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	Obsolete
Core Processor	68060
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	66MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	3.3V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	206-BPGA
Supplier Device Package	206-PGA (47.25x47.25)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68lc060rc66

tests that set the BSUN bit in the FPSR status byte if an unordered condition is present when the conditional test is attempted (IEEE nonaware tests), and 16 tests that do not cause the BSUN bit in the FPSR status byte (IEEE aware tests). The set of IEEE nonaware tests is best used:

- ¥ When porting a program from a system that does not support the IEEE 754 standard to a conforming system, or
- ¥ When generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition).

An unordered condition occurs when one or both of the operands in a floating-point compare operation is a NAN. The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE). Rather, the opposite condition is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true; whereas, both FBGT and FBLE would be false since unordered fails both of these tests. Compiler programmers should be particularly careful of the lack of trichotomy in the floating-point branches since it is common for compilers to invert the sense of conditions.

When using the IEEE nonaware tests, the BSUN bit and the NAN bit are set in the FPSR, unless the branch is an FBEQ or an FBNE. If the BSUN exception is enabled in the FPCR, an exception is taken. Therefore, the IEEE nonaware program may be interrupted if an unexpected condition occurs.

Compilers and programmers who are knowledgeable of the IEEE 754 standard should use the IEEE aware tests in programs that contain ordered and unordered conditions. Since the ordered or unordered attribute is explicitly included in the conditional test, the BSUN bit is not set in the FPSR EXC byte when the unordered condition occurs.

Table 6-9 summarizes the conditional mnemonics, definitions, equations, predicates, and whether the BSUN bit is set in the FPSR EXC byte for the 32 floating-point conditional tests. The equation column lists the combination of FPCC bits for each test in the form of an equation.

dler is enabled. If the destination is a floating-point data register, the register is not affected, and a pre-instruction exception is reported. If the destination is a memory or integer data register, then an undefined result is stored, and a post-instruction exception is taken immediately. In addition, the processor incorrectly reports an underflow exception if the result of a floating-point multiply is a normalized number with an exponent of $0000. Exception processing begins with the M68060SP UNFL exception handler to provide MC68881-compatible operation. The M68060SP then determines whether or not control is passed back to normal instruction flow (the OVFL bit in the FPCR exception enable byte is cleared), to the user OVFL handler (the OVFL bit in the FPCR exception enable byte is set) or the user INEX handler (the OVFL bit in the FPCR exception enable byte is cleared, but the INEX bit in the FPCR exception enable byte is set and the corresponding INEX bit in the FPSR EXC byte is also set).

6.6.5.1 TRAP DISABLED RESULTS (FPCR UNFL BIT CLEARED). The result that is stored in the destination is either a denormalized number or zero. Denormalization is accomplished by shifting the mantissa of the intermediate result to the right while incrementing the exponent until it is equal to the denormalized exponent value for the destination format. The denormalized intermediate result is rounded to the selected rounding precision or destination format.

If, in the process of denormalizing the intermediate result, all of the significant bits are shifted off to the right, the selected rounding mode determines the value to be stored at the destination, as shown in Table 6-14.

Table 6-14. Underflow Rounding Mode Values

Rounding Mode	Result
RN	Zero, with the sign of the intermediate result.
RZ	Zero, with the sign of the intermediate result.
RM	For positive overflow, + zero; for negative underflow, smallest denormalized negative number.
RP	For positive overflow, smallest denormalized positive number; for negative underflow, Đzero.

6.6.5.2 TRAP ENABLED RESULTS (FPCR UNFL BIT SET). The result stored in the destination is the same as the result stored when traps are disabled. For an FMOVE OUT, the operand is stored in the destination memory or integer data register before control is passed to the user UNFL handler as a post-instruction exception. Otherwise, if the destination is a floating-point data register, control is passed to the user UNFL handler as a pre-instruction exception when the next floating-point instruction is encountered.

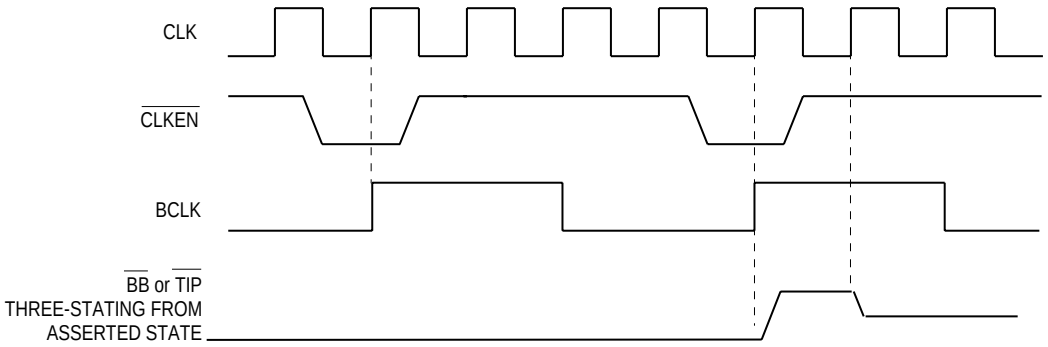


Figure 7-4. Quarter-Speed Clock

7.2 FULL-, HALF-, AND QUARTER-SPEED BUS OPERATION AND BCLK

To simplify the description of full-, half-, and quarter-speed bus operation, the term Òbus clockÓ or ÒBCLKÓ is introduced to describe the effective frequency of bus operation. The bus clock is analogous to the MC68040 clock input called BCLK. The MC68040 BCLK defines when input signals are sampled and when output signals begin to transition. Once the relationship of CLK, CLKEN, and the virtual BCLK is established, it is possible to describe the MC68060 bus more easily, relative to BCLK.

CLKEN allows the bus to synchronize to BCLK which is running at half or quarter speed of the processor clock (CLK). On rising CLK edges in which CLKEN is asserted, inputs to the processor are recognized and outputs of the processor may begin to assert, negate, or three-state. On rising CLK edges in which CLKEN is negated, no inputs are recognized and no outputs begin to change (except BB and TIP). Figure 7-1 illustrates the general relationship between CLK, CLKEN, and most input and output signals.

For brevity, the term Òfull-speed busÓ is introduced to refer to systems in which BCLK is running at the same frequency as CLK. The term Òhalf-speed busÓ refers to systems in which BCLK is running at half the frequency of CLK. For those familiar with the MC68040, the half-speed bus is analogous to the MC68040 implementation. The term Òquarter-speed busÓ refers to systems in which BCLK is running at one quarter the frequency of CLK. The MC68060 clocking mechanism is designed so that systems designed today can be upgraded with higher-frequency MC68060s, without forcing the rest of the system to operate at the same higher processor frequency. This flexibility also allows the MC68060 to be used in existing MC68040 system designs.

A full-speed bus design is achieved by continuously asserting CLKEN as shown in Figure 7-2. A half speed bus is achieved by asserting CLKEN about every other rising edge of CLK. Figure 7-3 shows a timing diagram of the relationship between CLK, CLKEN, and BCLK for half-speed bus operation. A quarter-speed bus is achieved by asserting CLKEN once about every four rising edges of CLK. Figure 7-4 shows a timing diagram of the relationship between CLK, CLKEN, and BCLK for quarter-speed bus operation.

Note that once BCLK has been established, inputs and outputs appear to be synchronized to this virtual BCLK. To simplify the description of MC68060 bus operation, the rising edges

LEÑLock End Bit

0 = Negate external $\overline{\text{LOCKE}}$ signal.
1 = Assert external $\overline{\text{LOCKE}}$ signal.

SLEÑShadow Copy, Lock End Bit

0 = $\overline{\text{LOCKE}}$ asserted at time of exception.
1 = $\overline{\text{LOCKE}}$ negated at time of exception.

The external $\overline{\text{LOCK}}$ signal is asserted starting with the assertion of $\overline{\text{TS}}$ for the bus cycle of the next operand read or write after setting the L-bit in the BUSCR. The external $\overline{\text{LOCKE}}$ signal is asserted starting with the assertion of $\overline{\text{TS}}$ for the bus cycle of the next operand write after setting the LE bit in the BUSCR. Both the $\overline{\text{LOCK}}$ and $\overline{\text{LOCKE}}$ external signals are negated the cycle after the final $\overline{\text{TA}}$ assertion associated with the $\overline{\text{TS}}$ that asserted $\overline{\text{LOCKE}}$. The final operand write cycle must not be misaligned. A final write to the BUSCR must be made in order to clear the L and LE bits even though the external signals have already negated. The L and LE bits are cleared when the processor is reset.

The SL and SLE bits in the BUSCR are provided to retain a copy of the L and LE bits at the time of an exception. When an exception occurs, the MC68060 copies the L and LE bits to the SL and SLE bits respectively, negates the external $\overline{\text{LOCK}}$ and $\overline{\text{LOCKE}}$ pins, and clears the L and LE bits. It is recommended that all interrupts be masked prior to the use of BUSCR. If the cause of the exception is an access error, a bit in the fault status long word (FSLW) in the access error frame is used to signify that a locked sequence was being executed at the time of the fault.

7.5 DATA TRANSFER MECHANISM

Figure 7-6 illustrates how the bus designates operands for transfers on a byte boundary system. The integer unit handles floating-point operands as a sequence of related long-word operands. These designations are used in the figures and descriptions that follow.

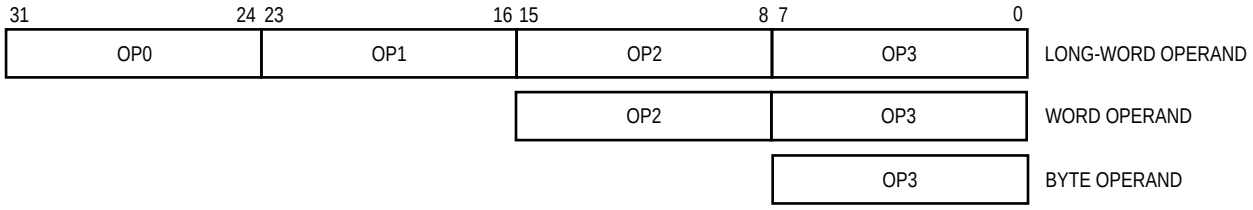


Figure 7-6. Internal Operand Representation

Figure 7-7 illustrates general multiplexing between an internal register and the external bus. The internal register connects to the external data bus through the internal data bus and multiplexer. The data multiplexer establishes the necessary connections for different combinations of address and data sizes.

Unlike the MC68020 and MC68030 processors, the MC68060 does not support dynamic bus sizing and expects the referenced device to accept the requested access width. The MC68150 dynamic bus sizer is designed to allow the 32-bit MC68060 bus to communicate

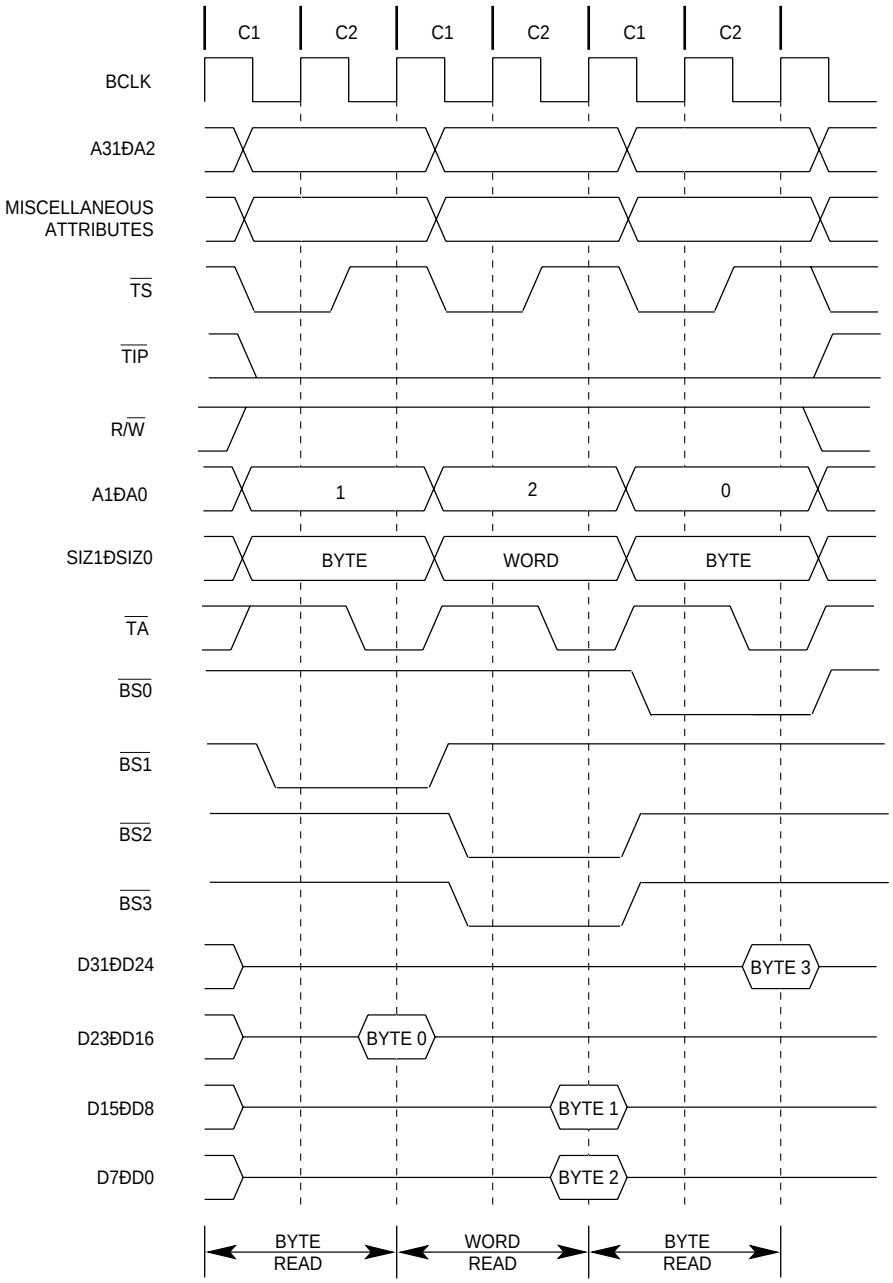
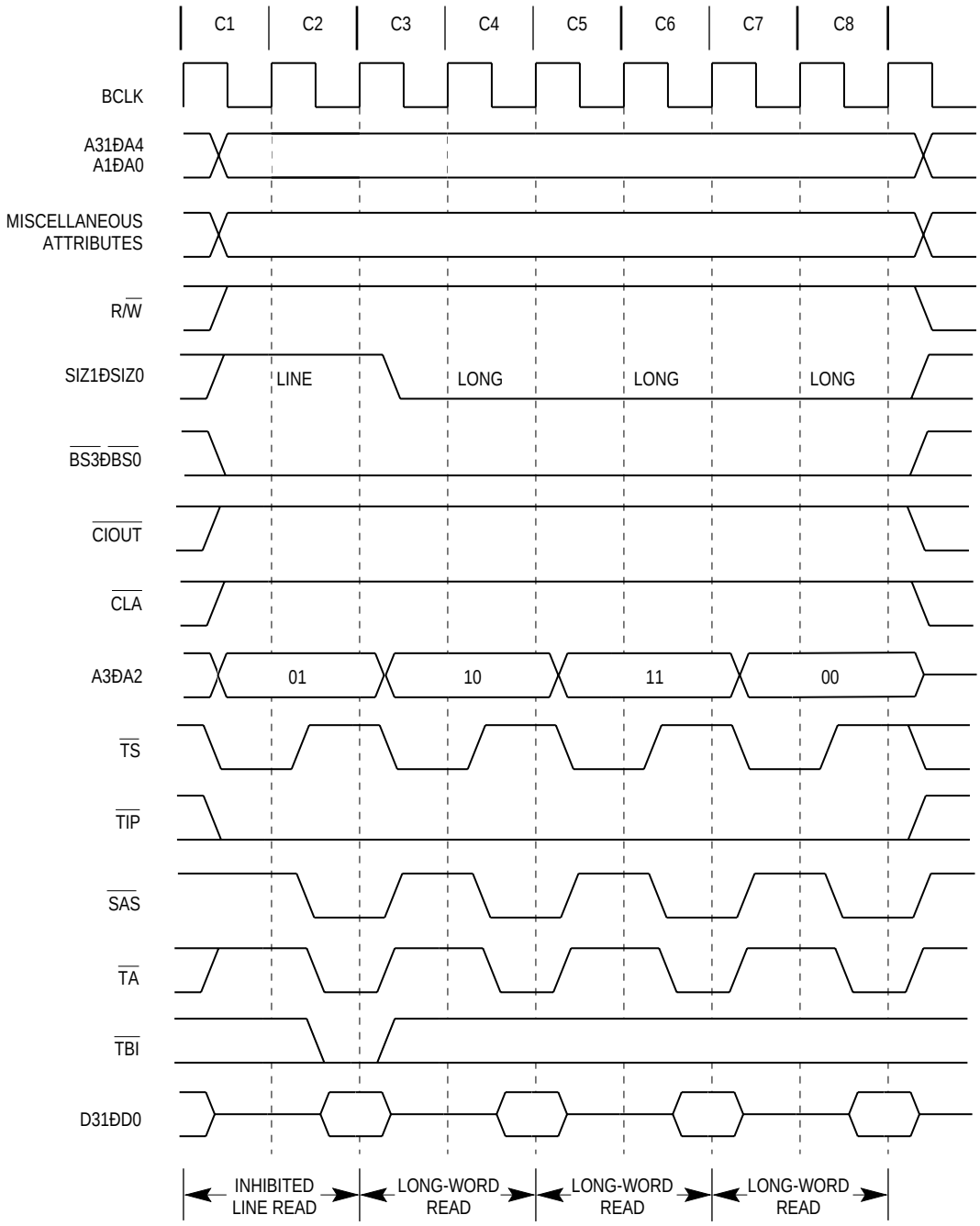


Figure 7-11. Misaligned Long-Word Read Bus Cycle Timing

The combination of operand size and alignment determines the number of bus cycles required to perform a particular memory access. Table 7-3 lists the number of bus cycles required for different operand sizes with all possible alignment conditions for read and write cycles. The table confirms that alignment significantly affects bus cycle throughput for non-cachable accesses. For example, in Figure 7-9 the misaligned long-word operand took three bus cycles because the byte offset = $1. If the byte offset = $0, then it would have taken one



NOTE: It is assumed that the acknowledge termination ignore state capability is disabled.

Figure 7-17. Burst-Inhibited Line Read Bus Cycle Timing

relinquish the bus. But, if the alternate master is another MC68060, it may not be advisable to allow locked sequences to be broken. Figure 7-46 illustrates $\overline{BGR}$ functionality on locked sequences.

When the bus has been granted to the processor in response to the assertion of $\overline{BR}$, one of two situations can occur. In the first situation, the processor monitors $\overline{BB}$ and $\overline{TS}$ to determine when the bus cycle of the alternate bus master is complete and to guarantee that another master has not already started another bus tenure. After the alternate bus master negates and three-states $\overline{BB}$, the processor asserts $\overline{BB}$ to indicate explicit bus ownership and begins the bus cycle by asserting $\overline{TS}$. The processor continues to assert $\overline{BB}$ until the external arbiter negates $\overline{BG}$, after which $\overline{BB}$ is driven negated at the completion of the bus cycle, then forced to a high-impedance state. As long as $\overline{BG}$ is asserted, $\overline{BB}$ remains asserted to indicate the bus is owned, and the processor continuously drives the address bus, attributes, and control signals. The processor negates $\overline{BR}$ when there are no pending internal requests to allow the external arbiter to grant the bus to an alternate bus master if necessary.

In the second situation, the processor samples $\overline{BB}$ until the alternate master negates $\overline{BB}$. Then the processor takes implicit ownership of the bus. Implicit ownership of the bus occurs when the processor is granted the bus, but there are no pending bus cycles. The MC68060 does not drive the bus and $\overline{BB}$ if the bus is implicitly owned. This is different from the MC68040 which drives the address, attributes, and control signals during implicit ownership of the bus. If an internal access request is generated, the processor assumes explicit ownership of the bus and immediately begins an access, simultaneously asserting $\overline{BB}$, $\overline{BR}$, $\overline{TIP}$, and $\overline{TS}$. If the external arbiter keeps $\overline{BG}$ asserted to the processor, the processor keeps $\overline{BB}$ asserted and either executes active bus cycles or drives the address and attributes with undefined values in-between active bus cycles.

$\overline{BR}$ can be used by the external arbiter as an indication that the processor needs the bus. However, there is no guarantee that when the bus is granted to the processor, that a bus cycle will be performed. At best, $\overline{BR}$ must be used as status output that the processor needs the bus, but not as an indication that the processor is in a certain bus arbitration state. Figure 7-41 provides a high-level arbitration diagram that can be used by external arbiters to predict how the MC68060 operates as a function of external signals, and internal signals. For instance, note that the relationship between the internal $\overline{BR}$ and the external $\overline{BR}$ is best described as a synchronous delay off BCLK.

Figure 7-41 is a bus arbitration state diagram for the MC68040 bus arbitration protocol. Table 7-6 lists conditions that cause a change to and from the various states. Table 7-7 lists a summary of the bus conditions and states.

Table 9-3. Boundary Scan Bit Definitions (Continued)

Bit	Cell Type	Pin/Cell Name	Pin Type
130	O.Pin	A5	I/O
131	I.Pin	A5	I/O
132	IO.Ctl	A5ÐA4 ena	Ñ
133	O.Pin	A6	I/O
134	I.Pin	A6	I/O
135	O.Pin	A7	I/O
136	I.Pin	A7	I/O
137	IO.Ctl	A9ÐA6 ena	Ñ
138	O.Pin	A8	I/O
139	I.Pin	A8	I/O
140	O.Pin	A9	I/O
141	I.Pin	A9	I/O
142	O.Pin	D31	I/O
143	I.Pin	D31	I/O
144	O.Pin	D30	I/O
145	I.Pin	D30	I/O
146	IO.Ctl	D31ÐD28 ena	Ñ
147	O.Pin	D29	I/O
148	I.Pin	D29	I/O
149	O.Pin	D28	I/O
150	I.Pin	D28	I/O
151	O.Pin	D27	I/O
152	I.Pin	D27	I/O
153	O.Pin	D26	I/O
154	I.Pin	D26	I/O
155	IO.Ctl	D27ÐD24 ena	Ñ
156	O.Pin	D25	I/O
157	I.Pin	D25	I/O
158	O.Pin	D24	I/O
159	I.Pin	D24	I/O
160	O.Pin	D23	I/O
161	I.Pin	D23	I/O
162	O.Pin	D22	I/O
163	I.Pin	D22	I/O
164	IO.Ctl	D23ÐD20 ena	Ñ
165	O.Pin	D21	I/O
166	I.Pin	D21	I/O
167	O.Pin	D20	I/O
168	I.Pin	D20	I/O
169	O.Pin	D19	I/O
170	I.Pin	D19	I/O
171	O.Pin	D18	I/O
172	I.Pin	D18	I/O
173	IO.Ctl	D19ÐD16 ena	Ñ
174	O.Pin	D17	I/O
175	I.Pin	D17	I/O
176	O.Pin	D16	I/O
177	I.Pin	D16	I/O
178	O.Pin	D15	I/O

```
"191 (BC_2, *,      control,  0),  " &
"192 (BC_1, A(22),  input,    X),  " & -- a[23:20]
"193 (BC_2, A(22),  output3,  X,   191,   0,   Z), " &
"194 (BC_1, A(23),  input,    X),  " &
"195 (BC_2, A(23),  output3,  X,   191,   0,   Z), " &
"196 (BC_1, A(24),  input,    X),  " &
"197 (BC_2, A(24),  output3,  X,   200,   0,   Z), " &
"198 (BC_1, A(25),  input,    X),  " &
"199 (BC_2, A(25),  output3,  X,   200,   0,   Z), " &
--num cell  port    function safe ccell dsval rslt
"200 (BC_2, *,      control,  0),  "   & -- a[27:24]
"201 (BC_1, A(26),  input,    X),  "   &
"202 (BC_2, A(26),  output3,  X,   200,   0,   Z), " &
"203 (BC_1, A(27),  input,    X),  "   &
"204 (BC_2, A(27),  output3,  X,   200,   0,   Z), " &
"205 (BC_1, A(28),  input,    X),  "   &
"206 (BC_2, A(28),  output3,  X,   209,   0,   Z), " &
"207 (BC_1, A(29),  input,    X),  "   &
"208 (BC_2, A(29),  output3,  X,   209,   0,   Z), " &
"209 (BC_2, *,      control,  0),  " & -- a[31:28]
"210 (BC_1, A(30),  input,    X),  " &
"211 (BC_2, A(30),  output3,  X,   209,   0,   Z), " &
"212 (BC_1, A(31),  input,    X),  " &
"213 (BC_2, A(31),  output3,  X,   209,   0,   Z) ";
end MC68060;
```

9.2 DEBUG PIPE CONTROL MODE

A debug pipe control mode is implemented on the MC68060 to allow special chip functions to be accomplished. These functions are useful during system level hardware development and operating system debug. Access to the debug pipe control mode is achieved by negating the $\overline{\text{JTAG}}$ signal. When in the debug pipe control mode, the regular JTAG interface is used by the debug pipe control mode, and is therefore not available.

The debug pipe control mode uses the resulting serial interface to load commands that allow various operations on the processor to occur. Some of the operations are: halt the central processing unit (CPU), restart the CPU, insert select commands into the primary pipeline, disable select processor configurations, force all outputs to high-impedance state, release all outputs from high-impedance state, and generate an emulator interrupt.

The advantage of using the debug pipe control mode is that the processor is allowed to operate normally and at its normal frequency. The only difference is that the processor no longer has the regular JTAG interface. This should not be a problem since the regular JTAG interface is not used during normal processor operations.

Table 10-2. MC68060 Superscalar Classification of M680x0 Integer Instructions (Continued)

Mnemonic	Instruction	Superscalar Classification
BFEXTS	Extract Bit Field Signed	pOEP-only
BFEXTU	Extract Bit Field Unsigned	pOEP-only
BFFFO	Find First One in Bit Field	pOEP-only
BFINS	Insert Bit Field	pOEP-only
BFSET	Set Bit Field	pOEP-only
BFTST	Test Bit Field	pOEP-only
BKPT	Breakpoint	pOEP-only
BRA	Branch Always	pOEP-only
BSET Dy,	Test a Bit and Set	pOEP-only
BSET #<imm>,	Ò	pOEP-until-last
BSR	Branch to Subroutine	pOEP-only
BTST Dy,	Test a Bit	pOEP-only
BTST #<imm>,	Ò	pOEP-until-last
CAS	Compare and Swap with Operand	pOEP-only
CHK	Check Register Against Bounds	pOEP-only
CLR	Clear an Operand	pOEP | sOEP
CMP	Compare	pOEP | sOEP
CMPA	Compare Address	pOEP | sOEP
CMPI,Dx	Compare Immediate	pOEP | sOEP
CMPI,Ð(Ax)+	Ò	pOEP | sOEP
Remaining CMPI	Ò	pOEP-until-last
CMPM	Compare Memory	pOEP-until-last
DBcc	Test Condition, Decrement and Branch	pOEP-only
DIVS.L	Signed Divide Long	pOEP-only
DIVS.W	Signed Divide Word	pOEP-only
DIVU.L	Unsigned Long Divide	pOEP-only
DIVU.W	Unsigned Divide Word	pOEP-only
EOR	Exclusive OR Logical	pOEP | sOEP
EORI,Dx	Exclusive OR Immediate	pOEP | sOEP
EORI,Ð(Ax)+	Ò	pOEP | sOEP
Remaining EORI	Ò	pOEP-until-last
EORI to CCR	Exclusive OR Immediate to Condition Codes	pOEP-only
EXG	Exchange Registers	pOEP-only
EXT	Sign Extend	pOEP | sOEP
EXTB.L	Sign Extend Byte to Long	pOEP | sOEP
ILLEGAL	Take Illegal Instruction Trap	pOEP | sOEP
JMP	Jump	pOEP-only
JSR	Jump to Subroutine	pOEP-only
LEA	Load Effective Address	pOEP | sOEP
LINK	Link and Allocate	pOEP-until-last
LSL	Logical Shift Left	pOEP | sOEP
LSR	Logical Shift Right	pOEP | sOEP
MOVE,Rx	Move Data from Source to Destination	pOEP | sOEP
MOVE Ry,	Ò	pOEP | sOEP
MOVE <mem>y,<mem>x	Ò	pOEP-until-last
MOVE #<imm>,<mem>x	Ò	pOEP-until-last
MOVEA	Move Address	pOEP | sOEP
MOVE from CCR	Move from Condition Codes	pOEP-only

10.9 SHIFT/ROTATE EXECUTION TIMES

Table 10-13 indicates the number of clock cycles required for execution of the shift and rotate instructions. The number of operand read and write cycles is shown in parentheses (r/w). Where indicated, the number of clock cycles and r/w cycles must be added to those required for effective address calculation.

Table 10-13. Shift/Rotate Execution Times

Instruction	Size	Register	Memory [1]
ASL, ASR	Byte, Word	1(0/0)	1(1/1)
Ò	Long	1(0/0)	Ñ
LSL, LSR	Byte, Word	1(0/0)	1(1/1)
Ò	Long	1(0/0)	Ñ
ROL, ROR	Byte, Word	1(0/0)	1(1/1)
Ò	Long	1(0/0)	Ñ
ROXL, ROXR	Byte, Word	1(0/0)	1(1/1)
Ò	Long	1(0/0)	Ñ

[1] For entries in this column, add the effective address calculation time. These operations are word-size only.

10.10 BIT MANIPULATION AND BIT FIELD EXECUTION TIMES

Table 10-14 and Table 10-15 indicate the number of clock cycles required for execution of the bit manipulation instructions. The execution times for the bit field instructions is shown in Table 10-16. The number of operand read and write cycles is shown in parentheses (r/w). Where indicated, the number of clock cycles and r/w cycles must be added to those required for effective address calculation.

Table 10-14. Bit Manipulation (Dynamic Bit Count) Execution Times

Instruction	Size	Register	Memory [1]
BCHG	Byte	Ñ	1(1/1)
Ò	Long	1(0/0)	Ñ
BCLR	Byte	Ñ	1(1/1)
Ò	Long	1(0/0)	Ñ
BSET	Byte	Ñ	1(1/1)
Ò	Long	1(0/0)	Ñ
BTST	Byte	Ñ	1(1/0)
Ò	Long	1(0/0)	Ñ

[1] For entries in this column, add the effective address calculation time.

C.3.2.2.2 System-Supplied Floating-Point Arithmetic Exception Handler Call-Outs. The call-outs _real_bsun, _real_snan, _real_operr, _real_ovfl, _real_unfl, _real_dz, _real_inex are needed only if the system turns on the floating-point exceptions via the floating-point control register (FPCR) exception enable byte. These call-outs point to the arithmetic handlers that must be supplied for IEEE trap enabled operation. Documentation for these handlers are fully explained in Section 6 Floating-Point Unit . Additional information on how these call-outs are reached is found in C.3.2.3 Bypassing Module-Supplied Floating-Point Arithmetic Handlers and C.3.2.4 Exceptions During Emulation .

C.3.2.2.3 Exception-Related Call-Outs. When in the process of emulating any of the floating-point exception handlers, there are conditions that require the M68060SP to emulate an access error, trace, or trap exception. The M68060SP does so by cleaning up the stack to the conditions prior to executing the exception handler, converting the original stack frame to the appropriate stack frame and then branching to those system-supplied exception handlers.

The call-outs _real_access, _real_trace, and _real_trap are defined to provide the system integrator a choice of either having the module point directly to the actual access error, trace and trap exception handlers or to an alternate routine that would calculate the exception handler address from the vector table prior to jumping to actual handlers. The direct implementation is ideal for systems that do not anticipate any changes to the vector table, and for which performance is more critical. The indirect approach of consulting the vector table is more accurate in that if the instruction were implemented, the actual handlerÕs address is fetched from the appropriate vector table entry before branching there.

C.3.2.2.4 Exit Point Call-Outs. The _fpsp_done call-out is provided as a means for the system to do any clean-up, if necessary, before executing the RTE instruction to return to normal instruction execution. All the supplied floating-point handlers will either branch to this call-out or exit through the call-outs _real_fline, _real_fpu_disabled, _real_trace, _real_trap, _real_access, _real_bsun, _real_snan, _real_operr, _real_ovfl, _real_unfl, _real_dz, and _real_inex exit points.

C.3.2.3 BYPASSING MODULE-SUPPLIED FLOATING-POINT ARITHMETIC HANDLERS. A system that does not require full IEEE trap enabled exception compliance or does not require the services of the exceptional operand, may choose to bypass the _fpsp_{ovfl,unfl,snan,operr,dz,inex} entry points. To better assess whether or not to write a customized floating-point arithmetic handler, it is important to know what the processor hardware does and what the M68060SP handlers do individually.

The term ÒopclassÓ is used in the following paragraphs. An opclass zero instruction refers to a floating-point general instruction whose source operand(s) and destination operand are all floating-point data registers (no operands in memory). An opclass two instruction refers to a floating-point general instruction in which one source operand is in memory or an integer data register, but the destination is a floating-point data register. An opclass three instruction refers to an FMOVE instruction that has a memory or integer data register destination.

Table D-2. M68000 Family Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ANDI to CCR	AND Immediate to Condition Code Register
ANDI to SR	AND Immediate to Status Register
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
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BGND	Enter Background Mode
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CALLM	CALL Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CINV	Invalidate Cache Entries
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
cpBcc	Branch on Coprocessor Condition
cpDBcc	Test Coprocessor Condition Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRAPcc	Trap on Coprocessor Condition
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive-OR
EORI	Logical Exclusive-OR Immediate
EORI to CCR	Exclusive-OR Immediate to Condition Code Register
EORI to SR	Exclusive-OR Immediate to Status Register
EXG	Exchange Registers
EXT, EXTB	Sign Extend