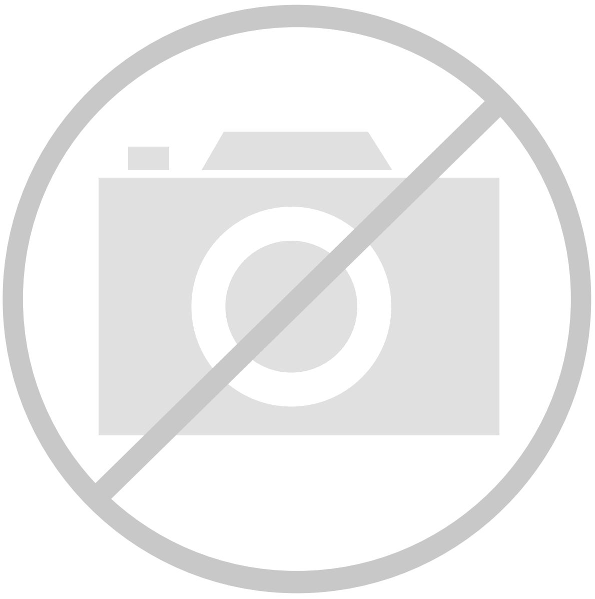
Motorola - MC68LC040RC33A Datasheet



Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Active
Core Processor	68040
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	33MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	179-BPGA
Supplier Device Package	179-PGA (47.24x47.24)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68lc040rc33a

TABLE OF CONTENTS (Continued)

- The DLE pin name has been changed to JS0 on both the MC68040V and MC68LC040. In addition, the MC68040V contains three new pins, system clock disable (SCD), low frequency operation (LFO), and loss of clock (LOC).
- The MC68040V and MC68LC040 do not implement the data latch enable (DLE), multiplexed, or output buffer impedance selection modes of operation. They implement only the small output buffer mode of operation. All timing and drive capabilities on both devices are equivalent to those of the MC68040 in small output buffer impedance mode. The MC68040V has an additional mode of operation, the low-power stop mode of operation.
- The MC68040V and MC68LC040 do not contain an FPU, causing unimplemented floating-point exceptions to occur using a new stack frame format.
- The MC68040V is a 3.3 volt static microprocessor that operates down to 0 MHz.

For specific details on the MC68LC040, refer to **Appendix A MC68LC040**. For specific details on the MC68040V, refer to both **Appendix A MC68LC040** and **Appendix C MC68040V and MC68EC040V**. **Disregard all information concerning the FPU** when reading the following subsections.

1.1.2 MC68EC040 and MC68EC040V

The MC68EC040 and MC68EC040V are derivatives of the MC68040. They implement the same IU as the MC68040, but have no FPU or MMU, which embedded control applications generally do not require. The MC68EC040 is pin compatible with the MC68040. The following differences exist between the MC68EC040, MC68EC040V, and the MC68040:

- The DLE and MDIS pin names have been changed to JS0 and JS1, respectively.
- PTEST and PFLUSH instructions cause an undetermined number of bus cycles; the user should not execute these instructions.
- The access control unit (ACU) replaces the MMU. The MC68EC040 and MC68EC040V ACU has two data and two instruction registers that are called data and instruction transparent translation registers in the MC68040.
- The MC68EC040 and MC68EC040V do not implement the DLE, multiplexed, or output buffer impedance selection modes of operation. They only implement the small output buffer mode of operation. All MC68EC040 and MC68EC040V timing and drive capabilities are equivalent to the MC68040 in small output buffer mode.
- The MC68EC040 and MC68EC040V do not contain an FPU, causing unimplemented floating-point exceptions to occur using a new stack frame format.
- The MC68040V is a 3.3 volt static microprocessor that operates down to 0 MHz.

Refer to **Appendix B MC68EC040** for specific details on the MC68EC040. Refer to **Appendix B MC68EC040** and **Appendix C MC68040V and MC68EC040V** for specific details on the MC68EC040V. **Disregard information concerning the FPU and MMU** when reading the following subsections.

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
BTST	–(bit number of Destination) ø Z;	BTST Dn,<ea> BTST #<data>,<ea>

CAS	CAS Destination – Compare Operand ø cc;
	if Z, Update Operand ø Destination
	else Destination ø Compare Operand
		CAS Dc,Du,<ea>

CAS2	CAS2 Destination 1 – Compare 1 ø cc;
	if Z, Destination 2 – Compare ø cc;
	if Z, Update 1 ø Destination 1;
	Update 2 ø Destination 2
	else Destination 1 ø Compare 1;
	Destination 2 ø Compare 2
		CAS2 Dc1–Dc2,Du1–Du2,(Rn1)–(Rn2)

CHK	If Dn < 0 or Dn > Source then TRAP	CHK <ea>,Dn
CHK2	If Rn < LB or If Rn > UB then TRAP	CHK2 <ea>,Rn
CINV	If supervisor state then invalidate selected cache lines else TRAP	CINVL <caches>, (An) CINVP <caches>, (An) CINVA <caches>
CLR	0 ø Destination	CLR <ea>
CMP	Destination – Source ø cc	CMP <ea>,Dn
CMPA	Destination – Source	CMPA <ea>,An
CMPI	Destination – Immediate Data	CMPI #<data>,<ea>
CMPM	Destination – Source ø cc	CMPM (Ay)+,(Ax)+
CMP2	Compare Rn < LB or Rn > UB and Set Condition Codes	CMP2 <ea>,Rn
CPUSH	If supervisor state then if data cache push selected dirty data cache lines; invalidate selected cache lines else TRAP	CPUSHL <caches>, (An) CPUSHP <caches>, (An) CPUSHA <caches>
DBcc	If condition false then (Dn–1 ø Dn; If Dn ≠ –1 then PC + d_n ø PC)	DBcc Dn,<label>
DIVS, DIVSL	Destination ÷ Source ø Destination	DIVS.W <ea>,Dn 32 ÷ 16 ø 16r:16q DIVS.L <ea>,Dq 32 ÷ 32 ø 32q DIVS.L <ea>,Dr:Dq 64 ÷ 32 ø 32r:32q DIVSL.L <ea>,Dr:Dq 32 ÷ 32 ø 32r:32q
DIVU, DIVUL	Destination ÷ Source ø Destination	DIVU.W <ea>,Dn 32 ÷ 16 ø 16r:16q DIVU.L <ea>,Dq 32 ÷ 32 ø 32q DIVU.L <ea>,Dr:Dq 64 ÷ 32 ø 32r:32q DIVUL.L <ea>,Dr:Dq 32 ÷ 32 ø 32r:32q
EOR	Source ⊕ Destination ø Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination ø Destination	EORI #<data>,<ea>
EORI to CCR	Source ⊕ CCR ø CCR	EORI #<data>,CCR
EORI to SR	If supervisor state then Source ⊕ SR ø SR else TRAP	EORI #<data>,SR

effective address. Also, some instructions access multiple memory operands and initiate fetches for each operand.

The instruction finishes execution in the execute stage. Instructions with write-back operands to memory generate pending write accesses that are passed to the write-back stage. The write occurs to the data memory unit if it is not busy. If the following instruction, which is in the <ea> fetch stage, requires an operand fetch, the write-back stalls in the write-back stage since it is at a lower priority. The write-back can stall indefinitely until either the data memory unit is free or another write is pending from the execution stage.

Figure 2-2 illustrates a write cycle, which begins in the IU pipeline. The IU stores the logical address and data for a write operation in a temporary holding register (WB3). Write operation control passes from the IU to the data memory unit once the data memory unit is idle. When the data memory unit receives the logical address and data from the IU, it stores the logical address and data to a second temporary holding register (WB2). The data memory unit then translates the logical address into a physical address. If the address translation is successful, the data memory unit either stores an address translation in the data cache (write hit) or passes it to the bus controller (write-through with write miss). Once the bus controller is ready to execute the external write operation, it multiplexes the data to the correct data byte lanes and stores the multiplexed data and physical address into a third holding register (WB1). WB1 is used in the actual write operation seen on the address and data buses. **Appendix B MC68EC040** contains details on address translation in the MC68EC040.

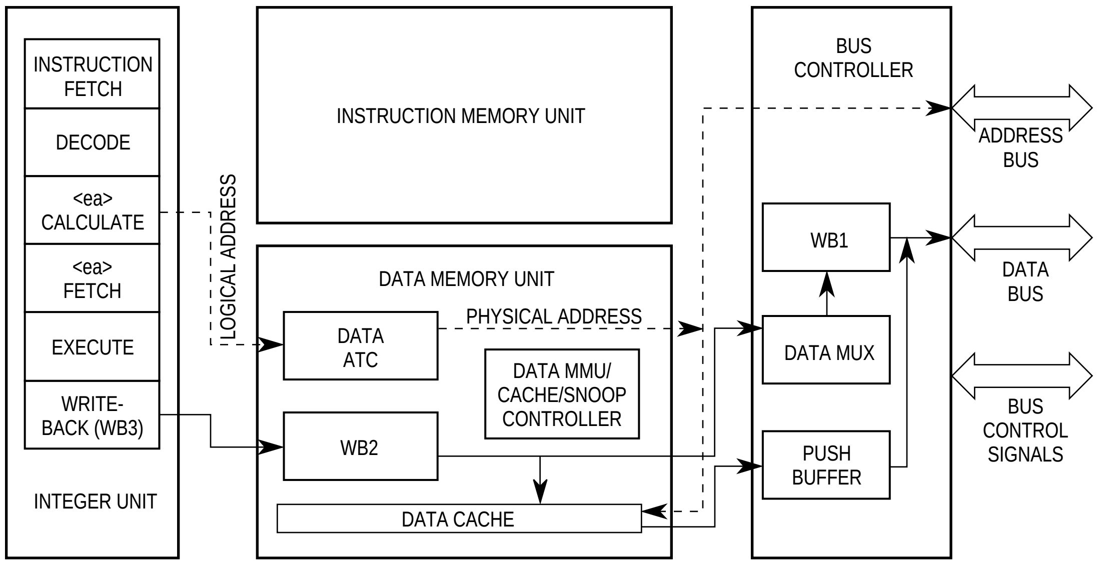


Figure 2-2. Write-Back Cycle Block Diagram

By appropriately configuring a TTR, flexible transparent mappings can be specified (refer to **3.1.3 Transparent Translation Registers** for field identification). For instance, to transparently translate the user address space, the S-field is set to $0, and the logical address mask is set to $FF in both an instruction and data TTR. To transparently translate supervisor accesses of addresses $00000000–$0FFFFFFF with write protection, the logical base address field is set to $0x, the logical address mask is set to $0F, the W-bit is set to one, and the S-field is set to $1. The inclusion of independent TTRs in both the instruction and data MMUs provides an exception to the merged instruction and data address space, allowing different translations for instruction and operand accesses. Also, since the instruction memory unit is only used for instruction prefetches, different instruction and data TTRs can cause PC relative operand fetches to be translated differently from instruction prefetches.

If either of the TTRs matched during an access to a memory unit (either instruction or data), the access is transparently translated. If both registers match, the TT0 status bits are used for the access. Transparent translation can also be implemented by the translation tables of the translation tables if the physical addresses of pages are set equal to their logical addresses.

3.5 ADDRESS TRANSLATION SUMMARY

The instruction and data MMUs process translations by first comparing the logical address and privilege mode with the parameters of the TTRs. If there is a match, the MMU uses the logical address as a physical address for the access. If there is no match, the MMU compares the logical address and privilege mode with the tag portions of the entries in the ATC and uses the corresponding physical address for the access when a match occurs. When neither a TTR nor a valid ATC entry matches, the MMU initiates a table search operation to obtain the corresponding physical address from the translation table. When a table search is required, the processor suspends instruction execution activity and, at the end of a successful table search, stores the address mapping in the appropriate ATC and retries the access. The MMU creates a valid ATC entry for the logical address, and the access is retried. If an access hits in the ATC but an access error or invalid page descriptor was detected during the table search that created the ATC entry, the access is aborted, and a bus error exception is taken.

If a write or read-modify-write access results in an ATC hit but the page is write protected, the access is aborted, and an access error exception is taken. If the page is not write protected and the modified bit of the ATC entry is clear, a table search proceeds to set the modified bit in both the page descriptor in memory and in the ATC; the access is retried. The ATC provides the address translation for the access if the modified bit of the ATC entry is set for a write or read-modify-write access to an unprotected page, if the resident bit is set (indicating the table search for the entry completed successfully), and if none of the TTRs (instruction or data, as appropriate) match.

An ATC access error is not reported immediately, if the last 16 bits of a page is either an A-line, illegal, CHK, or unimplemented instruction and the next page is non-resident. Instead, the M68040 attempts to prefetch the next instruction on the missing page, then the ATC access error exception is reported. The stacked PC points to the exceptional

Table 6-2. Boundary Scan Bit Definitions (Continued)

Bit	Cell Type	Pin/Cell Name	Pin Type	Output Ctrl Cell
74	O.Latch	D21	I/O[2]	io.db
75	O.Latch	D22	I/O[2]	io.db
76	O.Latch	D23	I/O[2]	io.db
77	O.Latch	D24	I/O[2]	io.db
78	O.Latch	D25	I/O[2]	io.db
79	O.Latch	D26	I/O[2]	io.db
80	O.Latch	D27	I/O[2]	io.db
81	O.Latch	D28	I/O[2]	io.db
82	O.Latch	D29	I/O[2]	io.db
83	O.Latch	D30	I/O[2]	io.db
84	O.Latch	D31	I/O[2]	io.db
85	I.Pin	D0	I/O	io.db
86	I.Pin	D1	I/O	io.db
87	I.Pin	D2	I/O	io.db
88	I.Pin	D3	I/O	io.db
89	I.Pin	D4	I/O	io.db
90	I.Pin	D5	I/O	io.db
91	I.Pin	D6	I/O	io.db
92	I.Pin	D7	I/O	io.db
93	I.Pin	D8	I/O	io.db
94	I.Pin	D9	I/O	io.db
95	I.Pin	D10	I/O	io.db
96	I.Pin	D11	I/O	io.db
97	I.Pin	D12	I/O	io.db
98	I.Pin	D13	I/O	io.db
99	I.Pin	D14	I/O	io.db
100	I.Pin	D15	I/O	io.db
101	I.Pin	D16	I/O	io.db
102	I.Pin	D17	I/O	io.db
103	I.Pin	D18	I/O	io.db
104	I.Pin	D19	I/O	io.db
105	I.Pin	D20	I/O	io.db
106	I.Pin	D21	I/O	io.db
107	I.Pin	D22	I/O	io.db
108	I.Pin	D23	I/O	io.db
109	I.Pin	D24	I/O	io.db
110	I.Pin	D25	I/O	io.db
111	I.Pin	D26	I/O	io.db
112	I.Pin	D27	I/O	io.db
113	I.Pin	D28	I/O	io.db
114	I.Pin	D29	I/O	io.db
115	I.Pin	D30	I/O	io.db
116	I.Pin	D31	I/O	io.db
117	O.Latch	A9	I/O[2]	io.ab
118	I.Pin	A9	I/O	io.ab
119	O.Latch	A8	I/O[2]	io.ab
120	I.Pin	A8	I/O	io.ab
121	O.Latch	A7	I/O[2]	io.ab
122	I.Pin	A7	I/O	io.ab
123	O.Latch	A6	I/O[2]	io.ab
124	I.Pin	A6	I/O	io.ab
125	O.Latch	A5	I/O[2]	io.ab
126	I.Pin	A5	I/O	io.ab
127	O.Latch	A4	I/O[2]	io.ab
128	I.Pin	A4	I/O	io.ab
129	O.Latch	A3	I/O[2]	io.ab
130	I.Pin	A3	I/O	io.ab
131	O.Latch	A2	I/O[2]	io.ab
132	I.Pin	A2	I/O	io.ab
133	O.Latch	A1	I/O[2]	io.ab
134	I.Pin	A1	I/O	io.ab
135	O.Latch	A0	I/O[2]	io.ab
136	I.Pin	A0	I/O	io.ab
137	O.Latch	TM2	TS-Output[2]	io.0
138	O.Latch	TM1	TS-Output[2]	io.0
139	O.Latch	TM0	TS-Output[2]	io.0
140	O.Latch	TLN1	TS-Output[2]	io.0
141	O.Latch	TLN0	TS-Output[2]	io.0
142	O.Latch	SIZ0	I/O[2]	io.0
143	I.Pin	SIZ0	I/O	io.0
144	O.Latch	R/W	I/O[2]	io.0
145	I.Pin	R/W	I/O	io.0
146	O.Latch	LOCKE	TS-Output[2]	io.1
147	O.Latch	SIZ1	I/O[2]	io.0

7.4.1 Byte, Word, and Long-Word Read Transfers

During a read transfer, the processor receives data from a memory or peripheral device. Since the data read for a byte, word, or long-word access is not placed in either of the internal caches by definition, the processor ignores the level on the transfer cache inhibit (TCI) signal when latching the data. The bus controller performs byte, word, and long-word read transfers for the following cases:

- Accesses to a disabled cache.
- Accesses to a memory page that is specified noncachable.
- Accesses that are implicitly noncachable (read-modify-write accesses and accesses to an alternate logical address space via the MOVES instruction).
- Accesses that do not allocate in the data cache on a read miss (table searches, exception vector fetches, and exception stack deallocation for an RTE instruction).
- The first transfer of a line read is terminated with transfer burst inhibit (TBI), forcing completion of the line access using three additional long-word read transfers.

Figure 7-8 is a flowchart for byte, word, and long-word read transfers. Bus operations are similar for each case and vary only with the size indicated and the portion of the data bus used for the transfer. Figure 7-9 is a functional timing diagram for byte, word, and long-word read transfers.

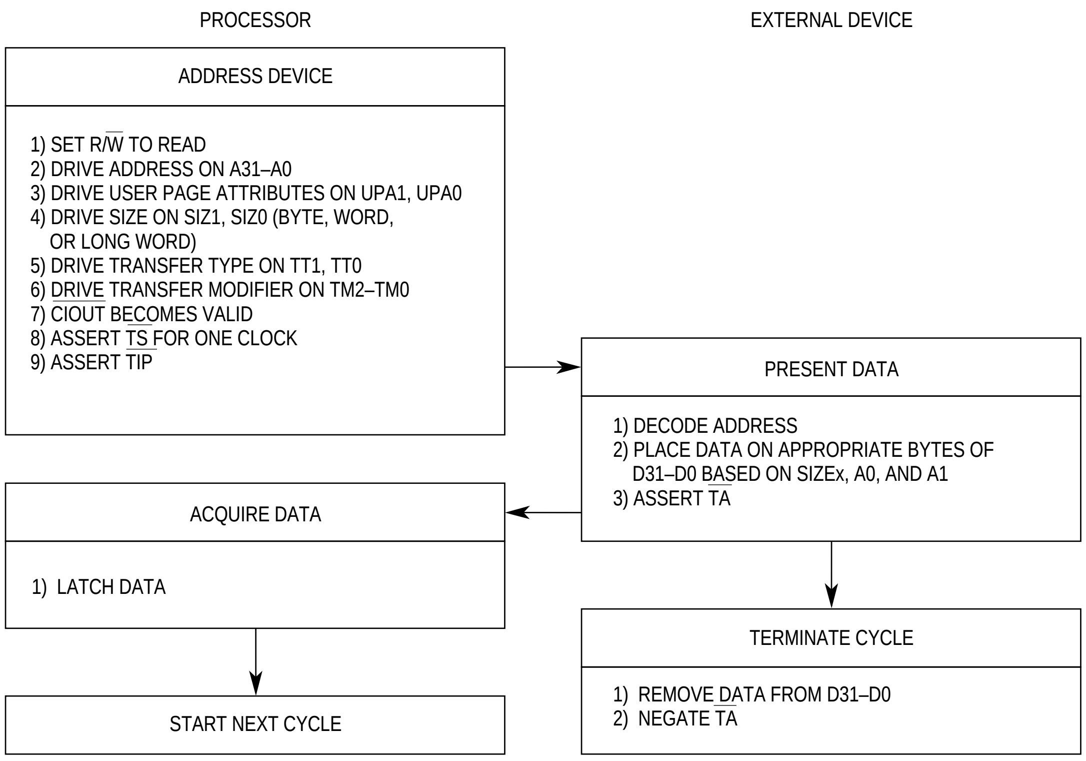

PROCESSOR
EXTERNAL DEVICE
ADDRESS DEVICE
1) SET R/W TO READ
2) DRIVE ADDRESS ON A31–A0
3) DRIVE USER PAGE ATTRIBUTES ON UPA1, UPA0
4) DRIVE SIZE ON SIZ1, SIZ0 (BYTE, WORD, OR LONG WORD)
5) DRIVE TRANSFER TYPE ON TT1, TT0
6) DRIVE TRANSFER MODIFIER ON TM2–TM0
7) CIOUT BECOMES VALID
8) ASSERT TS FOR ONE CLOCK
9) ASSERT TIP
PRESENT DATA
1) DECODE ADDRESS
2) PLACE DATA ON APPROPRIATE BYTES OF D31–D0 BASED ON SIZEx, A0, AND A1
3) ASSERT TA
ACQUIRE DATA
1) LATCH DATA
TERMINATE CYCLE
1) REMOVE DATA FROM D31–D0
2) NEGATE TA
START NEXT CYCLE

Figure 7-8. Byte, Word, and Long-Word Read Transfer Flowchart

(Except MC68EC040 and MC68EC040V) for information on the M68040 and MC68LC040 memory units and **Appendix B MC68EC040** for information on the MC68EC040 memory unit.

The processor asserts $\overline{TS}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, $\overline{TIP}$ is also asserted at this time to indicate that a bus cycle is active.

Clock 2 (C2)

During the first half of the first clock after C1, the processor negates $\overline{TS}$. The selected device uses R/$\overline{W}$, SIZ1, and SIZ0 to place the data on the data bus. (The first transfer must supply the long word at the corresponding long-word boundary.) Concurrently, the selected device asserts $\overline{TA}$ and either negates or asserts $\overline{TBI}$ to indicate it can or cannot support a burst transfer. At the end of the first clock cycle after C1, the processor samples the level of $\overline{TA}$, $\overline{TBI}$, and $\overline{TCI}$ and latches the current value on the data bus. If $\overline{TA}$ is asserted, the transfer terminates and the data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted, the processor ignores the data and inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$, $\overline{TBI}$, and $\overline{TCI}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted. The latched data and the level on $\overline{TCI}$ are then passed to the appropriate memory unit.

If $\overline{TBI}$ was negated with $\overline{TA}$, the processor continues the cycle with C3. Otherwise, if $\overline{TBI}$ was asserted, the line transfer is burst inhibited, and the processor reads the remaining three long words using long-word read bus cycles. The processor increments A3 and A2 for each read, and the new address is placed on the address bus for each bus cycle. Refer to **7.4.1 Byte, Word, and Long-Word Read Transfers** for information on long-word reads. If no wait states are generated, a burst-inhibited line read completes in eight clocks instead of the five required for a burst read.

Clock 3 (C3)

The processor holds the address and transfer attribute signals constant during C3. The selected device must increment A3 and A2 to reference the next long word to transfer, place the data on the data bus, and assert $\overline{TA}$. At the end of C3, the processor samples the level of $\overline{TA}$ and latches the current value on the data bus. If $\overline{TA}$ is asserted, the transfer terminates, and the second long word of data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted at the end of C3, the processor ignores the latched data and inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until it is recognized. The latched data is then passed to the appropriate memory unit.

Clock 4 (C4)

This clock is identical to C3 except that once $\overline{TA}$ is recognized asserted, the latched value corresponds to the third long word of data for the burst.

On the initial cycle of a line transfer, a retry causes the processor to retry the bus cycle as illustrated in Figure 7-29. However, the processor recognizes a retry signaled during the second, third, or fourth cycle of a line as a bus error and causes the processor to abort the line transfer. A burst-inhibited line transfer can only be retried on the initial transfer. A burst-inhibited line transfer aborts if a retry is signaled for any of the three long-word transfers used to complete the line transfer. Negating the bus grant ($\overline{BG}$) signal on the M68040 while asserting both $\overline{TA}$ and $\overline{TEA}$ provides a relinquish and retry operation for any bus cycle that can be retried (see Figure 7-31).

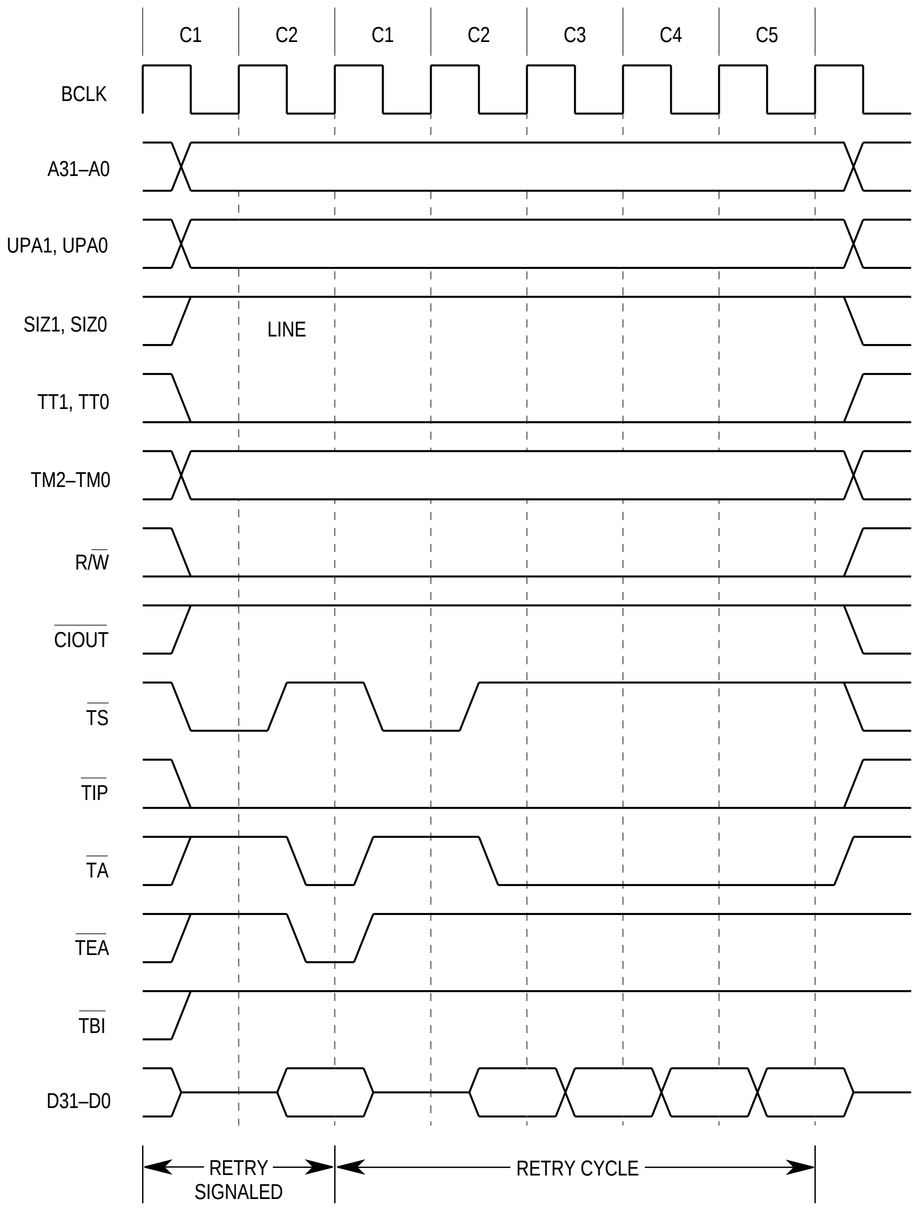


Figure 7-29. Retry Operation on Line Write

7.9 BUS SNOOPING OPERATION

When required, the M68040 can monitor alternate bus master transfers and intervene in the access to maintain cache coherency. The encoding of the SCx signals generated by the alternate bus master for each bus cycle controls the process of bus monitoring and intervention called snooping. Only byte, word, long-word, and line bus transfers can be snooped. Refer to **Section 4 Instruction and Data Caches** for SCx encodings.

When the M68040 recognizes that an alternate bus master has asserted $\overline{TS}$, the processor latches the level on the byte offset, SIZx, TMx, and R/$\overline{W}$ signals during the rising edge of BCLK for which $\overline{TS}$ is first asserted. The processor then evaluates the SCx and TTx signals to determine the type of access (TTx = \$0 or \$1), if it is snoopable, and, if so, how it should be snooped. If snooping is enabled for the access, the processor inhibits memory from responding by continuing to assert the memory inhibit signal ($\overline{MI}$) while checking the internal caches for matching lines. During the snooped bus cycle, the M68040 ignores all $\overline{TA}$ assertions while $\overline{MI}$ is asserted. Unless the data cache contains a dirty line corresponding to the access and the requested snoop operation indicates sink data for a write or source data for a read, $\overline{MI}$ is negated, and memory is allowed to respond and complete the access. Otherwise, the processor continues to intervene in the access by keeping $\overline{MI}$ asserted and responding to the alternate bus master as a slave device. The processor monitors the levels of $\overline{TA}$, $\overline{TEA}$, and $\overline{TBI}$ to detect normal, bus error, retry, and burst-inhibited terminations. Note that for alternate bus master burst-inhibited line transfers, the M68040 snoops each of the four resulting long-word transfers. If snooping is disabled, $\overline{MI}$ is negated, and the M68040 counts the appropriate number of $\overline{TA}$ or $\overline{TEA}$ assertions before proceeding. For example, if the SIZx signals are pulled high, the M68040 requires four $\overline{TA}$ assertions, one $\overline{TEA}$ assertion, or one retry termination before proceeding.

As a bus master, the M68040 can be configured to request snooping operations on a page-by-page basis. The UPAx signals are connected to the SCx inputs of the snooping processors. Appropriately programming the user attribute bits in the corresponding page descriptor selects the required snooping operation for a page. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for details on configuring the caching mode and user attribute bits for each memory page for the M68040 and MC68LC040, and refer to **Appendix B MC68EC040** for the MC68EC040.

In a system with multiple bus masters, the memory unit must wait for each snooping bus master to negate $\overline{MI}$ before responding to an access. A termination signal asserted before the negation of $\overline{MI}$ leads to undefined operation and must be avoided at all costs. Also, if the system contains multiple caching masters, then each master must access shared data using write-through pages that allow writes to the data to be snooped by other masters. The copyback caching mode is typically used for data local to a processor because in a multimaster caching system only one master at a time can access a given page of copyback data. The copyback caching mode also prevents multiple snooping processors from intervening in a specific access.

For processor resets after the initial power-on reset, $\overline{\text{RSTI}}$ should be asserted for at least 10 clock periods. Figure 7-45 illustrates timings associated with a reset when the processor is executing bus cycles. Note that $\overline{\text{BB}}$ and $\overline{\text{TIP}}$ (and $\overline{\text{TA}}$ if driven during a snooped access) are negated before transitioning to a three-state level.

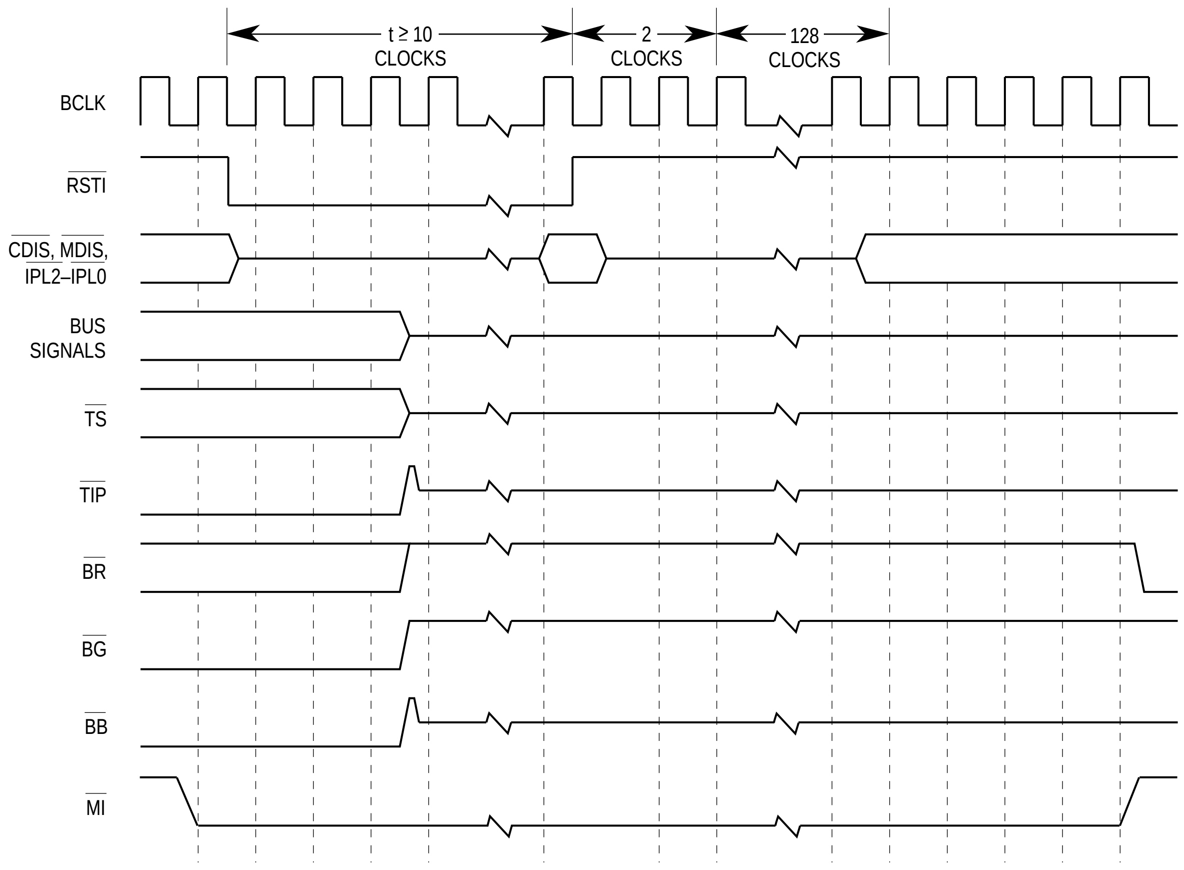


Figure 7-45. Normal Reset Timing

Resetting the processor causes any bus cycle in progress to terminate as if $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ had been asserted. In addition, the processor initializes registers appropriately for a reset exception. **Section 8 Exception Processing** describes exception processing. When a RESET instruction is executed, the processor drives the reset out ($\overline{\text{RSTO}}$) signal for 512 BCLK cycles. In this case, the processor resets the external devices of the system, and the internal registers of the processor are unaffected. The external devices connected to the $\overline{\text{RSTO}}$ signal are reset at the completion of the RESET instruction. An $\overline{\text{RSTI}}$ signal that is asserted to the processor during execution of a RESET instruction immediately resets the processor and causes the $\overline{\text{RSTO}}$ signal to negate. $\overline{\text{RSTO}}$ can be logically ANDed with the external signal driving $\overline{\text{RSTI}}$ to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

Table 9-12. Overflow Rounding Mode Values

Rounding Mode	Result
RN	Infinity, with the sign of the intermediate result.
RZ	Largest magnitude 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.

a. If the user OVFL exception handler is disabled, the M68040FPSP OVFL exception handler checks for an INEX1 or INEX2 exception condition with the user INEX exception handler enabled. If not, the processor returns to normal instruction flow. Otherwise, the M68040FPSP OVFL exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior its execution, and continues instruction execution at the user INEX exception handler. No parameters are passed to the user INEX exception handler since the M68040FPSP OVFL exception handler provides the illusion that it never existed. Otherwise, the M68040FPSP OVFL exception handler returns the processor to normal processing.

b. If the user OVFL exception handler is enabled, the M68040FPSP OVFL restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user OVFL exception handler. No parameters are passed to the user OVFL exception handler since the M68040FPSP OVFL exception handler provides the illusion that it never existed.

The user OVFL exception handler must execute an FSAVE as its first floating-point instruction. The destination contains the rounding mode values listed in Table 9-12, and the user OVFL exception handler can choose to modify these values. The E3 and E1 bits of the floating-point state frame are examined to determine which fields on the floating-point state frame are valid. E3 always takes precedence and must be serviced first. Table 9-16 lists the floating-point state frame fields for OVFL exceptions with E3 set or with E3 clear and E1 set. Note that it is possible for an FADD, FSUB, FMUL, and FDIV to report a post-instruction exception, although these instructions normally generate a pre-instruction exception. The following example illustrates the reason why a post-instruction exception is generated.

```
FADD       FP2,FP0        ; this instruction generates an overflow exception
FMOVE      FP0, <ea>      ; this instruction is executing when overflow occurs
```

In this example, assume that the FMOVE instruction starts once the FADD instruction generates an overflow. Given the register dependency on FP0, the destination of the FADD instruction, FP0 needs to be resolved prior to FMOVE instruction execution. For this example, there is no choice but to have the FADD instruction report a post-instruction exception immediately. Note that for this case, even though the T-bit of the floating-point state frame is set, (post-instruction exception), it does not imply an FMOVE OUT instruction. Therefore, the effective address field in the format $3 stack frame is invalid.

The FMOVE OUT instruction generates a post-instruction exception. For this case, the effective address field in the format $3 stack frame points to the destination memory location. If the destination is an integer data register, the FPIAR points to the F-line word

10.5 MISCELLANEOUS INTEGER UNIT INSTRUCTION TIMINGS

Instruction	Condition	<ea> Calculate	Execute
ABCD	Dy,Dx –(Ay),–(Ax)	1 3	3 1_L + 3
ADDX	Dy,Dx –(Ay),–(Ax)	1 3	1 1_L + 2
ANDI #<xxx>,CCR	—	1	4
ANDI #<xxx>,SR[a]	—	9	1_L + 8
Bcc	Branch Taken Branch Not Taken	2 3	2 3
BRA	Branch Taken Branch Not Taken	2 3	2 3
BSR <offset>	—	2	1_L + 1
CAS2[b]	True False	56 51	6_L + 49 6_L + 44
CMPM	—	3	1_L + 2
DBcc[c]	False, Count > –1 False, Count = –1 True	3 4 4	3 4 4
EORI #<xxx>,CCR	—	1	4
EORI #<xxx>,SR[a]	—	9	1_L + 8
EXG	Dy,Dx Ay,Ax Dy,Ax	1 2 1	1 1_L + 1 1
EXT	Word Long Word	1 1	2 1
EXTB	Long Word	1	1
ILLEGAL[a]	A-Line Unimplemented F-Line Unimplemented	16 16	16 16
LINK	—	3	2_L + 1
MOVE USP	USP,An An,USP[a]	3 7	2_L + 1 1_L + 6
MOVE16[c,d]	(Ax)+,(Ay)+ xxx.L,(An) xxx.L,(An)+ (An),xxx.L (An)+,xxx.L	6 4 5 4 4	1_L+ 7 7 8 7 7
MOVEC[b]	Rn,Rc Rc,Rn	7 11	1_L + 6 1_L + 10
MOVEP[c]	MOVEP.W Dn,d16(An) MOVEP.L Dn,d16(An) MOVEP.W d16(An),Dn MOVEP.L d16(An),Dn	11 13 4 8	2_L + 9 2_L + 11 2_L + 5 2_L + 8
MOVEQ	—	1	1
NOP[a]	—	8	1_L + 7

10.7.2 Integer Unit Support Timings (Continued)

Addressing Mode	FMOVE/FMOVEM to/from 1 Control Register[a]		FMOVEM <list>,<ea> and <ea>,<list>[a,b]		FScc[a]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	2	$1_L + 2$	—	—	5	6
An	2	$1_L + 2$	—	—	—	—
(An)	4	$2_L + 3$	17	$2_L + 15$	4	5
(An)+	4	$2_L + 3$	17	$2_L + 15$	6	$2_L + 5$
–(An)	5	$3_L + 3$	16	$1_L + 15$	6	$2_L + 5$
(xxx).W, (xxx).L	4	$2_L + 3$	19	$3_L + 15$	4	5
#<xxx>	4	$2_L + 3$	19	$1_L + 17$	—	—
$(d_{16}$,An)	4	$2_L + 3$	17	$2_L + 15$	4	5
$(d_{16}$,PC)	5	$4_L + 3$	—	—	—	—
$(d_8$,An,Xn)	5	6	19	18	7	8
$(d_8$,PC,Xn)	6	$1_L + 6$	20	$1_L + 18$	—	—
(An,Xn)	7	$1_L + 7$	20	$1_L + 19$	9	$1_L + 9$
(bd,An,Xn)	8	$1_L + 8$	21	$1_L + 20$	10	$1_L + 10$
([bd,An,Xn])	11	$1_L + 11$	25	$1_L + 23$	13	$1_L + 13$
([bd,An,Xn],od)	12	$1_L + 13$	25	$1_L + 24$	14	$1_L + 14$
([bd,An],Xn)	12	$3_L + 10$	26	$3_L + 22$	14	$3_L + 12$
([bd,An],Xn,od)	13	$3_L + 12$	26	$3_L + 23$	15	$3_L + 13$

NOTES:
a. Timings are for an idle floating-point unit. Same as FMOVE <ea>,FPCR.
b. Add three clocks to both <ea> calculate and execute times for each additional floating-point register. Add one clock to both <ea> calculate and execute times for dynamic register list.

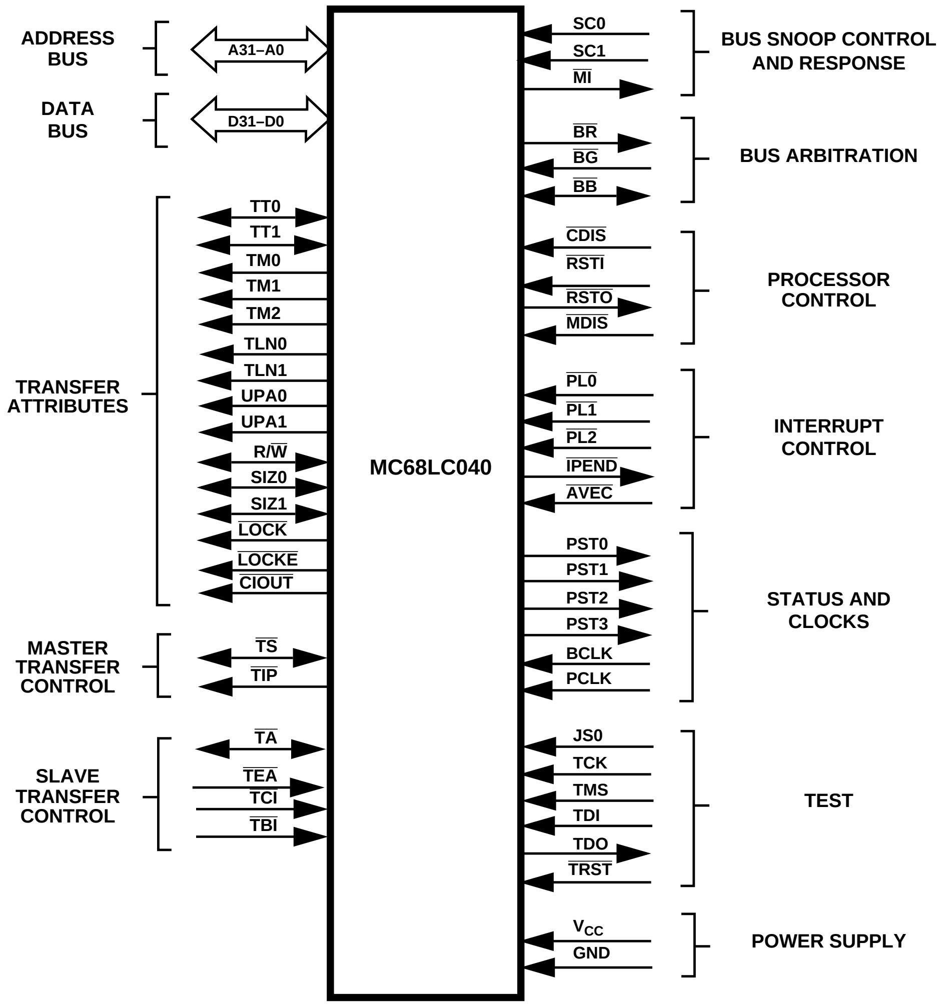


Figure A-3. MC68LC040 Functional Signal Groups

A.1 MC68LC040 DIFFERENCES

The following differences exist between the MC68LC040 and MC68040:

- The MC68LC040 does not implement the small output bufferr impedance selection mode.
- The DLE pin name has been changed to JS0
- The MC68LC040 does not implement the data latch (DLE) or multiplexed bus modes of operation. All timing and drive capabilities of the MC68LC040 are equivalent to those of the MC68040 in small output buffer impedance mode.
- The MC68LC040 does not contain an FPU, which causes unimplemented floating-point exceptions to occur using a new eight-word stack frame format.

A.2 INTERRUPT PRIORITY LEVEL ($\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$)

The $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$ pins do not have any affect on the selection of output buffer impedance.

A.3 JTAG SCAN (JS0)

The MC68040 DLE pin name has been changed to JS0. During normal operation, the JS0 pin cannot float, it must be tied to GND or Vcc directly or through a resistor. During board testing, this pin retains the functionality of the JTAG scan of the MC68040 for compatibility purposes. Refer to **Section 6 IEEE 1149.1A Test Access Port (JTAG)** for details concerning IEEE *1149.1 Standard Test Access Port and Boundary Scan Architecture*.

A.4 DATA LATCH AND MULTIPLEXED BUS MODES

The MC68LC040 does not implement the data latch or multiplexed modes of operation. The $\overline{\text{CDIS}}$ pin is ignored at the rising edge of reset. All timing and drive capabilities of the MC68LC040 are equivalent to those of the MC68040 in small output buffer impedance mode.

A.5 FLOATING-POINT UNIT (FPU)

The FPU is not implemented on the MC68LC040. All floating-point instructions cause an unimplemented floating-point exception to be taken with a new eight-word stack frame (format $4). The stack frame contains the status register (SR), program counter (PC), vector offset, effective address of the operand (where applicable), and PC value of the unimplemented floating-point instruction.

A.5.1 Unimplemented Floating-Point Instructions and Exceptions

All legal MC68040 and MC68881/MC68882 floating-point instructions are defined as unimplemented floating-point instructions on the MC68LC040. These instructions generate a format $4 stack frame during exception processing before taking an F-line exception. These instructions trap as an F-line exception, and the F-line exception handler can emulate them in software to maintain user-object-code compatibility.

The MC68LC040 assists the emulation process by distinguishing unimplemented floating-point instructions from other unimplemented F-line instructions. To aid emulation, the effective address is calculated and saved in the format $4 stack frame. This simplifies and speeds up the emulation process by eliminating the need for the emulation routine to determine the effective address and by providing information required to emulate the instruction on the exception stack frame in the supervisor address space. However, the floating-point instruction can reside in user space; therefore, the floating-point unimplemented exception handler may need to access user instruction space. The following processing steps occur for an unimplemented floating-point instruction:

1. When an unimplemented floating-point instruction is encountered, the instruction is partially decoded, and the effective address is calculated, if required.
2. The processor waits for all previous integer instructions, write-backs, and associated exception processing to complete before beginning exception processing for the unimplemented floating-point instruction. Any access error that occurs in completing the write-backs causes an access error exception, and the resulting stack frame indicates

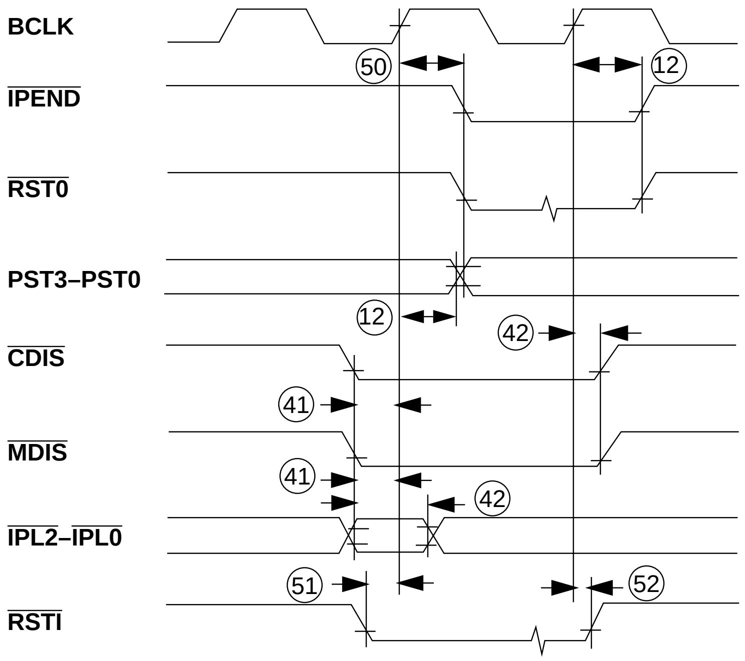


Figure A-9. Other Signal Timing

$\overline{\text{RSTO}}$ are reset at the completion of the RESET instruction. An $\overline{\text{RSTI}}$ signal that is asserted to the processor during execution of a RESET instruction immediately resets the processor and causes $\overline{\text{RSTO}}$ to negate. $\overline{\text{RTSO}}$ can be logically ANDed with the external signal driving $\overline{\text{RTSI}}$ to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

B.5 EXCEPTION PROCESSING

The MC68EC040 provides five different stack frames for exception processing and allows for a MC68040-specific stack frame. Refer to **Section 8 Exception Processing** for details on exception processing.

B.5.1 Unimplemented Floating-Point Instructions and Exceptions

All legal MC68040 and MC68881/MC68882 floating-point instructions are defined as unimplemented floating-point instructions on the MC68EC040. These instructions generate an eight-word stack frame (format $4) during exception processing before taking an F-line exception. These instructions trap as an F-line exception and can be emulated in software by the F-line exception handler to maintain user-object-code compatibility.

The MC68EC040 assists the emulation process by distinguishing unimplemented floating-point instructions from other unimplemented F-line instructions. To aid emulation, the effective address is calculated and saved in the format $4 stack frame. This simplifies and speeds up the emulation process by eliminating the need for the emulation routine to determine the effective address and by providing information required to emulate the instruction on the exception stack frame in the supervisor address space. However, the floating-point instruction can reside in user space; therefore, the floating-point unimplemented exception handler may need to access user instruction space. The following processing steps occur for an unimplemented floating-point instruction:

1. When an unimplemented floating-point instruction is encountered, the instruction is partially decoded, and the effective address is calculated, if required.
2. The processor waits for all previous integer instructions, write-backs, and associated exception processing to complete before beginning exception processing for the unimplemented floating-point instruction. Any access error that occurs in completing the write-backs causes an access error exception, and the resulting stack frame indicates a pending unimplemented floating-point instruction exception. The access error exception handler then completes the write-backs in software, and exception processing for the unimplemented floating-point instruction exception begins immediately after return from the access error handler.
3. The processor begins exception processing for the unimplemented floating-point instruction by making an internal copy of the current SR. The processor then enters the supervisor mode and clears the trace bits (T1 and T0). It creates a format $4 stack frame and saves the internal copy of the SR, PC, vector offset, calculated effective address, and PC value of the faulted instruction in the stack frame.

   The effective address field of the format $4 stack frame contains the calculated effective address of the operand for the faulted floating-point instruction using the addressing mode in which the effective address is calculated. For immediate and register

INDEX