NOTE: Capacitance is periodically sampled rather than 100% tested.

8.5.2 IEEE 1149.1 Preliminary Timing Specifications

Refer to Table 8-4 and Figures 8-7–8-9 for the IEEE 1149.1 Test Access Port timing specifications.

Table 8-4. Timing Specifications

Num	Characteristic	25 MHz		Unit
		Min	Max	
	TCK Frequency of Operation	0	10	MHz
1	TCK Cycle Time	100	–	ns
2	TCK Clock Pulse Width Measured at 1.5V	40	–	ns
3	TCK Rise and Fall Times	0	10	ns
4	$\overline{\text{TRST}}$ Setup Time to TCK Falling Edge	TBD	–	ns
5	$\overline{\text{TRST}}$ Assert Time	TBD	–	ns
6	Boundary Scan Input Data Setup Time	TBD	–	ns
7	Boundary Scan Input Data Hold Time	TBD	–	ns
8	TCK to Output Data Valid	TBD	TBD	ns
9	TCK to Output High Impedance	TBD	TBD	ns
10	TMS, TDI Data Setup Time	20	–	ns
11	TMS, TDI Data Hold Time	5	–	ns
12	TCK to TDO Data Valid	0	20	ns
13	TCK to TDO High Impedance	0	20	ns

TBD – To be determined

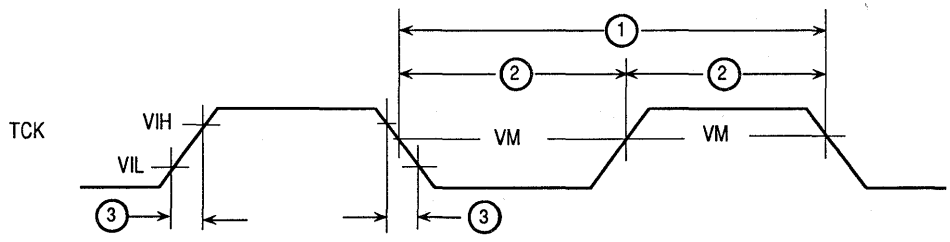


Figure 8-6. Clock Input Timing Diagram

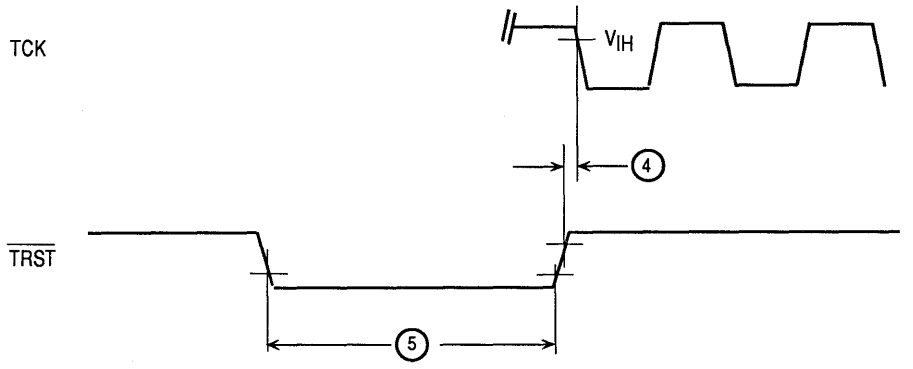


Figure 8-7. $\overline{\text{TRST}}$ Timing Diagram

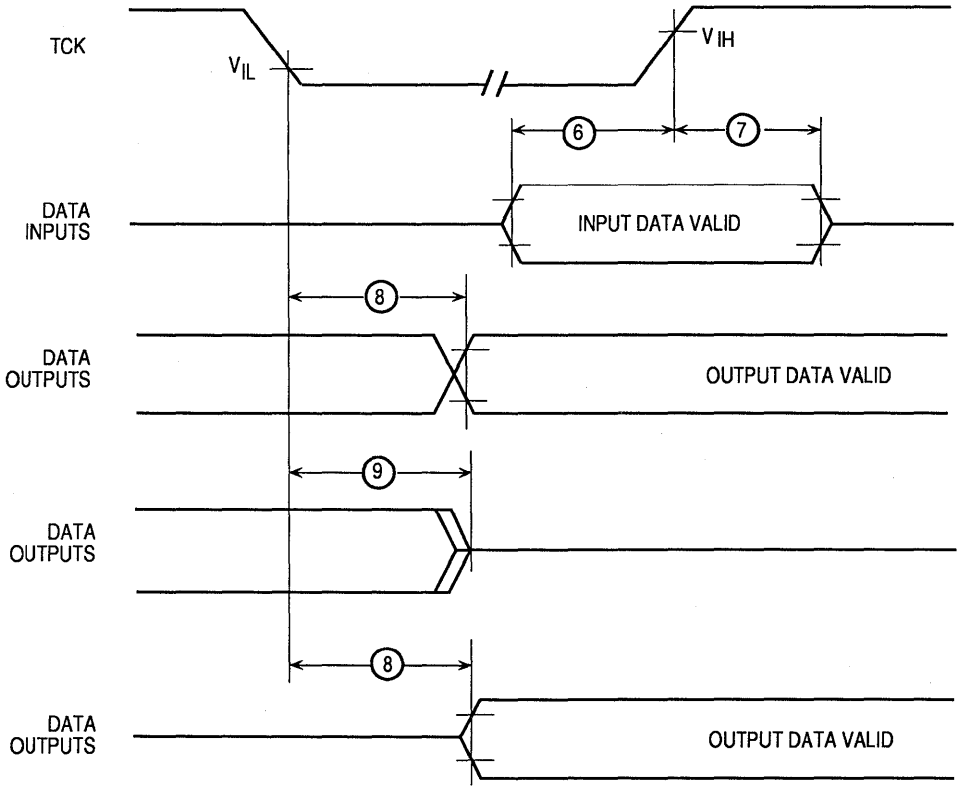


Figure 8-8. Boundary Scan Timing Diagram

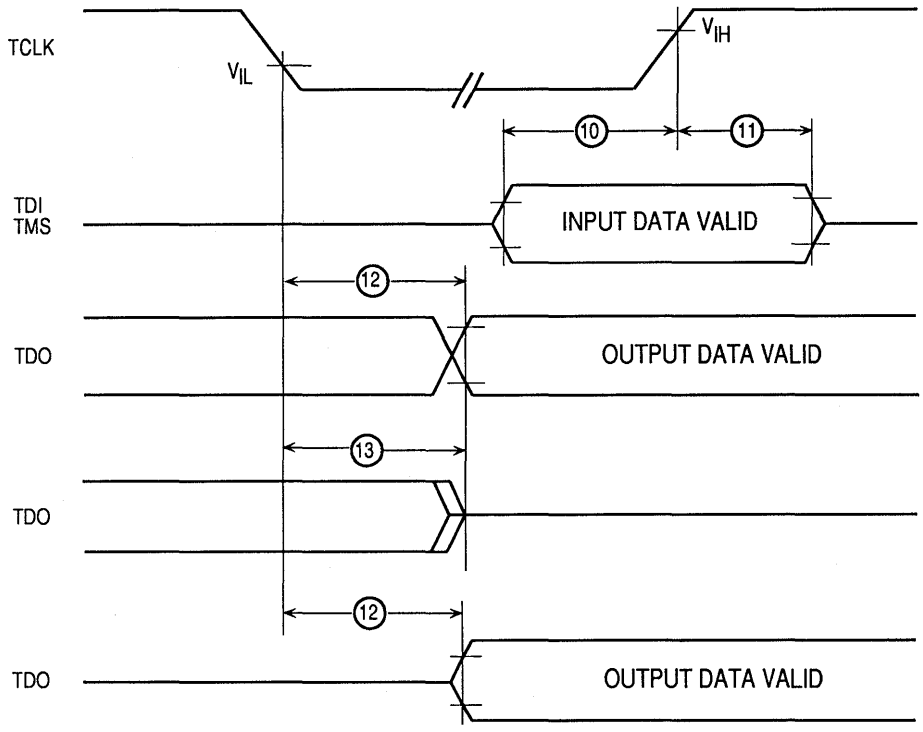


Figure 8-9. Test Access Port Timing Diagram

SECTION 9
BUS INTERFACE CONSIDERATIONS and EXAMPLES

This section provides examples of bus interfaces. Schematics and PAL coding is also provided.

9.1 Byte-Select Logic For The MC68040

The following paragraphs describe the generation of byte-select control signals, the transfer of various sized operands, and the transfer of misaligned operands to operate correctly.

The following signals control the MC68040 operand transfer mechanism:

A1, A0 — Address lines. The most significant byte of the operand to be transferred is addressed directly.

SIZ1, SIZ0 — Transfer size. Output of the MC68040 indicating the number of bytes to be transferred during a given bus cycle.

R/$\overline{W}$ — Read/Write. Output of the MC68040 indicating whether the MC68040 is performing a read or write to the specific memory location. R/$\overline{W}$ must be included in the byte-select enable logic.

$\overline{TS}$ — Transfer start. Output of the MC68040 which indicates the start of a bus cycle.

$\overline{TA}$— Transfer acknowledge. Driven by a synchronous port to acknowledge the receipt or transmission of data.

The MC68040 bus interface assumes that all ports connected to it are 32 bits wide. If there are 8- and 16-bit ports on the bus, the software must consider which data lines the ports reside on.

The need for byte-select logic is best illustrated by an example. Consider a long-word write cycle to an address specifying A1, A0 = 01. The transfer requires three bus cycles to complete. The first bus cycle transfers the most significant byte of the long word on D16–D23. The second bus cycle transfers a word on D0–D15, and the last bus cycle transfers the least significant byte of the original long word on D24–D31. To prevent overwriting those bytes which are not used, a unique byte data strobe must be generated for each byte.

For all write cycles, the required active bytes of the data bus for any given bus transfer are a function of the size (SIZ1/SIZ0) and lower address (A1/A0) outputs

(see Table 9-1). Individual strobes or select signals can be generated by decoding these four signals for every bus cycle.

Table 9-1. Data Bus Activity for Byte, Word, and Long-Word Transfers

Transfer Size	SIZ1	SIZ0	A1	A0	Data Bus Valid Sections			
					D31-D24	D23-D16	D15-D8	D7-D0
Byte	0	1	0	0	V	—	—	—
	0	1	0	1	—	V	—	—
	0	1	1	0	—	—	V	—
	0	1	1	1	—	—	—	V
Word	1	0	0	0	V	V	—	—
	1	0	1	0	—	—	V	V
Long Word	0	0	x	x	V	V	V	V
Line	1	1	x	x	V	V	V	V

Notes: x=Don't Care, V=Valid for this transfer

During read cycles, the memory device is only required to drive valid data on the active bytes of the data bus. Since the data on the inactive bytes is ignored, memory devices can drive all 32 bits of the data bus for all read accesses to simplify chip select or output enable logic. For I/O devices, only those bytes specifically selected by the bus cycle should be accessed, since many I/O devices have status and control features which are access triggered.

To selectively byte enable for only write bus cycles requires the byte-select logic for the MC68040 to consider the R/$\overline{W}$ signal. Figure 9-1 provides sample logic for write byte enable signals to a 32-bit static random access memory (SRAM) bank.

The logic presented is not intended to be the optimal implementation for every system. Depending on the CPU clock frequency, memory access times, and system architecture, different circuits may be required.

9.2 Memory Interface

The MC68040 is capable of running three types of external bus cycles as determined by the cycle termination and handshake signals:

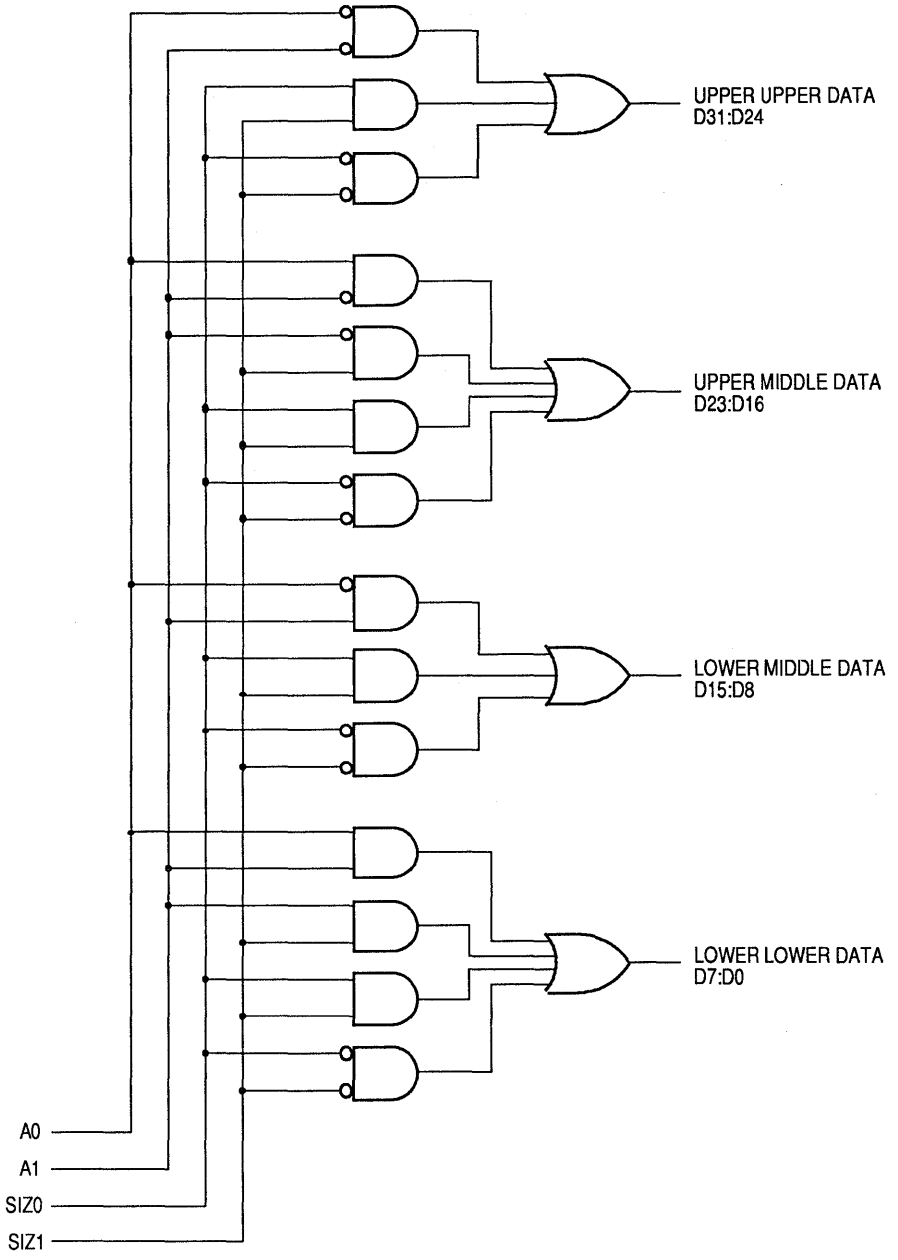


Figure 9-1. MC68040 Write Byte Select Logic

1. Synchronous cycles, terminated by the $\overline{TA}$ signal, have a minimum duration of two processor clock periods in which up to four bytes are transferred.
2. Burst operation cycles, terminated by the $\overline{TA}$ signal responding four times, have a duration of as little as five processor clock periods in which four long words (16 bytes) are transferred.
3. Burst inhibited cycles, terminated by $\overline{TA}$ and $\overline{TBI}$ signals, have a minimum duration of two processor clock periods in which four bytes are transferred.

During all nonbursting read and write operations, the MC68040 latches data on the last rising edge of the bus cycle when it receives some termination signal(s).

One of the benefits of the MC68040 is that it assumes that all cachable memory is capable of bursting unless terminated otherwise. This allows burst accesses to be made in fewer than no wait states to memory that supports bursting. The MC68040 has both internal data and instruction caches, which are updated on a line basis. These on-chip data and instruction caches lessen the effect of initial external wait states on system performance when the memory is capable of bursting. They also allow for no-wait-state back-to back cachable transfers from memory incapable of supporting the burst operation. This feature allows the caches to be utilized to their fullest potential. Also, because of the high degree of parallelism in the MC68040 architecture, the integer execution unit is allowed to proceed after attaining the particular long word(s) of data operand(s) it requires even before the line is cached.

9.2.1 Access Time Calculations

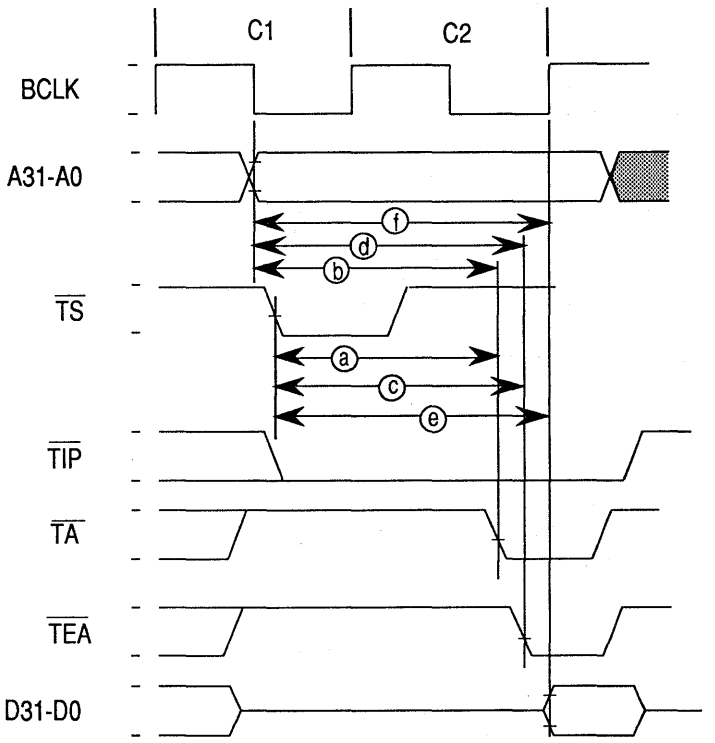
The time paths that are typically critical in any memory interface are illustrated and defined in Figure 9-2. For burst transfers, the first long word transferred also uses these parameters, but the subsequent transfers are different and are discussed in **9.2.3 Burst Mode Cycles**.

The type of device that is interfaced to the MC68040 determines exactly which of the paths is most critical. The address-to-data paths are typically the critical paths for static devices since there is no penalty for initiating a cycle to these devices and later validating that access with the appropriate bus control signal. Conversely, the transfer start and address-to-data-valid path is often most critical for dynamic devices since the cycle must be validated before an access can be initiated. Finally, the address and transfer-start-valid-to-transfer-acknowledge-asserted path is most critical for very fast devices and external caches since the time available between when the address is valid and when $\overline{TA}$ must be asserted to terminate the bus cycle is minimal. Table 9-2 provides the equations required to calculate the various memory access times assuming a 50-percent duty cycle clock.

During bus cycle to asynchronous peripherals, $\overline{TA}$ is used to terminate the current bus cycle. When accessing peripherals operating at a different clock frequency, the $\overline{TA}$ signal must be asserted while synchronized to the processor clock with the data held valid before and after the rising clock edge as defined by specifications #15 and

#16. With a 25-MHz processor, this setup time is 5 ns before the clock edge, and the hold time is 4 ns after the edge.

All cycles are forced to be synchronous to the processor. All cachable and noncachable accesses are terminated alike with the assertion of $\overline{TA}$, or a combination of $\overline{TA}$, $\overline{TEA}$, $\overline{TBI}$, and $\overline{TCI}$. Since all signals are synchronous with similar drive buffers outputting them from the MC68040, the access times available are similar.



Parameter	Description	System	Equation
a	$\overline{TS}$ Valid to $\overline{TA}$ Valid	t_{TSTA}	(9-1)
b	Address Valid to $\overline{TA}$ Valid	t_{AVTA}	(9-2)
c	$\overline{TS}$ Valid to $\overline{TEA}$ Valid	t_{TSTEA}	(9-3)
d	Address Valid to $\overline{TEA}$ Valid	t_{AVTEA}	(9-4)
e	$\overline{TS}$ Valid to Data Valid	t_{TSDV}	(9-5)
f	Address Valid to Data Valid	t_{AVDV}	(9-6)

Figure 9-2. Access Time Computation Diagram

Table 9-2. Memory Access Time Equations at 25 MHz

Equations	N=2	N=3	N=4	N=5	N=6
(9-1) t_{TSTA} = (N-1)*T-t_{13}-t_{22a}	49 ns	89 ns	129 ns	169 ns	209 ns
(9-2) t_{AVTA} = (N-1)*T-t_{11}-t_{22a}	49 ns	89 ns	129 ns	169 ns	209 ns
(9-3) t_{TSTEA} = (N-1)*T-t_{13}-t_{22b}	49 ns	89 ns	129 ns	169 ns	209 ns
(9-4) t_{AVTEA} = (N-1)*T-t_{11}-t_{22b}	49 ns	89 ns	129 ns	169 ns	209 ns
(9-5) t_{TSDV} = (N-1)*T-t_{13}-t_{15}	54 ns	94 ns	134 ns	174 ns	214 ns
(9-6) t_{AVDV} = (N-1)*T-t_{11}-t_{15}	54 ns	94 ns	134 ns	174 ns	214 ns

where:

t_X = Refers to AC Electrical Specification #x
T = Clock Period (@ 25 MHz = 40 ns)
t_{11} = Clock High To Address Valid (@ 25MHz = 21 ns Max)
t_{13} = Clock High To $\overline{TS}$ Valid (@ 25 MHz = 21 ns Max)
t_{15} = Data-In Valid To Clock High Setup (@ 25 MHz = 5 ns Min)
t_{22a} = $\overline{TA}$ Valid To Clock High Setup (@ 25 MHz = 10 ns Min)
t_{22b} = $\overline{TEA}$ Valid To Clock High Setup (@ 25 MHz = 10 ns Min)

It can be seen that there are only two initial access times which need to be considered: transfer start and/or address valid to termination and transfer start and/or address valid to data valid. Although the termination signal has the minimal time in an initial access, it can usually be generated more quickly than data since data usually suffers the delay of buffering. Therefore, it is generally helpful to have more time from address valid to data valid.

One way to optimize the CPU-to-memory access times in a system is to use a clock frequency less than the rated maximum of the specific MC68040 device. When doing this, the effect that the variation of the PCLK and BCLK has on the specifications for that device must be considered. Since the MC68040 is a synchronous processor which drives its outputs at PCLK edges, the timings referencing the BCLK edges are not valid at different clock frequencies. The following paragraphs discuss the relationship between BCLK , PCLK, and the specifications. Also, a background of how to calculate the adjusted specification for varying operating frequencies will be shown.

9.2.2 Calculating Frequency-Adjusted MC68040 Output Specifications

The MC68040 uses two clocks to generate timing — a bus clock (BCLK) and a processor clock (PCLK). The PCLK signal is exactly twice the frequency of the BCLK signal and is internally phase-locked to BCLK and distributed throughout the device to generate timing for all logic blocks. The BCLK signal is only used as the reference signal for the phase-lock loop (PLL), which synchronizes the PCLK. The use of dual clock inputs allows the bus interface to operate at half the speed of the internal logic. Since the rising edge of BCLK is used as the reference point for the PLL, all timing specs are referenced to this edge.

The general relationship between the clock signals and most input and output signals is shown in Figure 9-3. The rising edge of the internal phase-locked PCLK signal is aligned with the rising edge of BCLK, and the two PCLK cycles corresponding to each BCLK cycle are divided into four states, T1-T4. Most outputs change during state T4, whether transitioning between a driven and high-impedance state or switching between high and low logic levels. (The exceptions to this rule are the $\overline{TIP}$, $\overline{TA}$, and $\overline{BB}$ signals, which transition between logic levels during T4 but transition from a driven state to a high-impedance state during T1.)

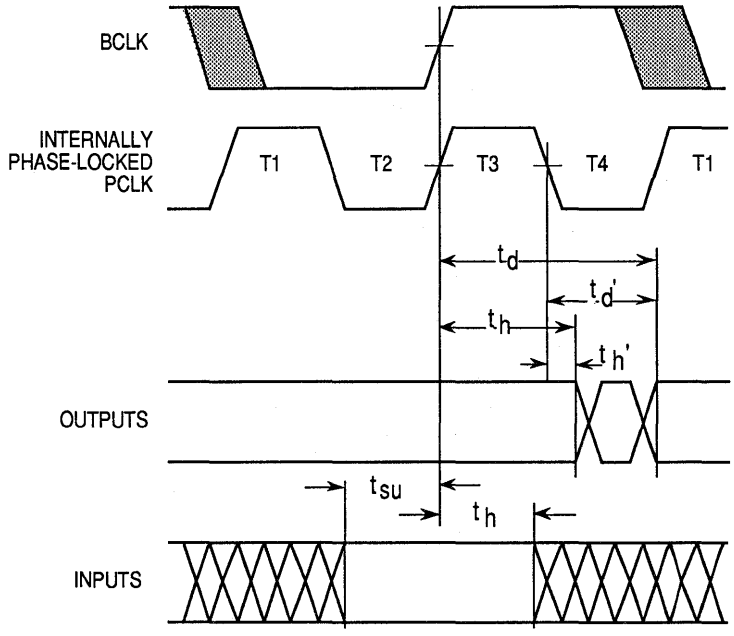


Figure 9-3. Signal Relationships to Clocks

Inputs to the MC68040 (other than the $\overline{IPL2}$-$\overline{IPL0}$ and $\overline{RSTI}$ signals) which are synchronously sampled, must be stable during the sample window defined by the input setup and hold times shown in Figure 9-3 to guarantee proper operation. The asynchronous $\overline{IPLn}$ and $\overline{RSTI}$ signals are also sampled on the rising edge of BCLK, but are internally synchronized to resolve the input to a high or low level before using it.

Since the timing specifications for the MC68040 are referenced to the rising edge of BCLK, they are valid only for the specified operating frequency and must be scaled for lower operating frequencies. The PCLK frequency, the buffer delay, and the jitter skew must be known to calculate the scaled specification for some specifications at various frequencies as shown in Table 9-3.

By referring to Fig. 9-3, each specification can be calculated for a lower operating frequency using the following formulas:

$$t_h = T3 + t_h' - Jitter \quad (9-7)$$

$$t_d = T3 + t_d' + Jitter \quad (9-8)$$

where :

t_h = BCLK rising edge to output held valid
t_h' = Output buffer hold delay (0 ns for 25-MHz Device)
T3 = Pulse width of internal state T3 (10 ns for operating frequency of 25-MHz)
Jitter = Maximum skew between BCLK and the synchronized PCLK (1 ns for 25-MHz Device)
t_d = BCLK rising edge to output driven valid
t_d' = Output buffer delay (10 ns for 25-MHz Device)

Operating a 25-MHz device at 16 MHz will delay outputs as much as 5 ns after the rising edge of BCLK as shown in Table 9-3. Also, operating a 25-MHz device at 20-MHz will delay outputs as much as 2.5 ns after the rising edge of BCLK. Therefore, if designing a system that is running at a lower operating frequency than the device has specified as a maximum operating frequency, all output specifications should be calculated as shown.

Table 9-3. Calculated Output Specifications for Selected Frequencies

Function	Specification Number	MC68040RC25 Operating Frequency					
		16.67 MHz		20.0 MHz		25.0 MHz	
		min	max	min	max	min	max
BCLK High To Address Valid	t_{11}	14	26	11.5	23.5	9	21
BCLK High To $\overline{TS}$ Valid	t_{13}	14	26	11.5	23.5	9	21

Table 9-4 provides calculated t_{avdv} (see Equation 9-6) results for an MC68040RC25 operating at various clock frequencies. If the system uses other clock frequencies, the Equations 9-7 and 9-8 can be used to calculate the exact access times.

Table 9-4.Calculated t_{AVDV} Values for Selected Frequencies

Equation 9-5 t_{AVDV}		MC68040RC25 Operating Frequency		
Clocks Per Bus Cycle (N)	Wait States	16.67 MHz	20 MHz	25 MHz
2 Clock Initial Access	0	89	71.5	54
3 Clock Initial Access	1	149	121.5	94
4 Clock Initial Access	2	209	171.5	134
5 Clock Initial Access	3	269	221.5	174
6 Clock Initial Access	4	329	271.5	214

9.2.3 Burst Mode Cycles

The memory access times for burst mode cycles follow Equations 9-7 and 9-8 for the first access only. For the subsequent accesses, the memory access time calculations depend on the architecture of the burst mode memory system. If the memory system is not capable of bursting, then it may respond with the assertion of $\overline{TBI}$ and $\overline{TA}$; the processor will then follow the same access times as above for all four long-word transfers.

Architectural tradeoffs include the width of the burst memory and the type of memory used. If the memory is 128 bits wide, the subsequent operand accesses do not affect the critical timing paths. For example, if a 3-1-1-1 burst accesses 128-bit wide memory, the first access is governed by the equations in Table 9-2 for N equal to three. The subsequent accesses also use these values as a base, but have additional clock periods which must be used to selectively enable and disable RAM buffers to eliminate bus contention. The second access has one additional clock period, the third access has two additional clock periods, and the fourth access has three additional clock periods. Thus the first cycle determines the critical timing path for the memory access time. However, the RAM buffer enable/disable timings can become critical as the system frequency increases.

A 64-bit-wide memory design presents a compromise between 128-bit-wide and 32-bit-wide configurations in both access times and parts count.

9.3 A 2-1-1-1 Burst Mode Memory Bank Using SRAMs

When the MC68040 is operating at a high clock frequency, a no-wait-state external memory system will most likely be composed of static RAMs. The following paragraphs discuss a static memory subsystem which may be used as shown or as a starting point for an external cache design.

The MC68040 normally attains the lowest bus utilization when the external memory system can support a 2-1-1-1 burst protocol; however, exceptions can occur. For instance, when a large amount of memory accesses are not governed by the locality of reference principles, burst accesses may not decrease bus utilization. This section describes a complete 2-1-1-1,1-Mbyte memory subsystem that can operate with a 25-MHz MC68040 operating in high output buffer mode on all outputs. Nonburst read and write cycles execute in two clocks.

Figure 9-4 shows a block diagram of the SRAM configuration. The required parts include:

(32) 64K x 4 SRAMs 25-ns access time (Motorola MCM6208-25 or equivalent)
(8) 74F545 Bidirectional Transceivers
(7) 74F244 Buffers
(1) 74F04 Inverter
(2) PAL20R8D (or equivalent)
(2) PAL16L8D (or equivalent)
(1) PAL16R4C(or equivalent)

Refer to **9.4 SCHEMATICS** for detailed information on the SRAM configuration.

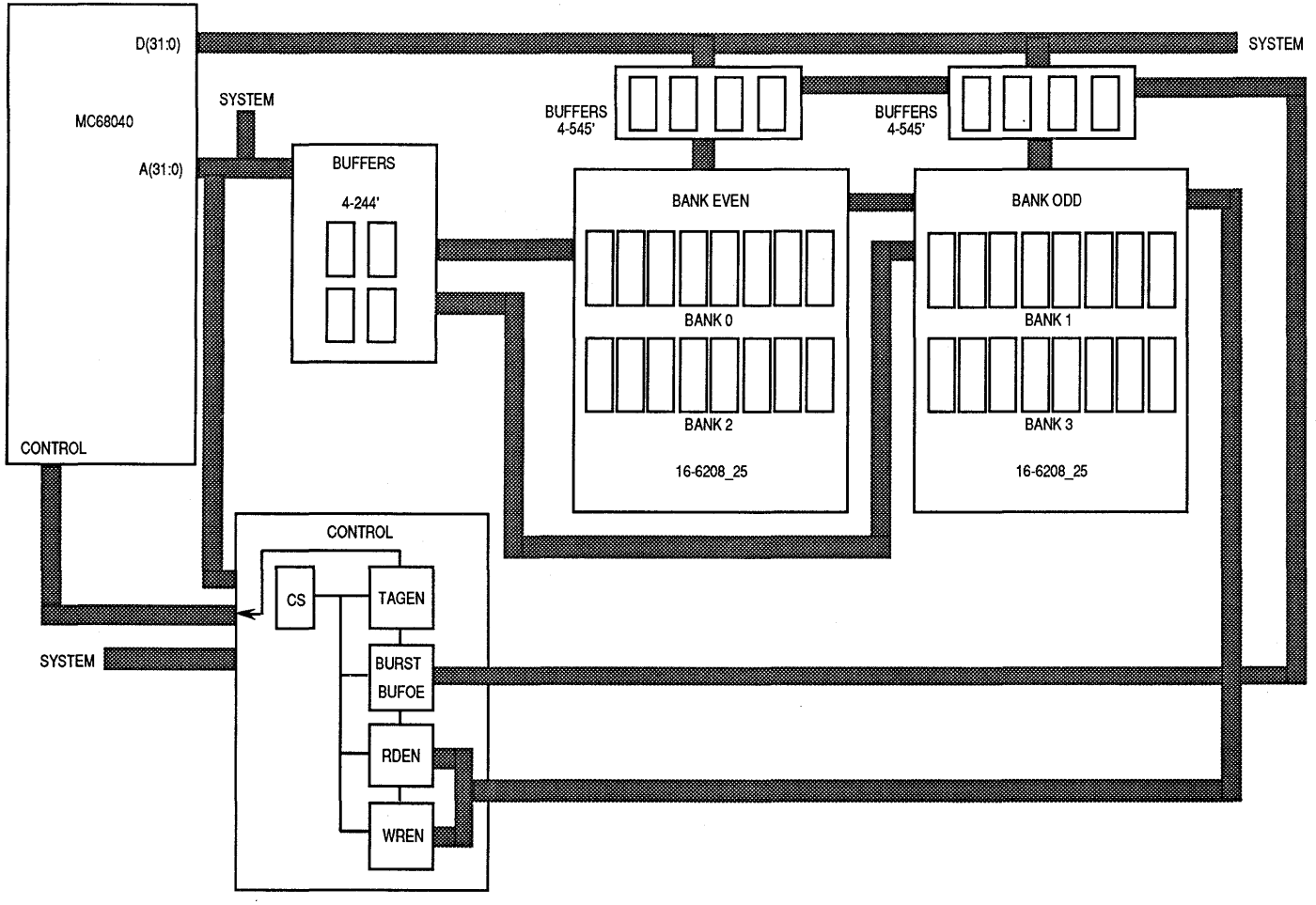


Figure 9-4. SRAM Configuration

The system must also provide the consolidation circuitry required to return all $\overline{TA}$ signals from each port or memory bank in the system to the processor as a single signal. This is done in Figure 9-4 by using a PAL equation in the ramlwe_pal (see Figure 9-5) to combine all other transfer acknowledge signals in the system with the memory transfer acknowledge signal.

```
module ramrdenable_pal;                          flag '-r3','-t1','-x0'
title 'Eval_040 Burst Decode and Longword Read Enable Logic'

                Ramlwe device 'p16l8';  " 7.5 ns

" Inputs
A3              pin 1;
A2              pin 2;
~RAMCS          pin 3;
L1              pin 4;
L2              pin 5;
SIZ1            pin 6;
SIZ0            pin 7;
~OTHERTA        pin 8;
R_~W            pin 9;
RAMCSH          pin 11;


" Outputs
~WE             pin 12;
~TA             pin 19;


" Outputs (I/O)
~ELW0           pin 13;
~ELW1           pin 14;
~ELW2           pin 15;
~ELW3           pin 16;
BURST           pin 17;
CLK25           pin 18;  "Input


" State definition
S0 = (!L1 & !L2);
S1 = ( L1 & !L2);
S2 = ( L1 &  L2);
S3 = (!L1 &  L2);

equations


~ELW0 =         !( !A3 & !A2 & S0 & !~RAMCS & R_~W          #
                !~ELW0 & S0 & RAMCSH & R_~W                 #
                A3 &  A2 & BURST & !S2 & !S3 & R_~W         #
                A3 & !A2 & S1 & ~ELW2 & !CLK25 & R_~W       #
                A3 & !A2 & !~ELW0 & RAMCSH & R_~W           #
                !A3 &  A2 & S2 & ~ELW2 & !CLK25 & R_~W      #
                !A3 &  A2 & !~ELW0 & RAMCSH & R_~W  );
```

Figure 9-5. ramlwe PAL Equations (Sheet 1 of 2)

```
~ELW1 =     !( !A3 &  A2 & S0 & !~RAMCS & R_~W          #
            !~ELW1 & S0 & RAMCSH & R_~W                 #
            !A3 & !A2 & BURST & !S2 & !S3 & R_~W        #
            A3 &  A2 & S1 & ~ELW3 & !CLK25 & R_~W       #
            A3 &  A2 & !~ELW1 & RAMCSH & R_~W           #
            A3 & !A2 & S2 & ~ELW3 & !CLK25 & R_~W       #
            A3 & !A2 & !~ELW1 & RAMCSH & R_~W  );

~ELW2 =     !(  A3 & !A2 & S0 & !~RAMCS & R_~W          #
            !~ELW2 & S0 & RAMCSH & R_~W                 #
            !A3 &  A2 & BURST & !S2 & !S3 & R_~W        #
            !A3 & !A2 & S1 & ~ELW0 & !CLK25 & R_~W      #
            !A3 & !A2 & !~ELW2 & RAMCSH & R_~W          #
            A3 &  A2 & S2 & ~ELW0 & !CLK25 & R_~W       #
            A3 &  A2 & !~ELW2 & RAMCSH & R_~W  );

~ELW3 =     !(  A3 &  A2 & S0 & !~RAMCS & R_~W          #
            !~ELW3 & S0 & RAMCSH & R_~W                 #
            A3 & !A2 & BURST & !S2 & !S3 & R_~W         #
            !A3 &  A2 & S1 & ~ELW1 & !CLK25 & R_~W      #
            !A3 &  A2 & !~ELW3 & RAMCSH & R_~W          #
            !A3 & !A2 & S2 & ~ELW1 & !CLK25 & R_~W      #
            !A3 & !A2 & !~ELW3 & RAMCSH & R_~W  );

BURST =     ( SIZ0 & SIZ1 &  RAMCSH & S0 & !CLK25       #
            BURST & !S3 & RAMCSH  );

~TA =       !( RAMCSH                   #
            !~OTHERTA  );

~WE =       !( !R_~W & !~RAMCS          #
            !R_~W & RAMCSH  );

end ramrdenable_pal;
```

Figure 9-5. ramlwe PAL Equations (Sheet 2 of 2)

The memory subsystem can be divided into five sections:

1. The address decode section (provided by ramcs PAL).
2. The burst state machine and buffer/SRAM control section (provided by bufferoe and ramlwe PALs).
3. The write byte enable section (provided by rambe1 and rambe2 PALs).
4. The actual memory section (SRAMs).
5. The buffer section (address and data).

The first section is completely contained within the ramcs PAL. As shown in Figure 9-6, the PAL equations generate two outputs, which indicate whether the MC68040 transfer is selecting an access to either the SRAM or elsewhere in the system dependant upon address decoding. Depending on the system memory map (i.e., less address pins require decoding), gate logic could be used to determine the $\overline{\text{RAMCS}}$ signal. This would provide the same fast address decode available with a 7.5-ns PAL, which would still allow the SRAMs the longest access time possible for the initial access of any SRAM transfer. The important thing to remember when using this signal is that it could glitch prior to one decoder logic propagation delay after when both $\overline{\text{TS}}$ and the address are guaranteed to be valid. In this design, no crucial action is propagated from this signal until it is qualified with the next rising edge of PCLK25. $\overline{\text{RAMCS}}$ meets a setup time to a registered output clocked by the rising edge of PCLK25.

```
module ramcs_pal;    flag '-r3','-t1','-x1'
title 'Eval_040 Ramcs Decode'

            Ramcs device 'p16l8'; " 7.5 ns

" Inputs
A31,A30,A29,A28,A27,A26,A25,A24,A23,A22      pin 1,2,3,4,5,6,7,8,9,11;


" Inputs  (I/O pins)
A21          pin 13;
A20          pin 14;
~TS          pin 15;
~MAP         pin 18;


" Outputs
CSOTHER   pin 12;
~RAMCS    pin 19;


equations

~RAMCS =  !(!A31 & !A30 & !A29 &  !A28 & !A27 & !A26 & !A25 &
          !A24 & !A23 & !A22 & !A21 & !A20 & !~TS  & !~MAP  );

CSOTHER = (!(!A31 & !A30 & !A29 &  !A28 & !A27 & !A26 & !A25 &
          !A24 & !A23 & !A22 & !A21 & !A20) & !~TS & !~MAP #
          ~MAP & !~TS  );


end ramcs_pal;
```

Figure 9-6. ramcs PAL Equations

The second section is basically contained within the bufferoe and ramlwe PALs shown in Figures 9-7 and 9-5 respectively. The bufferoe PAL generates six signals: two create a burst state machine, two control the data buffers, one controls the write byte enable, and one is a RAM chip-select signal that is held throughout the transfer. The ramlwe PAL generates seven signals: one indicates a burst cycle, one provides a write enable for the RAM and data buffers, four control RAM read enables, and one responds a transfer acknowledge to the MC68040.

```
module bufferoe_pal;    flag '-r3','-t1','-x1'
title 'Eval_040 SRAM Buffer Output Enable Logic'

            bufferoe  device 'p16r4';  " 7.5 ns

" Inputs
clk25       pin 1;
BURST       pin 2;
~RAMCS      pin 3;
CLK25       pin 4;
A2          pin 5;
~RST        pin 6;
~TA         pin 7;
nc1         pin 8;
BRW         pin 9;
~OE         pin 11;

" Registered Outputs
L1          pin 14;
L2          pin 15;
nc1         pin 16;
RAMCSH      pin 17;

" Output  (I/O)
nc1         pin 12;
~DOE0       pin 13;   "output
~DOE1       pin 18;   "output
~WOE        pin 19;   "output

" State definition
S0 = (!L1 & !L2);
S1 = ( L1 & !L2);
S2 = ( L1 &  L2);
S3 = (!L1 &  L2);

equations

~DOE0 =     !( !A2 & S0 & !~RAMCS & ~RST          #
            !~DOE0 & S0 & ~RST & RAMCSH           #
            A2 & S1 & ~DOE1 & ~RST & RAMCSH       #
            !A2 & S2 & ~DOE1 & ~RST & RAMCSH      #
            A2 & S3 & ~DOE1 & ~RST & RAMCSH  );

~DOE1 =     !(  A2 & S0 & !~RAMCS & ~RST          #
            !~DOE1 & S0 & ~RST & RAMCSH           #
            !A2 & S1 & ~DOE0 & ~RST & RAMCSH      #
            A2 & S2 & ~DOE0 & ~RST & RAMCSH       #
            !A2 & S3 & ~DOE0 & ~RST & RAMCSH  );

L1:=         (S0 & !~TA & BURST & ~RST            #
            S1 & !~TA & BURST & ~RST              #
            L1 & !(S2 & !~TA) & ~RST  );

L2:=        (L1 & !~TA & ~RST                     #
            L2 & !(S3 & !~TA) & ~RST  );
```

Figure 9-7. bufferoe PAL Equations (Sheet 1 of 2)

```
~WOE =          !(!BRW & RAMCSH);

RAMCSH:=        (!~RAMCS                #
                 RAMCSH & BURST  );

end bufferoe_pal;
```

Figure 9-7. bufferoe PAL Equations (Sheet 2 of 2)

The burst state generator never leaves state zero unless the MC68040 is requesting a burst access as indicated by the status of the BURST signal from the ramlwe PAL. If the BURST signal is asserted because the SIZ0/SIZ1 signals decode to a line access, then the burst state machine proceeds to count through states 1, 2, and 3, and back to state 0 when it receives an asserted $\overline{TA}$ on each clock edge. These states control the data buffer output enable logic, the read long-word enable logic, and the write byte enable logic. Thus, these states control the complete response of the four banks of RAMS and their two sets of 32 bit data buffers to the MC68040.

The memory and data buffers are configured in such a way that there are two sets of data buffers which, in combination with the byte enable logic manipulate the access of one of four banks of memory at one time to the MC68040 (see Figure 9-4). Since the state machine never leaves state 0 during a nonburst access, it only accesses the specific byte, word, or long-word that was requested by enabling the correct data buffer and RAM bank and then disabling both the RAM and its buffer when the access is complete in preparation for the next access. The determination of which 32-bit buffer is enabled is based on whether the long-word address for the specific byte, word , or long word being accessed is even or odd. If it is even, then signal $\overline{DOE0}$ will be asserted, and the even buffers will be enabled. If it is odd, then signal $\overline{DOE1}$ will be asserted and the odd buffers will be enabled. The $\overline{DOE0}$ and $\overline{DOE1}$ signals are generated in the bufferoe PAL by decoding whether A2 is even or odd in combination with a $\overline{RAMCS}$ assertion for the initial access, and the combination of burst state condition and the status of A2 for burst accesses. The even buffers correspond to RAM bank 0 and RAM bank 2; whereas, the odd buffers correspond to RAM bank 1 and RAM bank 3.

The determination of which 32-bit RAM bank is enabled is based on whether the long-word address for the specific byte, word, or long word being accessed is 0, 1, 2, or 3. The longword address correlates directly to which RAM bank is chosen. For example, if the long-word address for the specific byte, word, or longword being accessed is 3, then the odd buffer ($\overline{DOE1}$) will be enabled along with RAM bank 3. Since the MC68040 ignores any of the driven bytes on the data bus that it did not request for any noncachable reads that might vary in size and requests the full long word for all cachable reads, the read enable logic for the RAM drives the data bus with a full long word of data for every read. However, because the write enable ($\overline{WR}$) on the RAMs is being asserted throughout a write cycle, and the enable signal ($\overline{CS}$) assertion and negation is controlling writing to the RAMs, there must be 16 different byte enable signals controlling byte writes to the 128-bit-wide memory.

Therefore, there are two sets of logic for enabling the RAMs: one for read and another for writes. These two sets of logic are merged to form a single signal to enable each byte of the 128-bit memory. Merging is accomplished by buffering the two sets of logic together and three-stating one while the other drives the signal, depending on whether a read or write transfer is occurring. The write byte enable signals generated in the rambe1 and rambe2 PALs (see Figures 9-8 and 9-9) are buffered onto the signals to the RAMs by controlling the PAL output enables with $\overline{WOE}$. $\overline{WOE}$, the output enable for the write byte enable signals, is basically driven by the buffered read/write (BRW) signal status. The RAMCSH signal is also generated in the bufferoe PAL. It essentially provides a signal for the rest of the logic, which is a constantly asserted RAM chip select throughout the entire transfer, either burst or nonburst. Since $\overline{TS}$ and $\overline{RAMCS}$ are pulsed signals at the beginning of the transfer, RAMCSH establishes a signal that RAMs can latch their enables on.

```
module ENABLESRAMBYTE_pal1;                    flag '-r3','-t1','-x1'
title 'Eval_040 SRAM Byte Enable 0-7 Logic'

          Rambe1 device 'p20r8';     " 10 ns

" Inputs
clk25     pin 1;
A3        pin 2;
A2        pin 3;
A1        pin 4;
A0        pin 5;
SIZ1      pin 6;
SIZ0      pin 7;
~RAMCS    pin 8;
BURST     pin 9;
~TA       pin 10;
L1        pin 11;
L2        pin 14;
~OE       pin 13;
~rst      pin 23;

" Registered Outputs
~EB0      pin 15;
~EB1      pin 16;
~EB2      pin 17;
~EB3      pin 18;
~EB4      pin 19;
~EB5      pin 20;
~EB6      pin 21;
~EB7      pin 22;


" State definition
S0 = (!L1 & !L2);
S1 = ( L1 & !L2);
S2 = ( L1 &  L2);
S3 = (!L1 &  L2);

equations

~EB0 :=   !(((!A3 & !A2 & !A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
          !A3 & !A2 & !A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
          !A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
          !A3 & !A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB0     #
          !~EB0 & ~TA                                         #
          A3 &  A2 & !~TA & S0 &  BURST                       #
          A3 & !A2 & !~TA & S1 &  BURST                       #
          !A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );
```

Figure 9-8. rambe1 PAL Equations (Sheet 1 of 3)

```
~EB1 :=     !((( !A3 & !A2 & !A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 & !A2 & !A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 & !A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB1     #
            !~EB1 & ~TA                                         #
            A3 &  A2 & !~TA & S0 &  BURST                       #
            A3 & !A2 & !~TA & S1 &  BURST                       #
            !A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );

~EB2 :=     !((( !A3 & !A2 &  A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 & !A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 & !A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB2     #
            !~EB2 & ~TA                                         #
            A3 &  A2 & !~TA & S0 &  BURST                       #
            A3 & !A2 & !~TA & S1 &  BURST                       #
            !A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );

~EB3 :=     !((( !A3 & !A2 &  A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 & !A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 & !A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB3     #
            !~EB3 & ~TA                                         #
            A3 &  A2 & !~TA & S0 &  BURST                       #
            A3 & !A2 & !~TA & S1 &  BURST                       #
            !A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );

~EB4 :=     !((( !A3 &  A2 & !A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 &  A2 & !A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 &  A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB4     #
            !~EB4 & ~TA                                         #
            !A3 & !A2 & !~TA & S0 &  BURST                      #
            A3 &  A2 & !~TA & S1 &  BURST                       #
            A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );

~EB5 :=     !((( !A3 &  A2 & !A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 &  A2 & !A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 &  A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB5     #
            !~EB5 & ~TA                                         #
            !A3 & !A2 & !~TA & S0 &  BURST                      #
            A3 &  A2 & !~TA & S1 &  BURST                       #
            A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );

~EB6 :=     !((( !A3 &  A2 &  A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 &  A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 &  A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB6     #
            !~EB6 & ~TA                                         #
            !A3 & !A2 & !~TA & S0 &  BURST                      #
            A3 &  A2 & !~TA & S1 &  BURST                       #
            A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );
```

Figure 9-8. rambe1 PAL Equations (Sheet 2 of 3)

```
~EB7 :=     !((( !A3 &  A2 &  A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS & S0     #
            !A3 &  A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS & S0       #
            !A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0            #
            !A3 &  A2 & SIZ1 &  SIZ0 & !~RAMCS & S0) & ~EB7     #
            !~EB7 & ~TA                                         #
            !A3 & !A2 & !~TA & S0 &  BURST                      #
            A3 &  A2 & !~TA & S1 &  BURST                       #
            A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );

end ENABLESRAMBYTE_pal1;
```

Figure 9-8. rambe1 PAL Equations (Sheet 3 of 3)

```
module ENABLESRAMBYTE_pal2;   flag '-r3','-t1','-x1'
title 'Eval_040 SRAM Byte Enable 7-15 Logic'

                Rambe2 device 'p20r8';   " 10 ns

" Inputs
clk25           pin 1;
A3              pin 2;
A2              pin 3;
A1              pin 4;
A0              pin 5;
SIZ1            pin 6;
SIZ0            pin 7;
~RAMCS          pin 8;
BURST           pin 9;
~TA             pin 10;
L1              pin 11;
L2              pin 14;
~OE             pin 13;
~rst            pin 23;

" Registered Outputs
~EB8            pin 15;
~EB9            pin 16;
~EB10           pin 17;
~EB11           pin 18;
~EB12           pin 19;
~EB13           pin 20;
~EB14           pin 21;
~EB15           pin 22;


" State definition
S0 = (!L1 & !L2);
S1 = ( L1 & !L2);
S2 = ( L1 & L2);
S3 = (!L1 & L2);

equations

~EB8 :=         !((( A3 & !A2 & !A1 & !A0 & !SIZ1 & SIZ0 & !~RAMCS & S0      #
                A3 & !A2 & !A1 & SIZ1 & !SIZ0 & !~RAMCS & S0        #
                A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0             #
                A3 & !A2 & SIZ1 &SIZ0 &  !~RAMCS & S0) & ~EB8       #
                !~EB8 & ~TA                                         #
                !A3 & A2 & !~TA & S0 & BURST                        #
                !A3 & !A2 & !~TA & S1 & BURST                       #
                A3 & A2 & !~TA & S2 & BURST) & ~rst  );
```

Figure 9-9. rambe2 PAL Equations (Sheet 1 of 3)

```
~EB9 :=     !((( A3 & !A2 & !A1 & A0 & !SIZ1 & SIZ0 & !~RAMCS & S0      #
            A3 & !A2 & !A1 & SIZ1 & !SIZ0 & !~RAMCS & S0        #
            A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS & S0             #
            A3 & !A2 & SIZ1 & SIZ0 & !~RAMCS & S0) & ~EB9       #
            !~EB9 & ~TA                                         #
            !A3 &  A2 & !~TA & S0 &  BURST                      #
            !A3 & !A2 & !~TA & S1 &  BURST                      #
            A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );

~EB10:=     !((( A3 & !A2 &  A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS  & S0   #
            A3 & !A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS  & S0       #
            A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS  & S0            #
            A3 & !A2 & SIZ1 &  SIZ0 & !~RAMCS  & S0) & ~EB10    #
            !~EB10 & ~TA                                        #
            !A3 &  A2 & !~TA & S0 &  BURST                      #
            !A3 & !A2 & !~TA & S1 &  BURST                      #
            A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );

~EB11:=     !((( A3 & !A2 &  A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS  & S0   #
            A3 & !A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS  & S0       #
            A3 & !A2 & !SIZ1 & !SIZ0 & !~RAMCS  & S0            #
            A3 & !A2 &  SIZ1 &  SIZ0 & !~RAMCS  & S0) & ~EB11   #
            !~EB11 & ~TA                                        #
            !A3 &  A2 & !~TA & S0 &  BURST                      #
            !A3 & !A2 & !~TA & S1 &  BURST                      #
            A3 &  A2 & !~TA & S2 &  BURST) & ~rst  );

~EB12:=     !((( A3 &  A2 & !A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS  & S0   #
            A3 &  A2 & !A1 &SIZ1 & !SIZ0 & !~RAMCS  & S0        #
            A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS  & S0            #
            A3 &  A2 &  SIZ1 &  SIZ0 & !~RAMCS  & S0) & ~EB12   #
            !~EB12 & ~TA                                        #
            A3 & !A2 & !~TA & S0 &  BURST                       #
            !A3 &  A2 & !~TA & S1 &  BURST                      #
            !A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );

~EB13:=     !((( A3 &  A2 & !A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS  & S0   #
            A3 &  A2 & !A1 &SIZ1 & !SIZ0 & !~RAMCS  & S0        #
            A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS  & S0            #
            A3 &  A2 &  SIZ1 &  SIZ0 & !~RAMCS  & S0) & ~EB13   #
            !~EB13 & ~TA                                        #
            A3 & !A2 & !~TA & S0 &  BURST                       #
            !A3 &  A2 & !~TA & S1 &  BURST                      #
            !A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );

~EB14:=     !((( A3 &  A2 &  A1 & !A0 & !SIZ1 &  SIZ0 & !~RAMCS  & S0   #
            A3 &  A2 &  A1 & SIZ1 & !SIZ0 & !~RAMCS  & S0       #
            A3 &  A2 &  !SIZ1 & !SIZ0 & !~RAMCS  & S0           #
            A3 &  A2 &  SIZ1 &  SIZ0 & !~RAMCS  & S0) & ~EB14   #
            !~EB14 & ~TA                                        #
            A3 & !A2 &  !~TA & S0 &  BURST                      #
            !A3 &  A2 &  !~TA & S1 &  BURST                     #
            !A3 & !A2 &  !~TA & S2 &  BURST) & ~rst  );
```

Figure 9-9. rambe2 PAL Equations (Sheet 2 of 3)

```
~EB15:=     !((( A3 &  A2 &  A1 &  A0 & !SIZ1 &  SIZ0 & !~RAMCS  & S0     #
            A3 &  A2 &  A1 &SIZ1 & !SIZ0 & !~RAMCS  & S0         #
            A3 &  A2 & !SIZ1 & !SIZ0 & !~RAMCS  & S0             #
            A3 &  A2 &  SIZ1 &  SIZ0 & !~RAMCS  & S0) & ~EB15 #
            !~EB15  & ~TA                                        #
            A3 & !A2 & !~TA & S0 &  BURST                        #
            !A3 &  A2 & !~TA & S1 &  BURST                       #
            !A3 & !A2 & !~TA & S2 &  BURST) & ~rst  );

end ENABLESRAMBYTE_pal2;
```

Figure 9-9. rambe2 PAL Equations (Sheet 3 of 3)

As previously mentioned, the ramlwe PAL (see Figure 9-5), generates the BURST signal, which is basically a decode of the size signals for a line access. The $\overline{WE}$ signal is generated during the entire transfer to RAM if it is a write transfer. It is buffered through a 74F244 into five signals and sent to be the write enable on all four RAM banks as well as the direction indicator on the data buffers. There are also four long-word enable signals generated in the ramlwe PAL. Each signal enables an entire long word for a read access. The assertion of $\overline{ELW0}$ enables RAM bank 0, the assertion of $\overline{ELW1}$ enables RAM bank 1, etc. For an initial access, $\overline{ELWx}$ signal is asserted dependant upon whether it is a read RAM access with a long word address that equals x (i.e., A3,A2 = x; where $0 \leq x \leq 3$). However, for burst accesses, the $\overline{ELWx}$ signal is dependant upon the status of the burst state machine and the initial long-word address it accessed. For example, if there is a byte read from a RAM address with A3,A2 = 1, then $\overline{ELW1}$ would be asserted and sent through a 74F244 buffer to assert four byte enable signals, which would enable each byte in RAM bank 1. Similarly, if this were a burst transfer beginning at a long-word address of 1, bank 1 and bank 2 would be enabled immediately; after bank 1 was read it would be disabled, and bank 3 would be enabled while bank 2 was being read. After bank 2 had been read and bank 3 was being read, bank 2 would be disabled and bank 0 would be enabled. Finally bank 0 would be read after bank 3 had been read; then bank 3 would be disabled followed by bank 0. The last signal generated in the ramlwe PAL is that of $\overline{TA}$. Which consolidates all system transfer acknowledges into one for the MC68040.

The third section is the write byte enable section, which is completely contained within the rambe1 and rambe2 PALs. These PALs are shown in Figures 9-8 and 9-9. This section consists of the 16-byte enable signals which are generated and driven onto the RAM enable inputs for RAM write accesses only. If a RAM read access occurs, then the rambe1 and rambe2 PAL outputs are disabled, and the outputs from the $\overline{ELWx}$ signals are allowed to propagate through the 74F244 buffers to enable the appropriate long word(s). Since the memory subsystem is 128-bits wide, there must be a separate byte enable for each of the 16 bytes in the line if the writes are to be controlled by the $\overline{CS}$ input. The logic could be reduced significantly if the write enables controlled the RAM writes in combination with the enabling of a specific bank. This means that the $\overline{WR}$ pulse will negate before the $\overline{CS}$ signal to control the

end of a write, at which time there must be a clean way to negate the $\overline{CS}$ signal within 5 ns after the $\overline{WR}$ pulse negates to avoid the memories driving the data bus at the same time as the MC68040. Although this can be accomplished, the means appeared more expensive than separating the logic. Since it is possible to have a period of time between a RAM read access and a RAM write access in which neither sets of logic are driving the inputs to the RAM $\overline{CS}$ signals, there are pullup resistors on each $\overline{CS}$ input.

The fourth section contains the memory devices. The most important feature of the memory devices used was their density and size. They made it very easy to design a 128-bit memory that is only 64K byte deep. Since the memory devices use the same I/O pins for data and since the MC68040 provides a fairly long initial access time, bus contention problems were a more challenging issue to resolve than access times.

The fifth and last section of the memory subsystem is the address, data, and logic buffers. The address and logic buffers are shown as 74F244s, but other buffers are acceptable. 74F244 data buffers were chosen specifically because they have a disable time which is shorter than the enable time. This specification helps eliminate data bus contention during burst transfers of one clock period when one buffer needs to disable and the other needs to enable once the other is disabled.

Some modifications to this design can improve its simplicity. It was originally designed to provide an easy method to vary either initial or burst access wait states. This was done to evaluate the performance of the MC68040 with different memory subsystems. This is the reason that the whole memory subsystem response to the MC68040 is based on a burst state machine. This feature allows a system designer to vary how soon $\overline{TA}$ is asserted and to control the state of the burst cycle since all data buffering onto the data bus is controlled by the burst state machine. Therefore, by simply varying the one parameter, $\overline{TA}$, the entire memory subsystem will react differently. Many memory subsystem performance levels may be evaluated by adding a specified number of wait states to the system on both the initial access and the burst accesses for both reads and writes.

9.4 Schematics

Figures 9-10—9-15 show the schematics for the SRAM configuration.

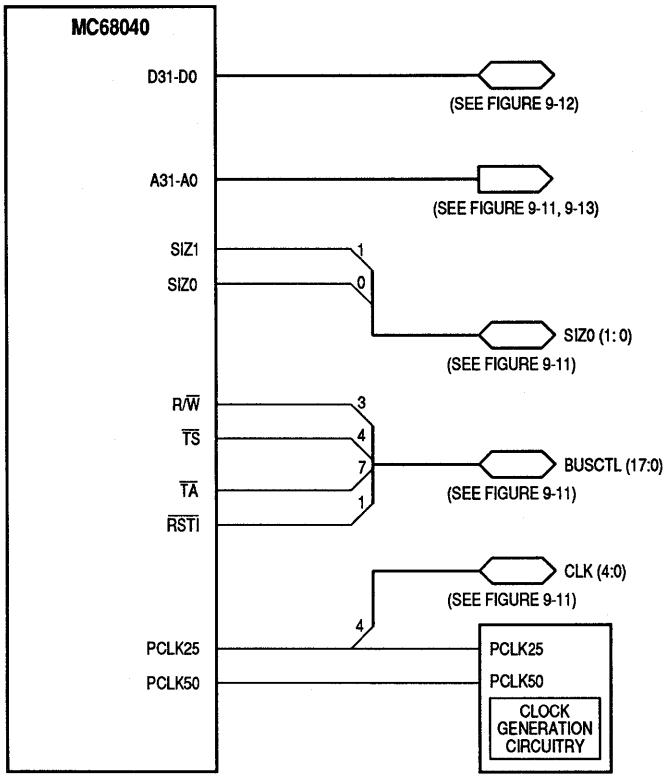


Figure 9-10. MC68040

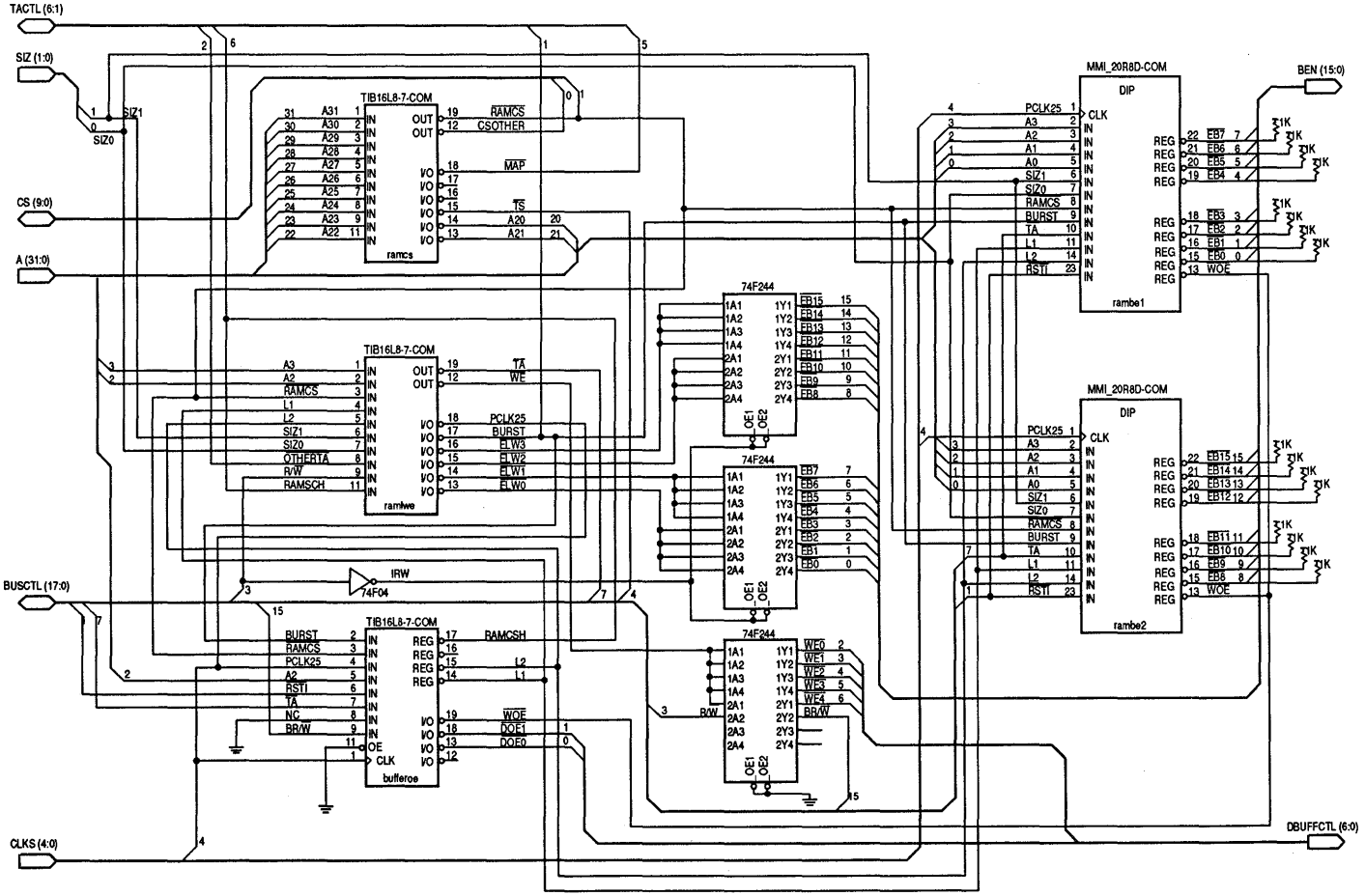


Figure 9-11. Control Unit

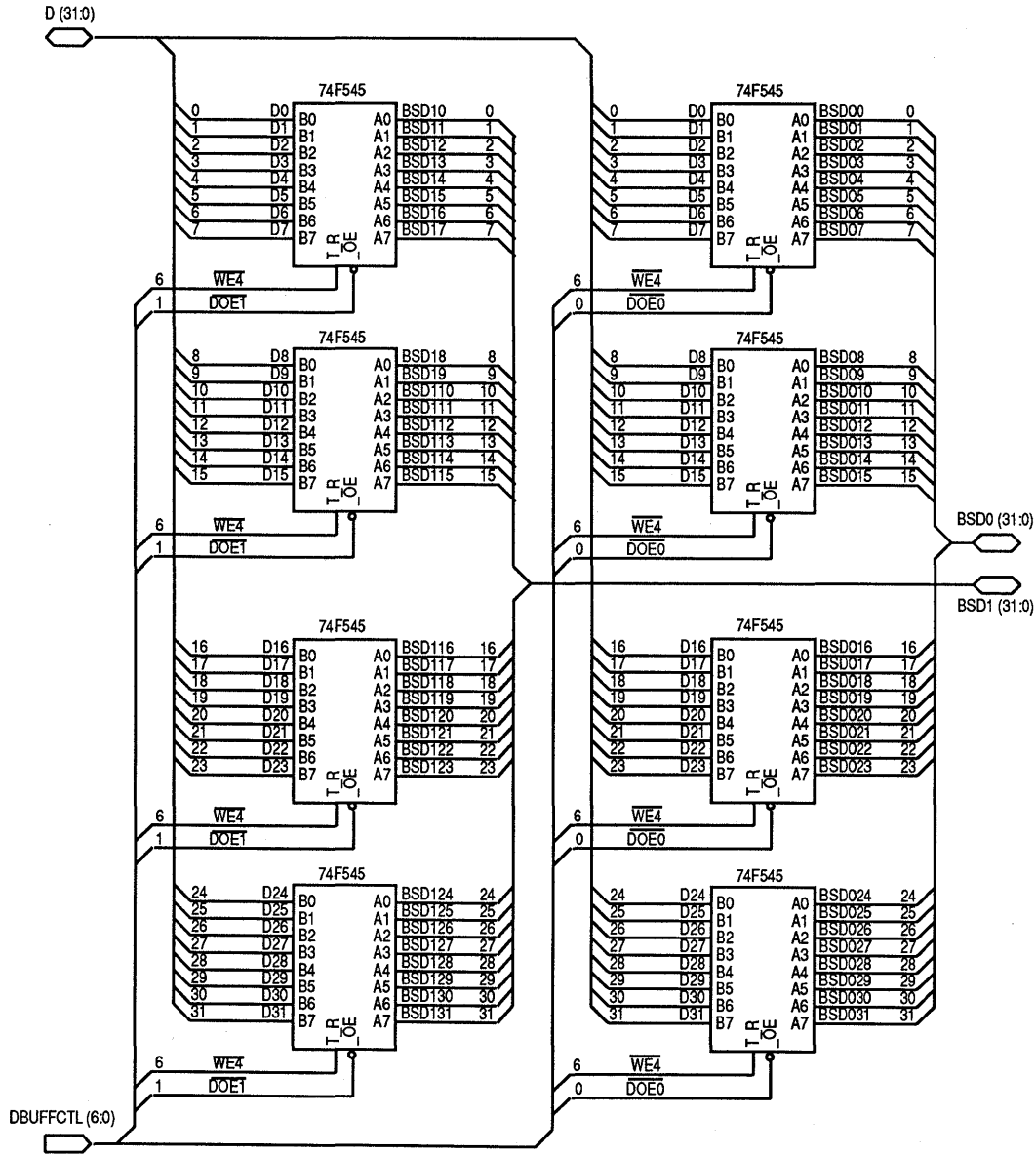


Figure 9-12. SRAM Data Buffers

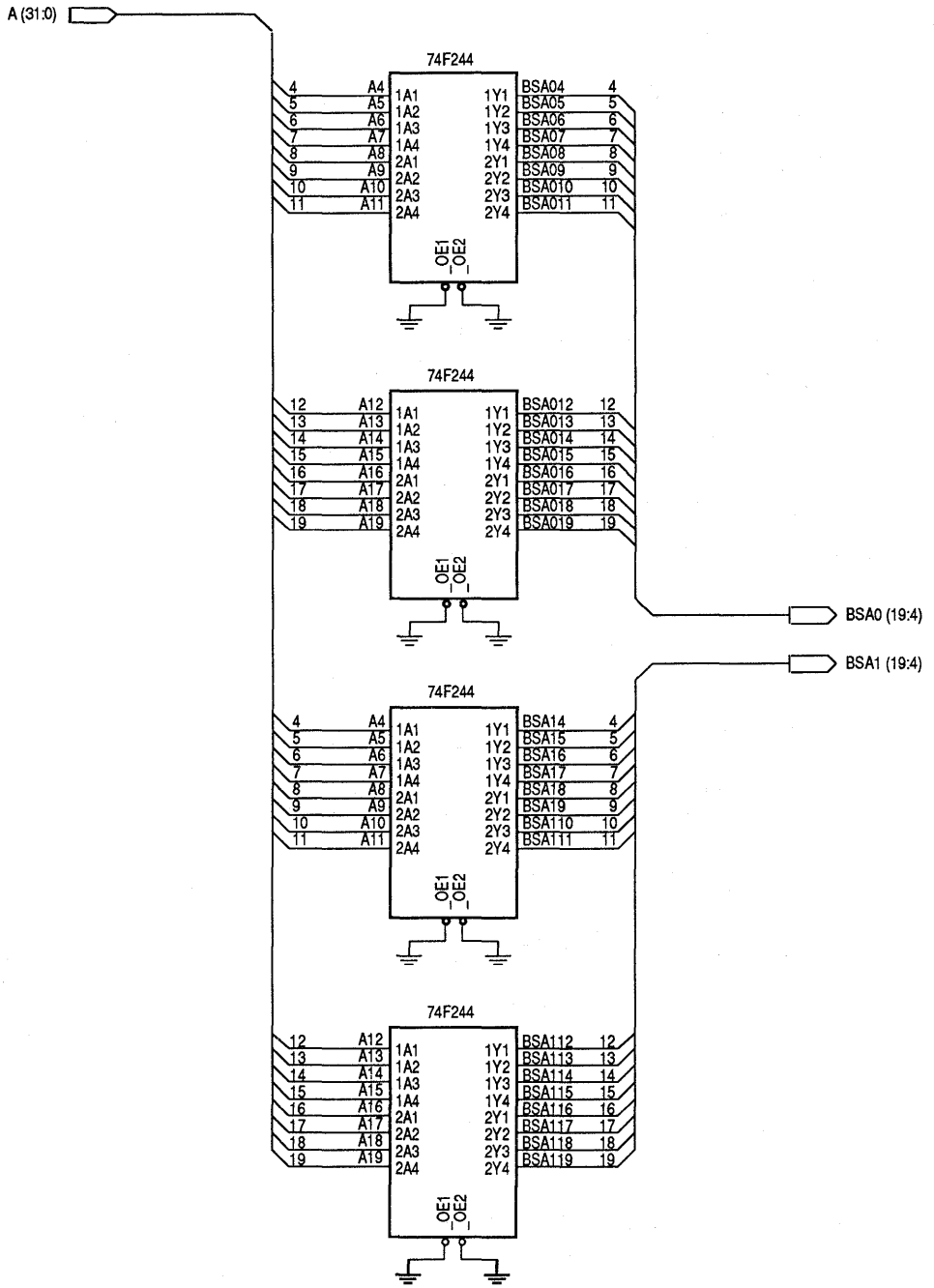


Figure 9-13. SRAM Address Buffers

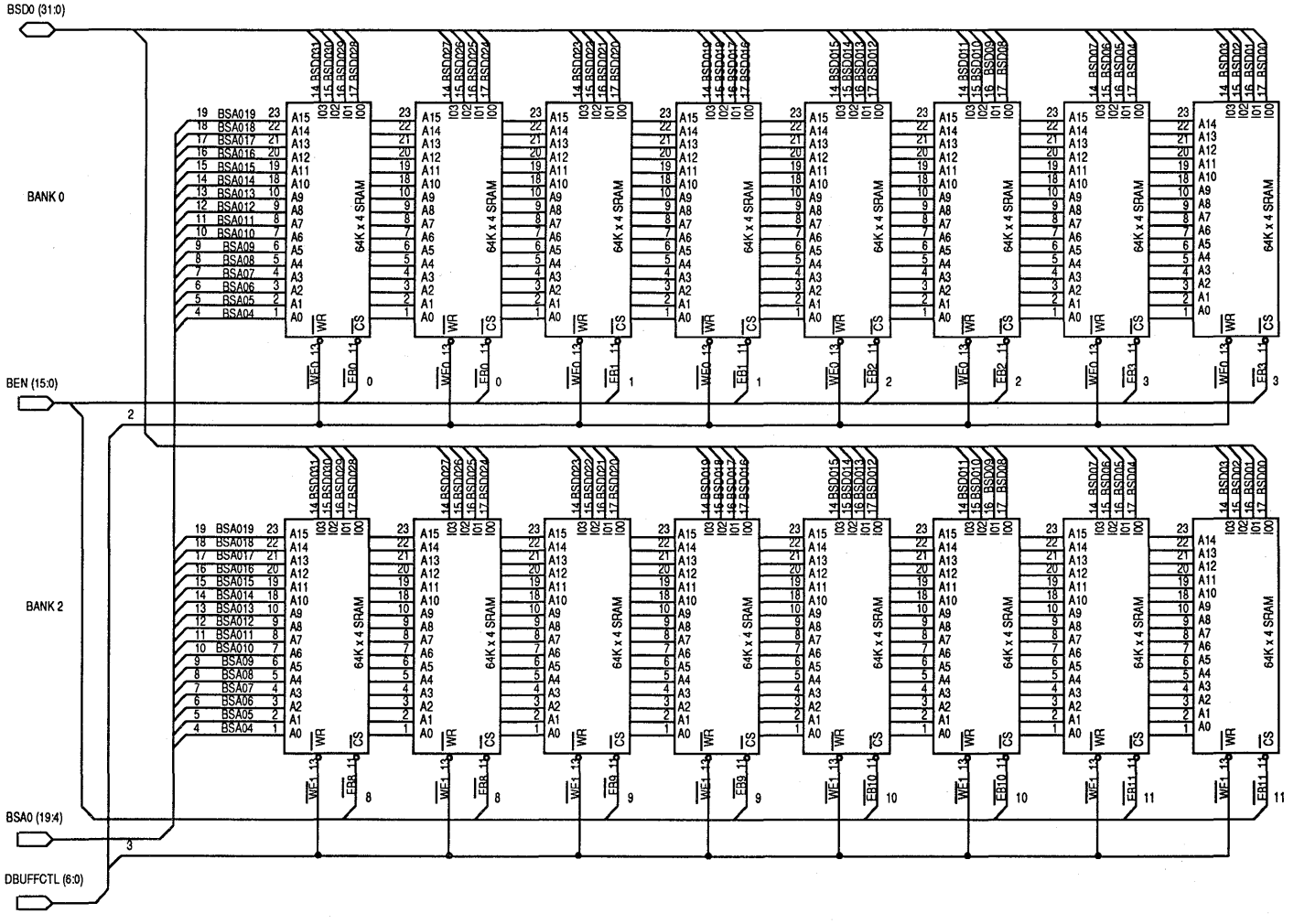


Figure 9-14. SRAM Bank Even

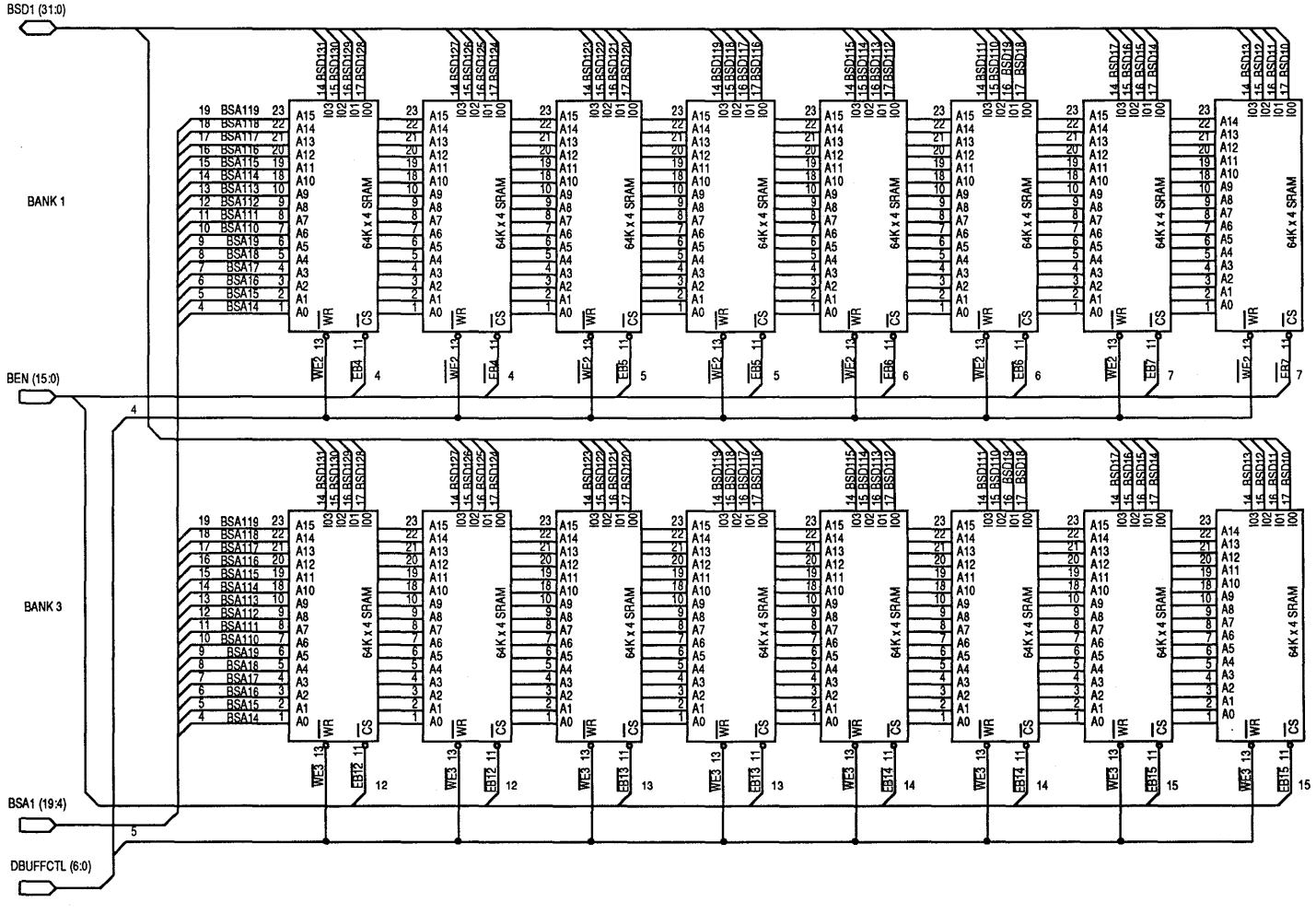


Figure 9-15. SRAM Bank Odd

APPENDIX A
ASCII CHART

LS Dig. \ MS Dig.	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	'	<	L	v	l	|
D	CR	GS	—	=	M	]	m	}
E	SO	RS	.	>	N	Λ	n	~
F	SI	US	/	?	O	—	o	DEL

A.1 HEXADECIMAL AND DECIMAL CONVERSION

How to use:

Conversion to Decimal: Find the decimal weights for corresponding hexadecimal characters beginning with the least significant character. The sum of the decimal weights is the decimal value of the hexadecimal number.

Conversion to Hexadecimal: Find the highest decimal value in the table which is lower than or equal to the decimal number to be converted. The corresponding hexadecimal character is the most significant. Subtract the decimal value found from the decimal number to be converted. With the difference repeat the process to find subsequent hexadecimal characters.

15		Byte		8	7		Byte		0
15	Char	12	11	Char 8	7	Char	4	3	Char 0
Hex		Dec	Hex	Dec	Hex		Dec	Hex	Dec
0		0	0	0	0		0	0	0
1		4,096	1	256	1		16	1	1
2		8,192	2	512	2		32	2	2
3		12,228	3	768	3		48	3	3
4		16,384	4	1,024	4		64	4	4
5		20,480	5	1,280	5		80	5	5
6		24,576	6	1,536	6		96	6	6
7		28,672	7	1,792	7		112	7	7
8		32,768	8	2,048	8		128	8	8
9		36,864	9	2,304	9		144	9	9
A		40,960	A	2,560	A		160	A	10
B		45,056	B	2,816	B		176	B	11
C		49,152	C	3,072	C		192	C	12
D		53,248	D	3,328	D		208	D	13
E		57,344	E	3,584	E		224	E	14
F		61,440	F	3,840	F		240	F	15

APPENDIX B
PRODUCT SUPPORT

This appendix provides information on products supplied by other vendors for the MC68040. Motorola does not assume or guarantee that these products are compatibility with the MC68040.

B1. K680 CLOCK OSCILLATION

Kyocera's K680 clock oscillator was designed to meet the stringent timing requirements of the Motorola MC68040.

The K680 provides both P and B clock outputs in one package eliminating the need for buffering of the P clock output. The P clock drives 50pf and the B clock drives 120pf; ample drivability for many systems.

The advanced circuitry used in the K680 tightly controls symmetry over the entire operating condition range allowing the K680 to easily meet the 47.5/52.5% duty cycle requirement and the ± 9ns CLK1, CLK2 skew specification of the 25 MHz MC68040. Table B1 lists the clock specifications for the K680 clock oscillator and Figure B-1 shows the timing and dimensions.

TABLE B1. Clock Specifications

Characteristic	Symbol	Min	Max	Unit
CLK1 period	t_{PCLK1}	20	TBD	ns
CLK1 rise time	t_{rCLK1}	0	1.7	ns
CLK1 fall time	t_{fCLK1}	0	1.6	ns
CLK1 high level width	t_{hwCLK1}	9.5	10.5	ns
CLK1 low level width	t_{lwCLK1}	9.5	10.5	ns
CLK1 frequency stability	—	-	1000	ppm
CLK2 period (tpCLK2 = 2 x tpCLK1)	t_{PCLK2}	40	TBD	ns
CLK2 rise time	t_{rCLK2}	0	4	ns
CLK2 fall time	t_{fCLK2}	0	4	ns
CLK2 high level width	t_{hwCLK2}	16	24	ns
CLK2 low level width	t_{lwCLK2}	16	24	ns
CLK2 frequency stability (tpCLK2 = 2 x tpCLK1)	—	-	1000	ppm
Skew between CLK1 & CLK2	t_s	-	±7	ns
High level output voltage	V_{OH}	2.0	V_{DD}	V
Midpoint threshold voltage	V_M	1.5	1.5	V
Low level output voltage	V_{OL}	0.0	.8	V
Operating Conditions				
Power supply voltage	V_{DD}	4.5	5.5	V
Operating temperature	T_{OPR}	0	70	°C
Clock 1 output load	CL1	0	50	pF
Clock 2 output load	CL2	0	120	pF

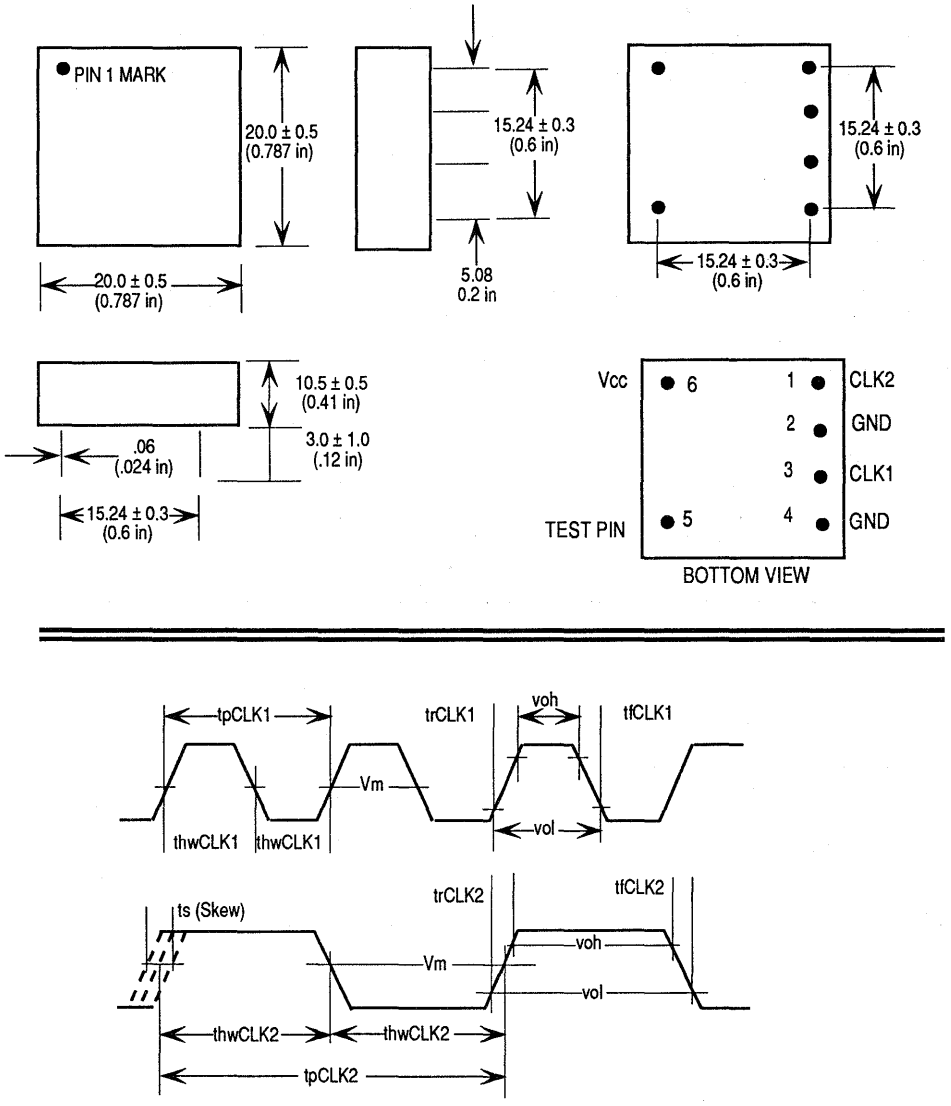


Figure B-1. K680 Timing Diagram

KYOCERA CORPORATION

5-22 Kitainoue-cho, Higashino, Yamashina-ku, Kyoto 607, Japan
Phone: (075)592-3851 Telex: 5422-479 KCJPN J
International Sales Division: Phone (075)594-2211/501-3851
Fax (075)501-0125/581-2332/501-6570

KYOCERA (HONG KONG) LTD.

Room 801-802, Tower 1, S. Sea Centre, 75 Mody Rd.
Tsimshatsui East, Kowloon, Hong Kong
Phone: 3-723-7183 Telex: 35743 Fax: 3-724-4501

2 Shenton Way, 05-09/10 ICB Bldg., Singapore 0106, Singapore
Phone: 221-9966 Telex: 20133 Fax: 220-7813

Chin Hwa Bldg. 9F-2, No. 303, Tung Hwa N. Rd., Taipei, Taiwan, R.O.C.
Phone: (02)718-3595/3596 Telex: 13724 KYOCETWN Fax: (02)718-3587

KYOCERA NORTHWEST, INC.

5701 E. Fourth Plain Blvd., Vancouver, WA 98661
Phone: (206) 696-2840 Telex: 4999180 KYOVAN Fax: (206)695-5836

Los Angeles	Phone: (714)361-6877	Fax: (714)361-6879
San Jose	Phone: (408)437-8777	Fax: (408)437-9534
Vancouver	Phone: (206)699-8746	Fax: (206)695-5836
Indianapolis	Phone: (317)257-3420	Fax: (317)257-9760
Austin	Phone: (512)338-4995	Fax: (512)338-9460
Boston	Phone: (508)820-9722	Fax: (508)872-5177
Ft. Lauderdale	Phone: (305)427-5955	Fax: (305)429-8726

KYOCERA CANADA, INC.

7470 Bath Rd., Mississauga Ontario, Canada L4T 1L2
Phone: (416)671-8942 Fax: (416)671-3746

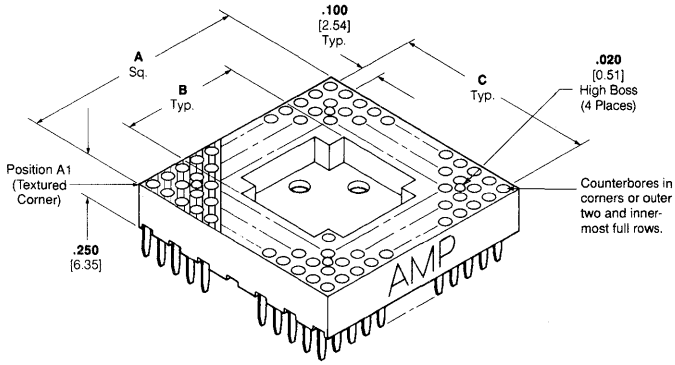
FELDMUHLE KYOCERA EUROPA ELEKTRONISCHE BAUELEMENTE GmbH

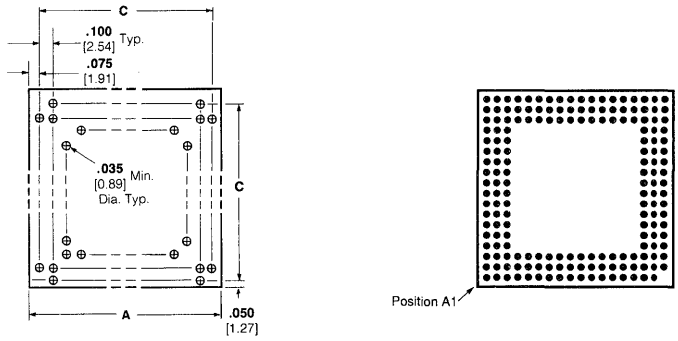
Alleen Strasse 24, D-7300, Esslingen-Zell, Federal Republic of Germany
Phone: (0711)31628-0 Telex: 7-2329931 Fax: 71-136-7599

26 Hemmells, Laindon N. Trade Centre, Laindon, Essex, SS1 56ED, England
Phone: (268)44876/418161 Telex: 99181 Fax: (268)410744

B2. PGA LIF SOCKETS

Figure B2 shows an PGA socket manufactured by AMP Incorporated of Harrisburg, PA. for the MC68040 processor.





Recommended
Pc Board Hole Pattern

Style	Solder Tail Length	AMP Socket Number
LIF-S	.140 3.56	916228-3
LIF-P	.140 3.56	916241-1

Legend:
LIF-P—Low Insertion Force Premium (Beryllium Copper Contacts)
LIF-S—Low Insertion Force Standard (Phosphor Bronze Contacts)

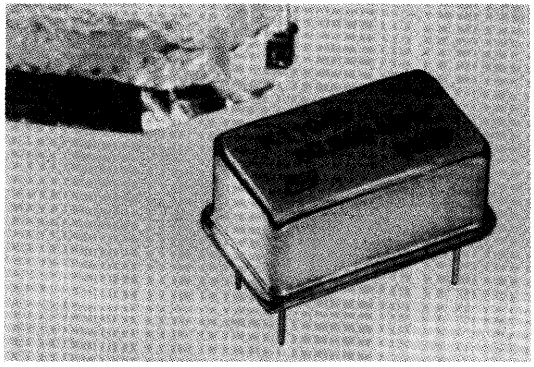
Figure B-2. AMP PGA Socket Package

Champion Technologies, Inc.

Application Note # 154

K11040 Data Clock Oscillators

40 to 70 MHz



Description

The K11040 Data Clock Oscillator is specifically designed to provide the highest system performance for the demands of today's microprocessor based applications. Initially designed in cooperation with Motorola as a 50 MHz data clock for the MC68030 and MC68040 families of 32-bit microprocessors, the K11040 features an extremely tight symmetry of 47.5% minimum and 52.5% maximum — a specification well above current industry standards. Additionally, at a temperature of 25°C with a 5V supply, nominal symmetry improves to the 49/51% level.

For meeting the tight electrical specifications of the high speed Motorola MC68030 and MC68040, a rise time of 1.7 nsec max and a fall time of 1.6 nsec is guaranteed. Meeting these stringent requirements for duty cycle and rise/fall times also makes the K11040 crystal oscillator ideal for other applications as well. With proper matching techniques, the K11040 will drive capacitive loads ranging from 25 pF through 75 pF. Figure 1 shows a typical waveform pattern of the K1040 driving a 50 pF capacitive load.

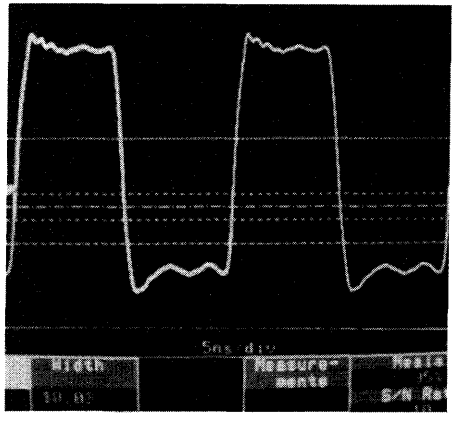
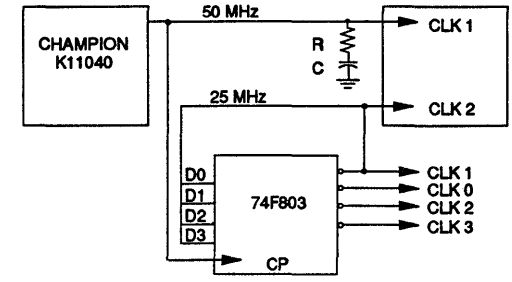


Fig. 1— K11040 Output Waveform Driving 50 pF Load

Various system application techniques will be discussed later in this application note. Recommended application designs are shown below in figures 2 and 3, along with the recommended test fixture shown in figure 4. These applications actually represent the least expensive approach for driving digital systems by avoiding the use of costly frequency-divider ICs and synchronization circuitry needed with multiple clock systems. *With the superior stability and noise immunity provided by the K11040 clock driver comes a substantial reduction in the overall cost of system design!*



C = 220 pF high quality ceramic, with low lead inductance
R = 40 - 80 ohm depending on characteristic impedance of PCB
For 68030 25 MHz signal generation not required

Fig. 2— System Configuration for Driving Motorola MC68030 and MC68040 Processors

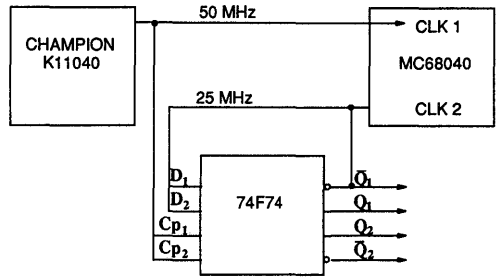


Fig. 3—Alternate System Configuration for Driving MC68030/40 Processors. CLK 2 Requires Only 60/40 Symmetry, Which Can Be Achieved With the 74F74

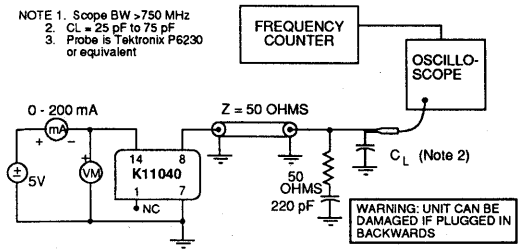


Fig. 4— Recommended Test Fixture for K11040 Properly Terminated in 50 Ohm System

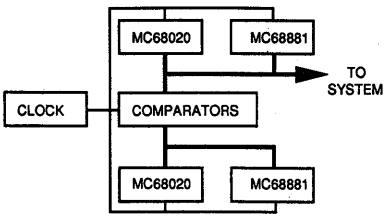


Fig. 5— K11040 Clock Redundant System Elements in Typical CPU Environment

CMOS Output Provides Improved Design Stability

Exceeds Standard TTL Drivers

The CMOS output of the K11040 oscillator provides maximum stability for high speed digital systems. Improvements over standard TTL logic include:

- Superior drive levels for synchronization of input signals to various digital devices.
- Maximum noise tolerance for "fault tolerant" systems using multiple processing elements.
- Sharper rise and fall times for increased stability of internal clock signals.

The K11040 driver was designed for "high data integrity" in fault tolerant computing environments. Where synchronization between all processing elements is important, the multiple driving capability of the K11040 insures controlled phase and skew relationships for full lock-step operation. This may not be feasible with TTL drivers because of the high capacitive loading in large fan-out systems. Using proprietary technology developed at Champion Technologies for maintaining edge rate and symmetry specs for loads as high as 100 pF, the K11040 is uniquely capable of operating redundant processing elements without causing incorrect operation of the system. Additionally, the sharper rise/fall times and wider voltage swing of the K11040 facilitates stable internal clock signals for digital elements such as the redundant processing element shown in figure 5.

In order to guarantee recognition of a logic state by all redundant processors, the clocking signal must remain constant for the setup and hold time required by synchronized processors for lock-step operation. Using standard TTL output devices with slower edge rates reduces the stability of processor operation. In systems requiring critical speed paths for synchronous operation, actual metastable levels can develop, causing system errors. The synchronization requirement can be provided by the K11040 oscillator in high-speed systems by clocking inputs of sufficient duration to meet the processors' setup and hold time. The sharp edge rates guaranteed for the K11040 under varied load conditions provide this stability.

System Applications

To fully utilize the advantages inherent in the fast rise and fall times of the K11040, the system designer must have a good understanding of the oscillator's performance capabilities within a variety of system environments. With the faster edge rates used, the integrity of the output waveform becomes increasingly dependent on system interconnections, loading, and the presence of transmission line characteristics. These areas of concern may be new to many system engineers unfamiliar with the high speed CMOS output of the K11040. However, certain system modifications can be made to guarantee performance values for this oscillator in multiple computing environments. Items of interest which need to be considered when using the K11040 data clock are:

- ***overshoot and ground bounce***
- ***transmission line effects***
- ***capacitive loading and fan-out***

It should be noted here that these characteristics are all inter-related. Capacitive loading not only bears a direct relationship to ground bounce and transmission line effects, but also changes propagation delay, rise time and other circuit parameters.

Overshoot and Ground Bounce

As the oscillator output switches from high-to-low and from low-to-high, large currents quickly change at the CMOS output. The effect of these current transitions contributes to a voltage developed across the parasitic inductances of the bondwires and lead frames used to house the K11040 oscillator (see figure 6). Additionally, stray inductances will be present in any non-ideal resistor or capacitor leads as well as within the devices themselves.

Severe ground bounce can cause the effective output ground to increase by several volts. This increases the possibility for false triggering whereby low level input signals rise several volts in magnitude with respect to the new ground level. Figure 7 illustrates the severe ringing that can result from excessive ground bounce.

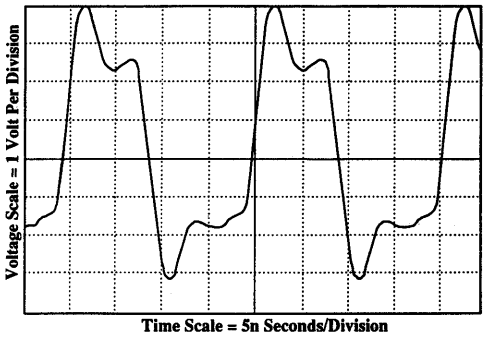


Fig. 6— K1040 CMOS Output Driving Non-Ideal Lumped Circuit Components With Stray Parasitics

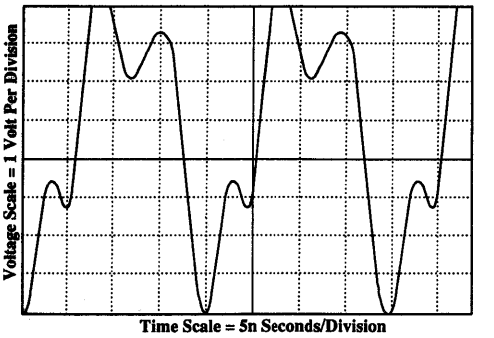


Fig. 7 — Unterminated K11040 Output With Ground Bounce and Overshoot

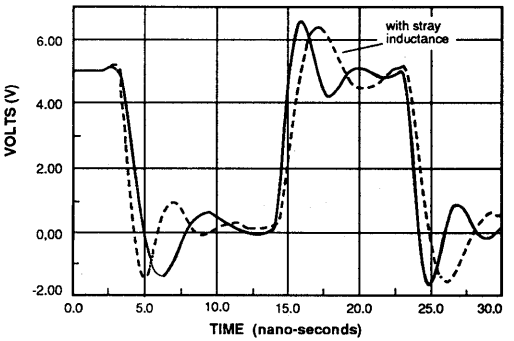


Fig. 8A— Effect of Inductance on Ground Bounce With a Transmission Line Terminated by a 30 pF Load

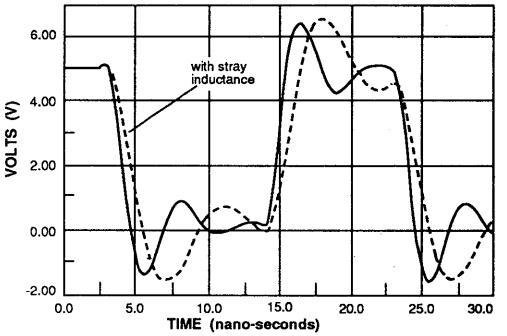


Fig. 8B— Effect of Inductance on Ground Bounce With a Transmission Line Terminated by a 50 pF Load

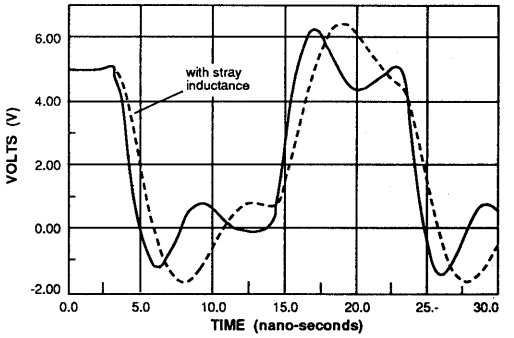


Fig. 8C— Effect of Inductance on Ground Bounce With a Transmission Line Terminated by a 75 pF Load

Several factors which affect the amplitude of ground bounce are:

1) *The number of outputs switching simultaneously*; more outputs will result in increased ground bounce effects.
2) *Use of test fixtures;* the stray inductance and capacitance of many fixtures greatly exaggerate ground bounce problems. The effect of inductance on ground bounce is shown in figures 8A, 8B and 8C for several capacitance terminations.
3) *Capacitive loading;* ground bounce is sometimes more than doubled with a load capacitor. As the load capacitor charges and discharges, the current $I = C_L \frac{\delta V}{\delta T}$ generated creates a larger voltage across the stray inductance $V = L \frac{\delta I}{\delta T}$. See figure 9.

Although ground bounce cannot be eliminated, simple precautions can be taken to minimize ground bounce problems.

1) Avoid the use of sockets or wirewrap when possible.
2) Use Vcc and ground planes when possible.
3) Decouple every device with 0.1 μF high-Q capacitors using RF dielectric material.

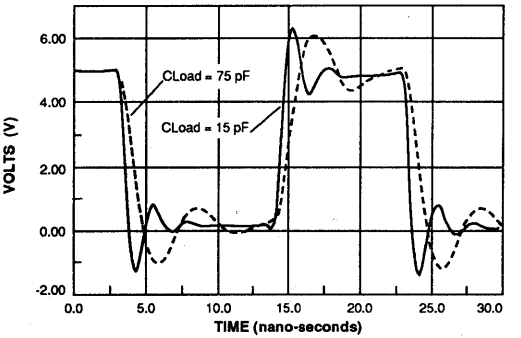


Fig. 9—SPICE Simulation Showing the Effect of Capacitance on Ground Bounce

4) Locate the loading device as close to the oscillator output as possible.

Figures 10A and 10B show the improvements made to the output waveform using these simple design recommendations.

Transmission Line Effects

With the high speed edge rates of the K11040 output, transmission line effects become more significant. Figures 11A and 11B illustrate the effect of transmission line characteristics on unmatched and matched loads. These SPICE simulations show the K11040 output signal into a 60 Ohm transmission line with a 50 pF load. Figure 11A contains no matching network. The transmission line in figure 11B is matched with a series combination of a 60 Ohm resistor and a 220 pF capacitor in parallel with the load.

Transmission line properties generally become prominent when edge rates are less than three times the propagation delay of the line. Evaluation of sample K11040 units indicate rise and fall times as fast as 500 psec into a 50 pF load. Assuming a typical substrate dielectric constant of 5 for a printed circuit board, the minimum line length between oscillator and load for transmission line effects to be considered is approximately 1.5 cm. Additionally, loading the transmission line effectively decreases its characteristic impedance. This increases the likelihood of transmission line effects for short line lengths between oscillator output and load. Figure 12 shows the decrease in characteristic impedance as a function of capacitive loading for several impedances.

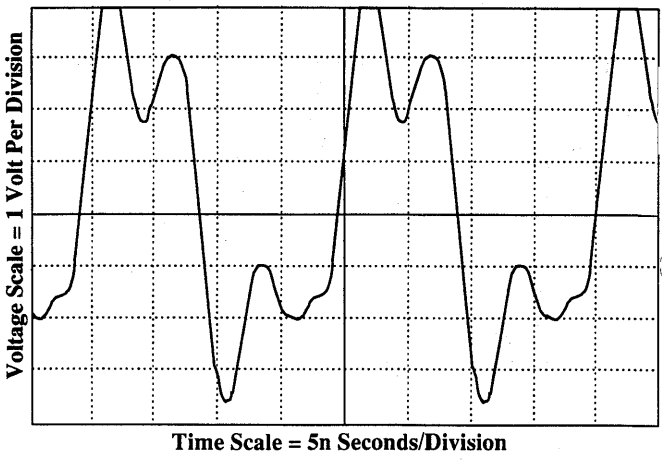


Fig. 10A—K11040 Output Waveform With Stray Inductances

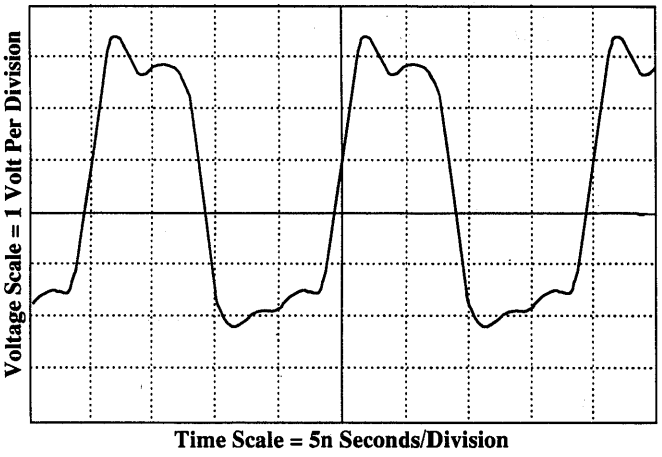


Fig. 10B— K11040 Output Waveform With Stray Inductance Minimized

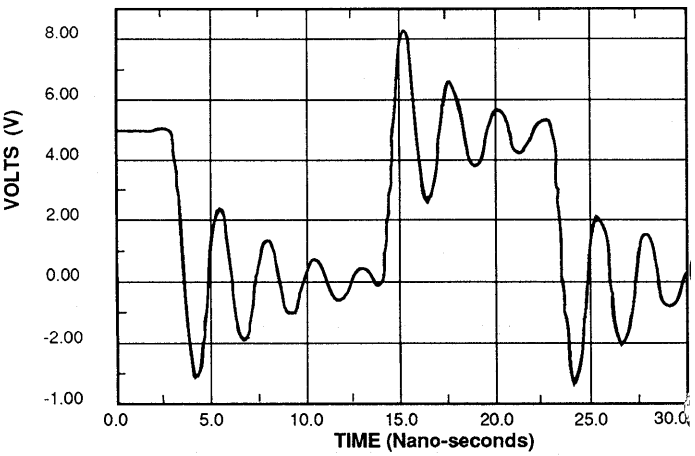


Fig. 11A— SPICE Simulation: K11040 Output Into 60 Ohm Transmission Line; 50 pF Load, No Matching Network.

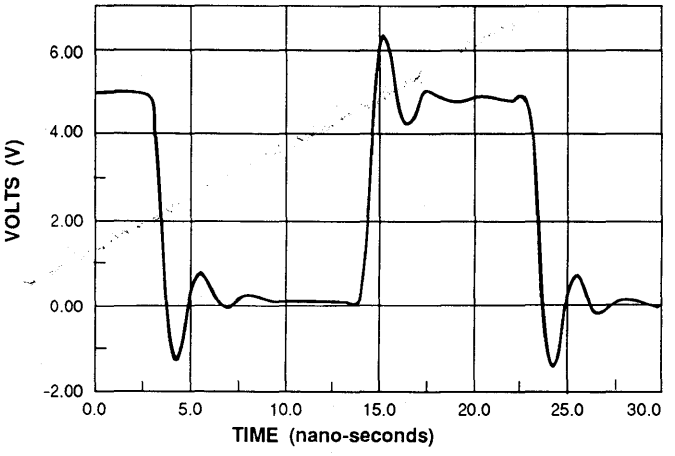


Fig. 11B— SPICE Simulation: K11040 Output Into 60 Ohm Transmission Line; 50 pF Load, Series 60 Ohms and 220 pF In Parallel With The Load

The decrease in transmission line characteristic impedance with capacitive load also increases the propagation delay of the signal. The subsequent increase in propagation delay can be determined by the formula $\tau'_{pd}=\tau_{pd}Z_0/Z_{0e}$, where:

τ'_{pd} = the loaded propagation delay

τ_{pd} = the unloaded propagation delay

Z_0 = the unloaded characteristic impedance

Z_{0e} = the loaded characteristic impedance

Figure 13 also illustrates this change in propagation delay as a direct function of load capacitance.

Using rise/fall times in the 500 psec range, the maximum line lengths between the K11040 and load can then be calculated for determining whether transmission line effects should be considered.

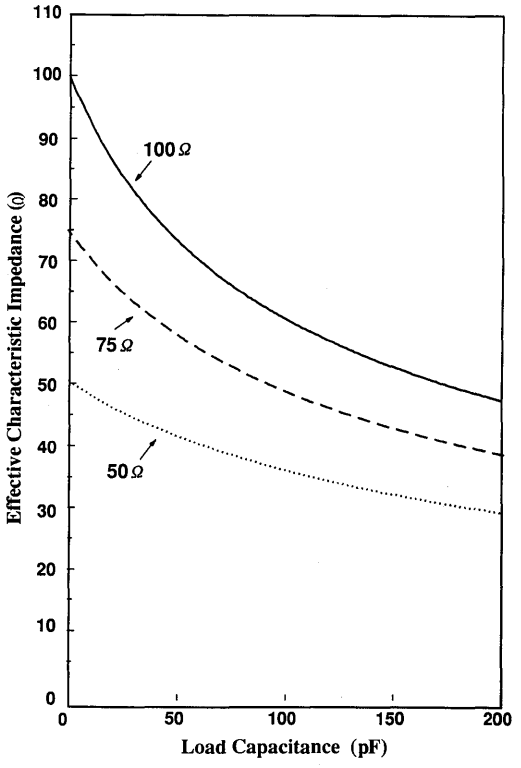


Fig. 12— Change In Effective Characteristic Impedances As a Function of Load Capacitance for Three Line Impedances

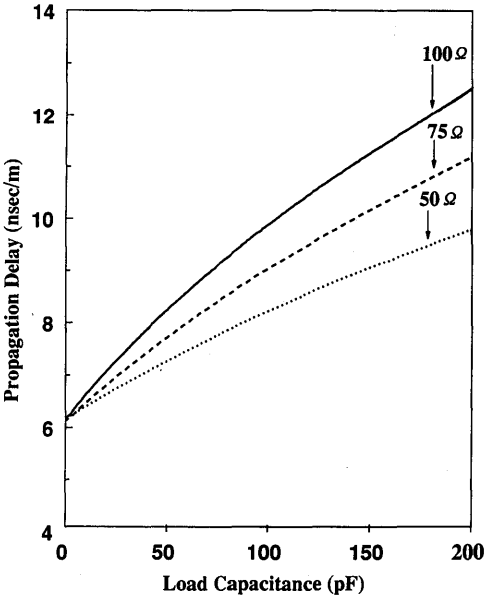


Fig. 13— Propagation Delay As a Function of Load Capacitance for Several Line Impedances

Line Driving

This section discusses transients in transmission lines for capacitive termination, which is the most common loading seen in high speed digital systems. Capacitive loads permit no instantaneous change in output voltage V_L. Thus, V_L remains zero immediately after the input signal reaches the load. After a period of time, the output eventually changes into an open circuit. This transition between the two voltage levels is an exponential with a time constant that is the product of the load capacitance and the resistance it sees, i.e., the characteristic impedance of the line. Since the capacitive load is effectively a short circuit immediately after a voltage transition, a voltage $V_r=-V_{in}$ is reflected back to the source. The magnitude of this reflected voltage approaches V_{in} with a time constant equal to the product $Z_0 \cdot C_L$.

In predicting output waveforms for transmission lines terminated with nonlinear loads, graphical methods have been developed to determine transients. These methods originate with Bergeron[1]. Figure 14A is a SPICE simulation showing the current/voltage characteristics of the K11040 CMOS output switching low-to-high and high-to-low. Figure 14B and 14C illustrate the predicted voltage levels for the K1140 when switching from both high-to-low and low-to-high for a 60 ohm transmission line. The theoretically predicted levels of overshoot and undershoot can be seen at both K11040 output and load positions.

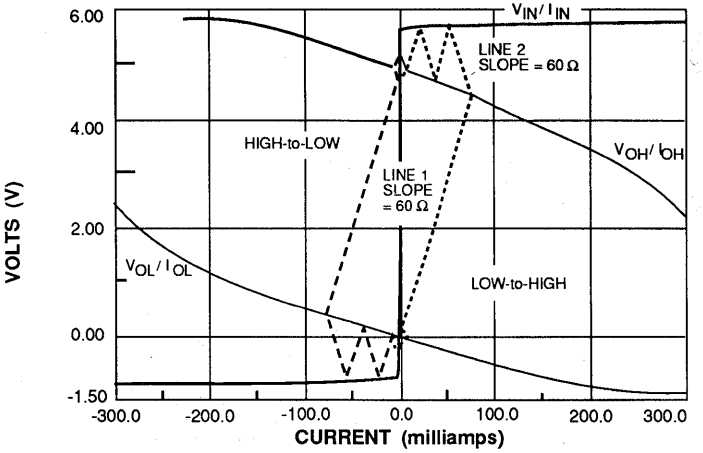


Fig. 14A— SPICE Simulation of I-V Characteristics of K11040 CMOS Outputs From "0" to "1" and "1" to "0"

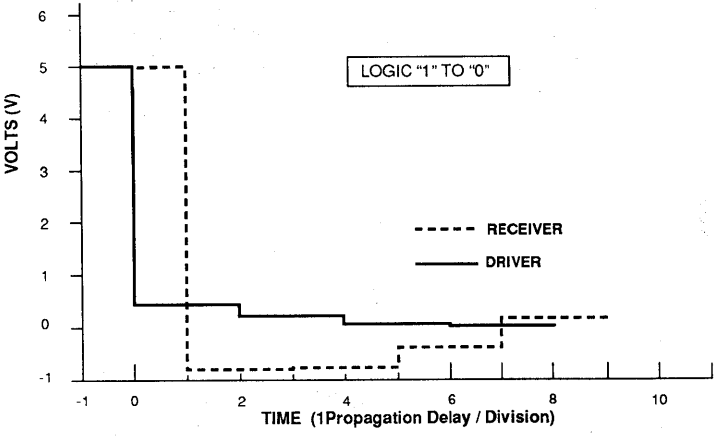


Fig. 14B— Predicted Undershoot Level for Logic "1" to "0." A 60 Ohm Transmission Line Is Assumed

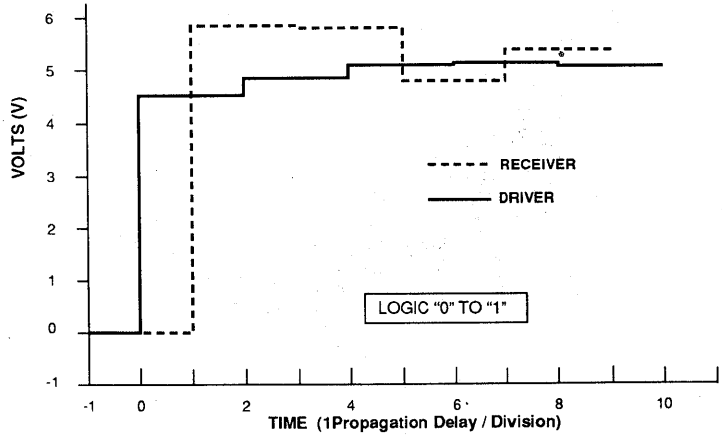


Fig.14C— Predicted Undershoot Level for Logic "0" to "1." A 60 Ohm Transmission Line Is Assumed

Matching Networks

There are many termination schemes that can be used for properly matching the characteristic impedance of the transmission line. Several of these techniques include the series, parallel, AC parallel and Thevenin terminations. These matching networks are shown in figure 15.

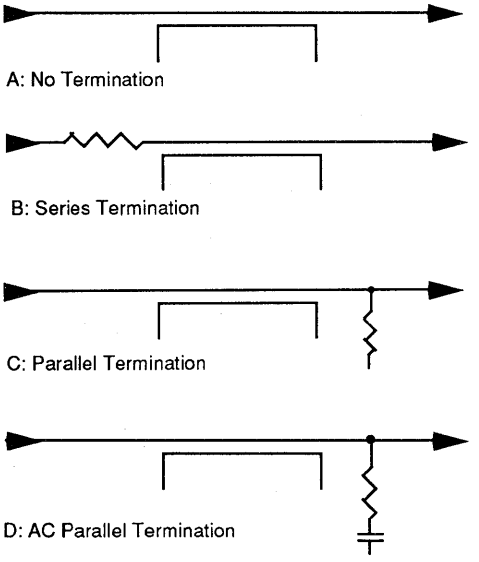


Fig. 15—Termination Schemes for Matching K11040 Output to Transmission Line

The AC parallel termination is recommended for use with the K11040. This network avoids the delays caused by series termination while reducing dc power consumption with the aid of a dc blocking capacitor.

The values chosen for the terminating resistor and capacitor are determined by the effective characteristic impedance of the transmission line. Typically, the resistor should be the same value as the effective line impedance. Although the size of the blocking capacitor is somewhat arbitrary, experimental testing has demonstrated that a value of 220 pF provides good matching properties. At the oscillating frequency of 50 MHz, the impedance is negligible. At the same time, the 220 pF capacitor is not too large where the excessive internal inductance within the non-ideal capacitor would become a problem. Figures 16, 17 and 18 compare the effects on symmetry and waveform integrity using several matching capacitors in a transmission line environment.

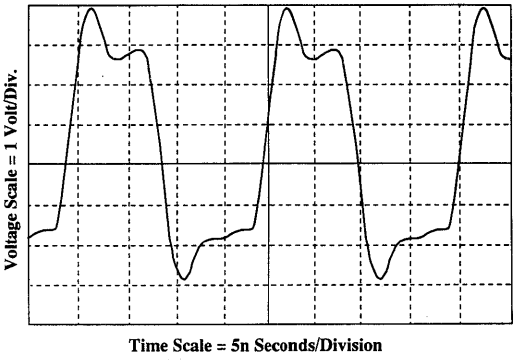


Fig. 16— K11040 Waveform Using a 100 pF Matching Capacitor

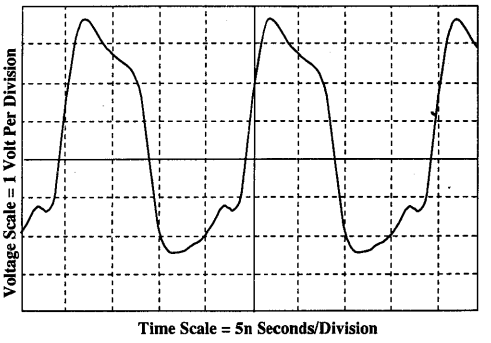


Fig. 17— K11040 Waveform Using a 140 pF Matching Capacitor

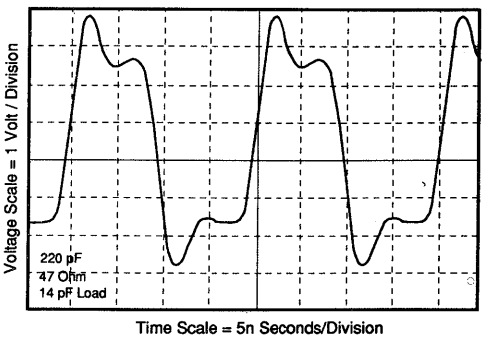


Fig. 18— K11040 Waveform Using a 220 pF Matching Capacitor

In order to minimize stray inductances in the system, matching component leads must be made as short as possible. Resistors with one inch leads may have inductances as large as 10 nH. Additionally, capacitors with good RF dielectrics should be used to maintain low noise levels in the system. The consequences of mismatching the K11040 output driving a capacitively loaded transmission line can readily be seen in the digitized waveforms of figures 19A, 19B and 19C, which show the K11040 driving a 15 pF load in a 50 ohm transmission line system.

SPICE simulations were also compiled showing the change in symmetry and rise time for the K11040 output in a 60 Ohm transmission line system. Symmetry is calculated at the 1.5 Volt level while rise time is determined between the 0.8 and 2.0 Volt levels. Interpolation was performed on the SPICE files using the point-slope formula. The plots are shown in figures 20 and 21.

The ability to terminate a line effectively is also a function of how constant the characteristic impedance is over the line's length. To insure this controlled impedance, ground planes should be used whenever possible. Additionally, the use of external connectors and vias which cause line mismatches should be avoided. Multiple reflections which severely degrade the signal can occur where large mismatches are introduced into the system. Connectors such as BNC or 3 mm SMA should be used for direct microstrip-to-coax connections.

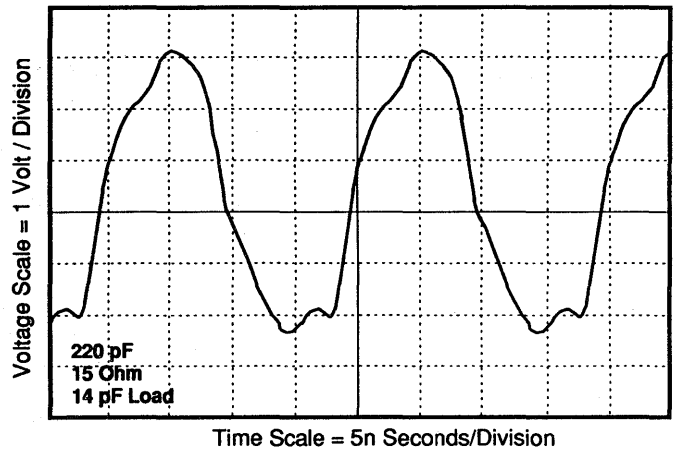


Fig. 19A— Output Waveform, AC Parallel Matching Network With 15 Ohm Resistor

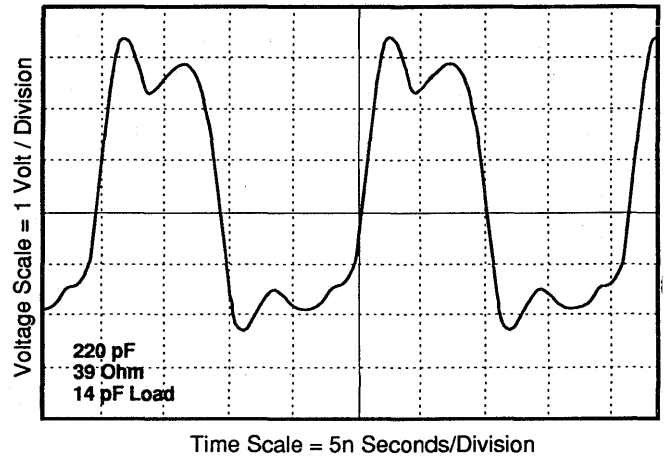


Fig. 19B— Output Waveform, AC Parallel Matching Network With 39 Ohm Resistor

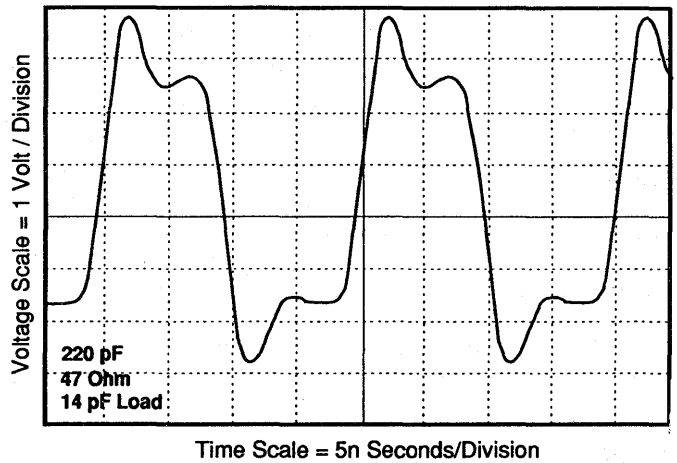


Fig. 19C— Output Waveform, AC Parallel Matching Network With 47 Ohm Resistor

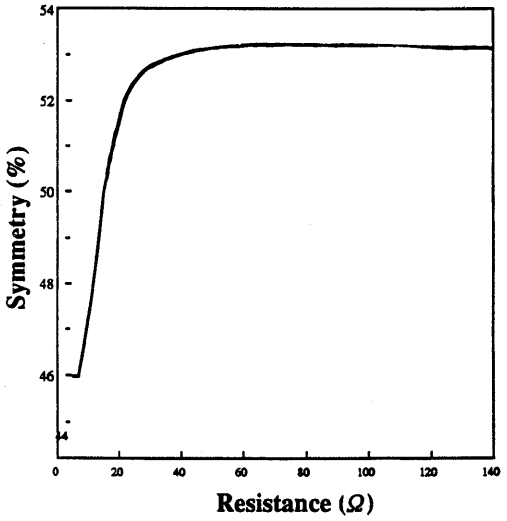


Fig. 20— SPICE Simulation: Symmetry as a Function of Matching Load Resistor (60 Ohm Transmission Line)

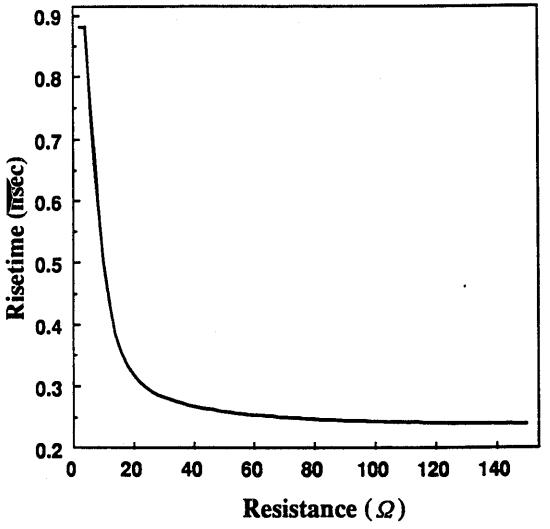


Fig. 21— SPICE Simulation: Rise Time as a Function of Matching Load Resistor (60 Ohm Transmission Line)

Capacitive Loading and Fan-Out

Capacitive Loading of transmission lines effectively increases the propagation delay of the system. In high-speed digital environments, timing can be a critical factor such that the effects of gate loading and large fan-out must be determined.

Procedures for terminating transmission lines with various capacitive loads should be used as outlined in this application note. To insure similar signals on all lines in a bus, all lines must be terminated equally.

Electromagnetic Considerations

Digital devices which function in a switching mode are common sources of electromagnetic interference (EMI). With the K11040 crystal oscillator, additional precautions must be taken because of the high-speed switching action of the signal. As the clock edge rates sharpen, the harmonic content of the waveform increases proportionately. At rise times in the 300 to 400 psec range, signals may have up to 25% of their total energy in frequencies above the second harmonic. Radiated interference then results both from the oscillator casing itself as well as from interconnecting leads which act as miniature antennas.

To insure compliance with FCC standards for RF emissions, the mechanical design of the K11040 has been developed to minimize electromagnetic radiation. Each oscillator unit is fitted within a hermetically sealed all-metal package and must maintain a mass spectrometer leak rate of less than $2X10^{-8}$ atmosphere cc/sec of helium to insure that it is mechanically tight to EMI. Large impedances at the seam would allow high frequency voltages to develop, thus allowing RF energy to propagate. A process of continuous welding is therefore used whereby the RF impedance of the seam matches that of the adjoining materials.

When an electromagnetic plane wave strikes a conductive surface, it loses energy through both reflection and absorption. Reflective losses occur due to the discontinuities in characteristic impedances at the media boundaries between free space and the metal oscillator enclosure. Energy is also absorbed within the metal through Ohmic losses which transform the RF energy into heat. To produce maximum suppression of emitted RF interference, the metal casing housing the K11040 is composed of highly conductive materials to reflect EM propagation.

It is important to remember, moreover, that because of the "antenna effect," interconnection conductors can often radiate more interference than the oscillator itself. Since the power radiated by these conductive strips is directly proportional to their physical length, conductors should be kept as short as possible. Antennas whose physical length is much less than 1/2 wavelength are generally not well matched to their sources or to the surrounding environment. Therefore, unless special designs are developed, conductors will become inefficient radiators because of this mismatching with their sources. In addition to reducing conductor length where possible, the length of IC leads should be minimized. ICs with unterminated long leads are susceptible to propagation of electromagnetic radiation and inductive coupling to surrounding media.

Many additional techniques for reducing unwanted RF emissions parallel those mentioned in the transmission line section of this application note. Proper terminations must be designed for each digital system relative to capacitive load, fan-out, and conductor characteristic impedance. Systems using large ground planes may also contain differences in electric potential between different points because of currents flowing into and out of the ground plane. Ground leads must therefore be kept sufficiently short to avoid adverse coupling through the ground plane. Such ground plane currents could easily be induced into various digital circuits, causing unwanted system noise. Decoupling with high qual-

ity bypass capacitors will serve to minimize induction of these load-induced transients into the power supply bus, other cascaded stages, and bias supplies. Where feasible, separate ground returns can be maintained for signals, power systems and case grounds. This will serve to reduce transient coupling into other circuits. Circuit configuration can also be designed to reduce coupling by maximizing the physical separation between conductors to reduce "cross-talk." Where signal and power leads must cross, the crossing should be designed such the the conductors are perpendicular to each other. This reduces cross-talk interference and noise within the power supply bus.

Biography

Mark Heimbach

Mark is an RF Design Engineer in the Product Development Engineering Department at Champion Technologies, Inc. He is a graduate of the University of Wisconsin - Madison with a Masters Degree in Electrical Engineering. At Madison, his research work included antenna radiation analysis and investigation of electric field patterns propagating in circular waveguides. Mark is currently working on high frequency oscillator design at Champion. He is a member of the IEEE and Tau Beta Pi.

Bibliography

1. M. Abdel-Latif and M.. J. O. Strutt, "Simple Graphical Methods to Determine Line Reflections Between High-Speed-Logic Integrated Circuits," Electronic Letters, 4, 496-498 (November 1968).
2. W. Blood, Jr., "MECL System Design Handbook," Motorola Inc. (1988).
3. Motorola Inc, "Motorola FACT Data," Motorola Technical Information Center (1988)
4. A. Barna, "High Speed Pulse and Digital Techniques" (1980).
5. D. Hodges and H. Jackson, "Analysis and Design of Digital Integrated Circuits" (1983).
6. Motorola Semiconductor Group, "Considerations for System Design Using Redundant MC68020/MC68881 Pairs for High System Integrity," Motorola Engineering Bulletin EB115 (1987).
7. Morrison, R., "Grounding and Shielding Techniques in Instrumentation," Wiley and Sons (1967)

Champion Technologies, Inc.
2553 N. Edgington Street
Franklin Park, IL 60131
708/451-1000 FAX: 708/451-7585
EasyLink: 62931824 Telex: 499-0104 CTI UD

A25548-1 PRINTED IN USA 11/90 IMPERIAL LITHO 75870 20,000 MPU YGABAA