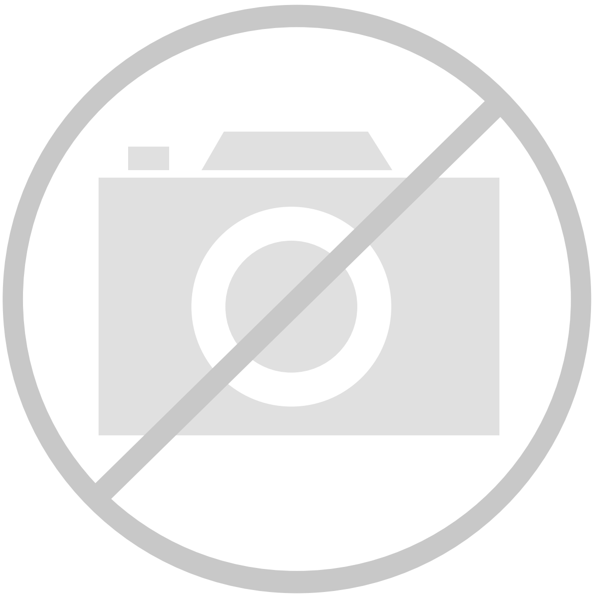
E·XFL

Motorola - MC68EC060RC50 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	68060
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	50MHz
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	206-BPGA
Supplier Device Package	206-PGA (47.25x47.25)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68ec060rc50

Architectural highlights of the MC68060 include:

- Four-Stage Instruction Fetch Unit (IFU)
  - — 64-Entry Instruction Address Translation Cache (ATC), Organized as 4-Way Set-Associative, for Fast Virtual-to-Physical Address Translations
  - — 8- Kbyte, 4-Way Set-Associative, Physically-Mapped Instruction Cache
  - —256-Entry, 4-Way Set-Associative, Virtually-Mapped Branch Cache, Which Predicts the Direction of Branches Based on Their Past Execution History
  - —96-Byte FIFO Instruction Buffer to Allow Decoupling of the IFP and OEPs
- Four-Stage Execution Pipelines Featuring Primary Pipeline (pOEP), Secondary Pipeline (sOEP), and Register File (RGF) Containing Program-Visible General Registers
  - — 64-Entry Operand Data ATC, Organized as 4-Way Set-Associative, for Fast Virtual-to-Physical Address Translations
  - — 8- Kbyte, 4-Way Set-Associative, Physically-Mapped Operand Data Cache
  - — The Operand Data Cache Is Organized in a Banked Structure to Allow Simultaneous Read/Write Accesses
  - — Integer Execute Engines Optimized to Perform Most Instruction Executions in a Single Machine Cycle
  - —Floating-Point Execute Engine, with Floating-Point Register File, Optimized for Performance with Extended-Precision-Wide Internal Datapaths.
  - —Four-Entry Store Buffer and One-Entry Push Buffer That Provide the Performance Feature of Decoupling the Processor Pipeline from External Memory for Certain Cache Modes of Operation.

This pipeline architecture supports extremely high data transfer rates within the MC68060 processor. The on-chip instruction and operand data caches provide 600 MBytes/sec @ 50 MHz to the pipelines, while the integer execute engines can support sustained transfer rates of 1.2 GBytes/sec.

1.4 PROCESSOR OVERVIEW

The following paragraphs provide a general description of the MC68060.

1.4.1 Functional Blocks

Figure 1-1 illustrates a simplified block diagram of the MC68060.

modifying the S-bit of the SR. After these instructions execute, the instruction pipeline is flushed and is refilled from the appropriate address space.

The MC68060 integrates the functions of the integer unit, FPU, and MMU. The registers depicted in the programming model (see Figure 1-2) provide operand storage and control for these three units. The registers are partitioned into two levels of privilege modes: user and supervisor. The user programming model is the same as the user programming model of the MC68040, which consists of 16 general-purpose 32-bit registers, two control registers, eight 80-bit floating-point data registers, a floating-point control register, a floating-point status register, and a floating-point instruction address register.

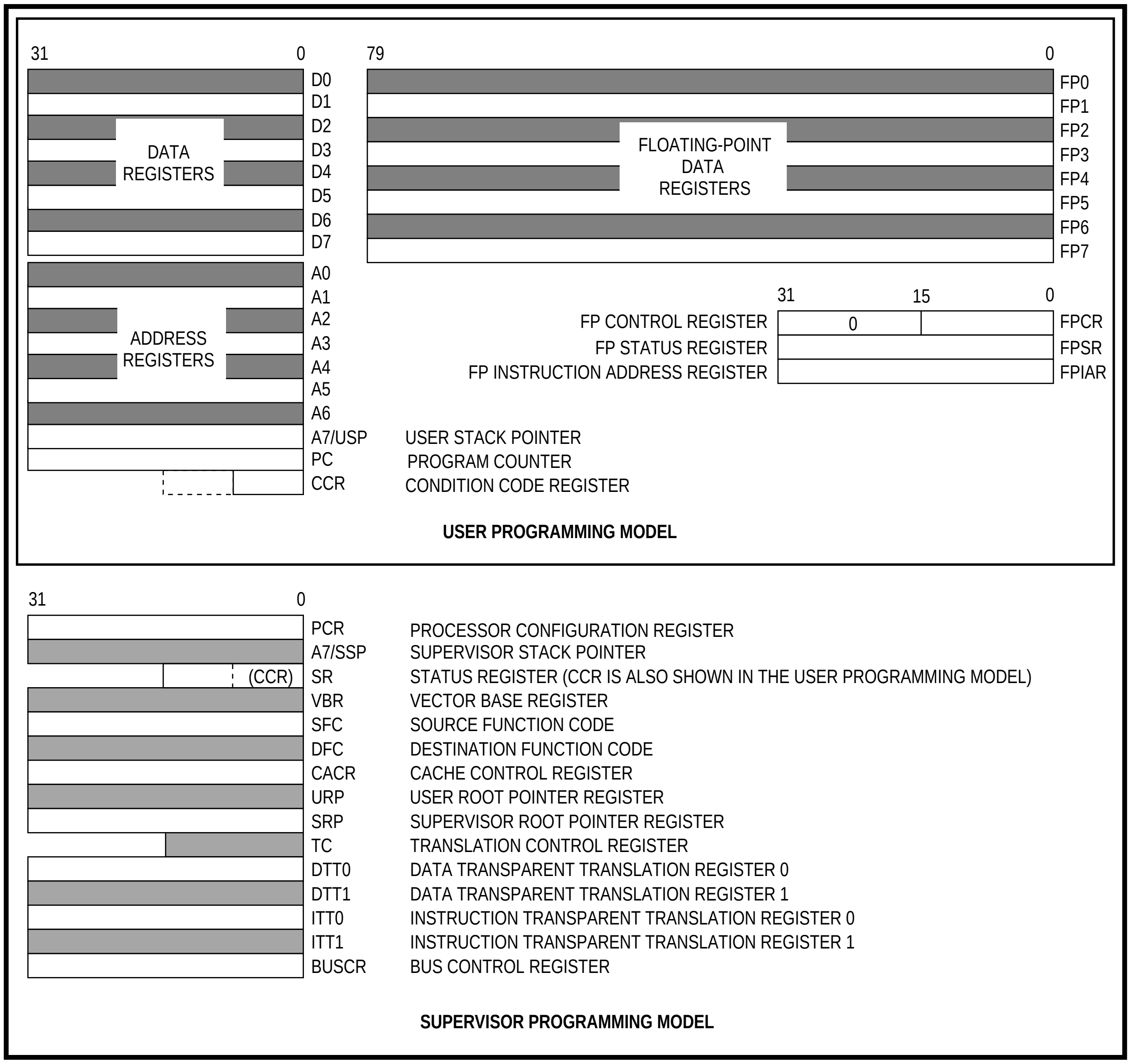

31
0
79
0
D0
D1
D2
D3
D4
D5
D6
D7
DATA REGISTERS
FP0
FP1
FP2
FP3
FP4
FP5
FP6
FP7
FLOATING-POINT DATA REGISTERS
A0
A1
A2
A3
A4
A5
A6
A7/USP
PC
CCR
ADDRESS REGISTERS
31
15
0
FP CONTROL REGISTER
FP STATUS REGISTER
FP INSTRUCTION ADDRESS REGISTER
FPCR
FPSR
FPIAR
USER STACK POINTER
PROGRAM COUNTER
CONDITION CODE REGISTER
USER PROGRAMMING MODEL
31
0
PCR
A7/SSP
SR
(CCR)
VBR
SFC
DFC
CACR
URP
SRP
TC
DTT0
DTT1
ITT0
ITT1
BUSCR
PROCESSOR CONFIGURATION REGISTER
SUPERVISOR STACK POINTER
STATUS REGISTER (CCR IS ALSO SHOWN IN THE USER PROGRAMMING MODEL)
VECTOR BASE REGISTER
SOURCE FUNCTION CODE
DESTINATION FUNCTION CODE
CACHE CONTROL REGISTER
USER ROOT POINTER REGISTER
SUPERVISOR ROOT POINTER REGISTER
TRANSLATION CONTROL REGISTER
DATA TRANSPARENT TRANSLATION REGISTER 0
DATA TRANSPARENT TRANSLATION REGISTER 1
INSTRUCTION TRANSPARENT TRANSLATION REGISTER 0
INSTRUCTION TRANSPARENT TRANSLATION REGISTER 1
BUS CONTROL REGISTER
SUPERVISOR PROGRAMMING MODEL

Figure 1-2. Programming Model

Only system programmers can use the supervisor programming model to implement operating system functions, I/O control, and memory management subsystems. This supervisor/

The MMU replaces an invalid entry when the ATC stores a new address translation. When all entries in an ATC set are valid, the ATC selects a valid entry to be replaced, using a pseudo round robin replacement algorithm. A 2-bit counter, which is incremented for each ATC access, points to the entry to replace when an access misses in the ATC. ATC hit rates are application and page-size dependent, but hit rates ranging from 98% to greater than 99% can be expected. These high rates are achieved because the ATCs are relatively large (64 entries) and utilization efficiency is high with 8-Kbyte and 4-Kbyte page sizes.

4.4 TRANSPARENT TRANSLATION

Four independent TTRs (DTT0 and DTT1 in the data MMU, ITT0 and ITT1 in the instruction MMU) define four blocks of logical address space to be translated to physical address space. These logical address spaces must be at least 16 Mbytes and can overlap or be separate. Each TTR can be disabled and completely ignored. The following description assumes that the TTRs are enabled.

When an MMU receives an address to be translated, the privilege mode and the eight high-order bits of the address are compared to the logical address spaces defined by the two TTRs for the corresponding MMU. The logical address space for each TTR is defined by an S-field, logical base address field, and logical address mask field. The S-field allows matching either user or supervisor accesses or both accesses. When a bit in the logical address mask field is set, the corresponding bit of the logical base address is ignored in the address comparison. Setting successively higher order bits in the address mask increases the size of the physical address space.

The address for the current bus cycle and a TTR address match when the privilege mode and logical base address bits are equal. Each TTR can specify write protection for the block. When write protection is enabled for a block, write or locked read-modify-write accesses to the block are aborted.

By appropriately configuring a TTR, flexible transparent mappings can be specified (refer to **4.1.3 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. It is not necessary for the mask field to specify a contiguous block of memory. 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.

If the paged MMU is disabled (the E-bit in the TCR register is clear) and the TTRs are disabled or do not match, then the status and protection attributes are defined by the default translation bits (DCO, DUO, DWO, DCI, and DUI) in the TCR.

4.5 ADDRESS TRANSLATION SUMMARY

If the paged MMU is enabled (the E-bit in the TCR is set), the instruction and data MMUs process translations by first comparing the logical address and privilege mode with the parameters of the TTRs if they are enabled. 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. If the table search encounters an invalid descriptor, or a write-protect for a write, or is a user access and encounters a supervisor-only flag, then the access error exception is taken whenever the access is needed (immediately for operands and deferred for instruction fetches).

If a write or locked 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 locked read-modify-write access to an unprotected page and if none of the TTRs (instruction or data, as appropriate) match.

Figure 4-21 illustrates a general flowchart for address translation. The top branch of the flowchart applies to transparent translation. The bottom three branches apply to ATC translation.

4.6 $\overline{\text{RSTI}}$ AND $\overline{\text{MDIS}}$ EFFECT ON THE MMU

The following paragraph describes how the MMU is affected by the $\overline{\text{RSTI}}$ and $\overline{\text{MDIS}}$ pins.

4.6.1 Effect of $\overline{\text{RSTI}}$ on the MMUs

When the MC68060 is reset by the assertion of the reset input signal, the E-bits of the TCR and TTRs are cleared, disabling address translation. This reset causes logical addresses to be passed through as physical addresses, allowing an operating system to set up the translation tables and MMU registers as required. After the translation tables and registers are initialized, the E-bit of the TCR can be set, enabling paged address translation. While address translation is disabled, the default TTR is used. The default TTR attribute bits are cleared upon reset, so that immediately after assertion of $\overline{\text{RSTI}}$ the attributes will specify write-through cachable mode, no write protection, user page attribute bits cleared, and 1/2-cache mode disabled.

A reset of the processor does not invalidate any entries in the ATCs page size. A PFLUSH instruction must be executed to flush all existing valid entries from the ATCs after a reset

5.9 STORE BUFFER

The MC68060 processor provides a four-entry store buffer (16 bytes maximum). This store buffer is a FIFO buffer that can be used for deferring pending writes to imprecise pages to maximize performance.

For operand writes destined for the store buffer, the operand execution pipeline incurs no stalls. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (one write per cycle maximum) and the ability of the system bus to retire those writes (one write per two cycles minimum). When writing to imprecise pages, only in the event the store buffer becomes full and there is a write operation in the EX cycle of the operand execution pipeline will a stall be incurred.

If the store buffer is not utilized (store buffer disabled or cache inhibited, precise mode), system bus cycles are generated directly for each pipeline write operation. The instruction is held in the EX cycle of the operand execution pipeline (OEP) until bus transfer termination is received. This means each write operation is stalled for a minimum of five cycles in the EX cycle when the store buffer is not utilized.

A store buffer enable bit is contained in the CACR. This bit can be set and cleared via the MOVEC instruction. Upon reset, this bit is cleared and all writes are precise. When the bit is set, the cache mode generated by the MMU is used. The store buffer is utilized by the cachable/writethrough and the cache-inhibited/imprecise modes.

The store buffer can queue data up to four bytes in width per entry. Each entry matches a corresponding bus cycle it will generate; therefore, a misaligned long-word write to a writethrough page will create two entries if the address is to an odd word boundary, three entries if to an odd byte boundary—one per bus cycle.

A misaligned write access which straddles a precise/imprecise page boundary will use the store buffer for the imprecise portion of the write.

5.10 PUSH BUFFER AND STORE BUFFER BUS OPERATION

Once either the store buffer or the push buffer has valid data, the MC68060 bus controller uses the next available bus cycle to generate the appropriate write cycles. In the event that during the continued instruction execution by the processor pipeline another system bus cycle is required (e.g., data cache miss to process, address translation cache (ATC) tablesearch to perform), the pipeline will stall until both push and store buffers are empty before generating the required system bus transaction.

Certain instructions and exception processing which synchronize the MC68060 processor pipeline guarantee both push and store buffers are empty before proceeding.

5.11 BRANCH CACHE

The branch cache plays a major role in achieving the performance levels of the MC68060 processor. The branch cache provides a table associating branch program counter values with the corresponding branch target virtual addresses. The fundamental concept is to provide a mechanism that allows the instruction fetch pipeline to detect and change instruction streams before the change-of-flow instructions enter an operand execution pipeline.

The branch cache implementation is made up of a five-state prediction model based on past execution history, in addition to the current program counter/branch target virtual address association logic.

For each instruction fetch address generated, the branch cache is examined to see if a valid branch entry is present. If there is not a branch cache hit, the instruction fetch unit continues to fetch instructions sequentially. If a branch cache hit occurs indicating a “taken branch”, the instruction fetch unit discards the current instruction steam and begins fetching at the location indicated by the branch target address. As long as the branch cache prediction is correct, which happens a very significant percentage of the time, the change-of-flow of the instruction stream is “invisible” to the OEP and performance is maximized. If the branch cache prediction is wrong, the internal pipelines are “cancelled” and the correct instruction flow is established.

The branch cache must be cleared by the operating system on all context switches (using the MOVEC to CACR instruction), because it is virtually-mapped.

The branch cache is automatically cleared by the hardware as part of any cache invalidate (CINV) or any cache push and invalidate (CPUSH) instruction operating on the instruction cache.

Programs that use the TRAPF instruction extension word as a possible branch target destination intefere with proper operation of the branch target cache, resulting in an access error exception. This condition is indicated by the BPE bit in the FSLW of the access error stack.

5.12 CACHE OPERATION SUMMARY

The instruction and data caches function independently when servicing access requests from the integer unit. The following paragraphs discuss the operational details for the caches and present state diagrams depicting the cache line state transitions.

5.12.1 Instruction Cache

The integer unit uses the instruction cache to store instruction prefetches as it requests them. Instruction prefetches are normally requested from sequential memory locations except when a change of program flow occurs (e.g., a branch taken) or when an instruction that can modify the status register (SR) is executed, in which case the instruction pipe is automatically flushed and refilled. The instruction cache supports a line-based protocol that allows individual cache lines to be in either the invalid or valid states.

For instruction prefetch requests that hit in the cache, the long word containing the instruction is places onto the internal instruction data bus. When an access misses in the cache, the cache controller requests the line containing the required data from memory and places it in the cache. If available, an invalid line is selected and updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit. If all lines in the set are already valid, a pseudo round-robin replacement algorithm is used to select one

Table 5-3. Data Cache Line State Transitions

Cache Operation	Current State					
	Invalid Cases		Valid Cases		Dirty Cases	
OPU Read Miss	(C,W)I1	Read line from memory and update cache; Supply data to OPU; Go to valid state.	(C,W)V1	Read new line from memory and update cache; supply data to OPU; Remain in current state.	CD1	Push dirty cache line to push buffer; Read new line from memory and update cache; Supply data to OPU; Write push buffer contents to memory; Go to valid state.
OPU Read Hit	(C,W)I2	Not possible.	(C,W)V2	Supply data to OPU; Remain in current state.	CD2	Supply data to OPU; Remain in current state.
OPU Write Miss (Copyback Mode)	CI3	Read line from memory and update cache; Write data to cache; Go to dirty state.	CV3	Read new line from memory and update cache; Write data to cache; Go to dirty state.	CD3	Push dirty cache line to push buffer; Read new line from memory and update cache; Write push buffer contents to memory; Remain in current state.
OPU Write Miss (Writethrough Mode)	WI3	Write data to memory; Remain in current state.	WV3	Write data to memory; Remain in current state.	WD 3	Write data to memory; Remain in current state.
OPU Write Hit (Copyback Mode)	CI4	Not possible.	CV$	Write data to cache; Go to dirty state.	CD4	Write data to cache; Remain in current state.
OPU Write Hit (Writethrough Mode)	WI4	Not possible.	WV4	Write data to memory and to cache; Remain in current state.	WD 4	Push dirty cache line to memory; Write data to memory and to cache; Go to valid state.
Cache Invalidate	(C,W)I5	No action; Remain in current state.	(C,W)V5	No action; Go to invalid state.	CD5	No action (dirty data lost); Go to invalid state.
Cache Push	(C,W)I6	No action; Remain in current state.	(C,W)V6	No action; Go to invalid state.	CD6	Push dirty cache line to memory; Go to invalid state or remain in current state, depending on the DPI bit the the CACR.
Alternate Master Snoop Hit	(C,W)I7	Not possible.	(C,W)V7	No action; Go to invalid state.	CD7	No action (dirty data lost); Go to invalid state.

6.1.3.1 FLOATING-POINT CONDITION CODE BYTE. The FPCC byte (see Figure 6-4) contains four condition code bits that are set at the end of all arithmetic instructions involving the floating-point data registers. These bits are sign of mantissa (N), zero (Z), infinity (I), and NAN. The FMOVE FPm,<ea>, FMOVEM FPm, and FMOVE FPCR instructions do not affect the FPCC.

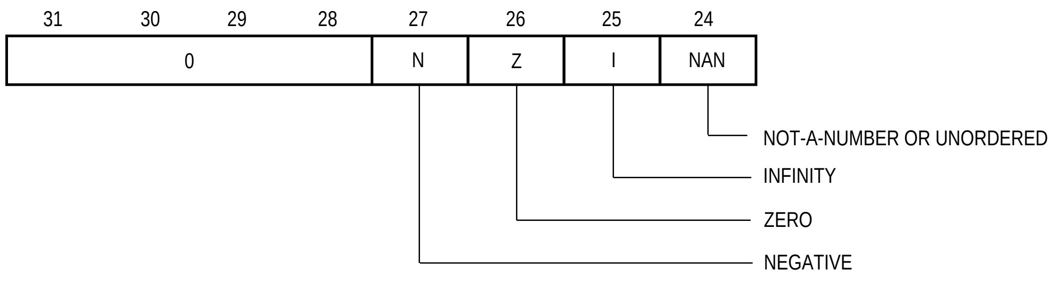


Figure 6-4. Floating-Point Condition Code (FPSR)

To aid programmers of floating-point subroutine libraries, the MC68060 implements the four FPCC bits in hardware instead of only implementing the four IEEE conditions. An instruction derives the IEEE conditions when needed. For example, the programmers of a complex arithmetic multiply subroutine usually prefer to handle special data types, such as zeros, infinities, or NANs, separately from normal data types. The floating-point condition codes allow users to efficiently detect and handle these special values.

6.1.3.2 QUOTIENT BYTE. The quotient byte (see Figure 6-5) provides compatibility with the MC68881/MC68882. This byte is set at the completion of the modulo (FMOD) or IEEE remainder (FREM) instruction, and contains the seven least significant bits of the unsigned quotient as well as the sign of the entire quotient.

The quotient bits can be used in argument reduction for transcendentals and other functions. For example, seven bits are more than enough to determine the quadrant of a circle in which an operand resides. The quotient field (bits 22–16) remains set until the user clears it.

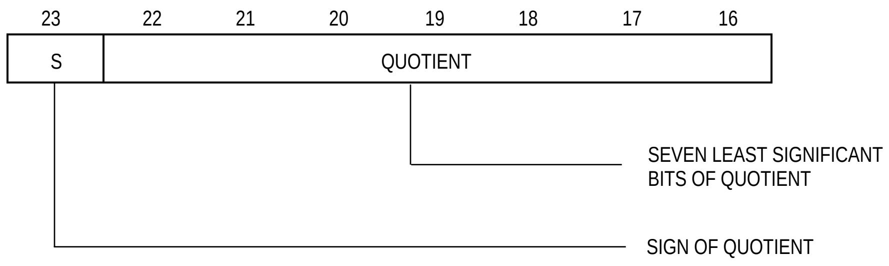
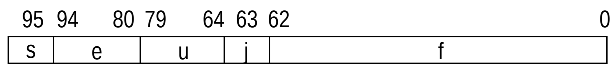


Figure 6-5. Floating-Point Quotient Byte (FPSR)

6.1.3.3 EXCEPTION STATUS BYTE. The EXC byte (see Figure 6-6) contains a bit for each floating-point exception that can occur during the most recent arithmetic instruction or move operation. The start of most operations clears this byte; however, operations that cannot generate floating-point exceptions (the FMOVEM and FMOVE control register instructions) do not clear this byte. An exception handler can use this byte to determine which floating-point exception(s) caused a trap.

Table 6-6. Extended-Precision Real Format Summary

Data Format	
 95 94 80 79 64 63 62 0 s e u j f	
Field Size (in Bits)	
Sign (s)	1
Biased Exponent (e)	15
Zero, Reserved (u)	16
Explicit Integer Bit (j)	1
Mantissa (f)	63
Total	96
Interpretation of Unused Bits	
Input	Don't Care
Output	All Zeros
Interpretation of Sign	
Positive Mantissa	s = 0
Negative Mantissa	s = 1
Normalized Numbers	
Bias of Biased Exponent	+16383 ($3FFF)
Range of Biased Exponent	0 <= e < 32767 ($7FFF)
Explicit Integer Bit	1
Range of Mantissa	Zero or Nonzero
Mantissa (Explicit Integer Bit and Fraction)	1.f
Relation to Representation of Real Numbers	$(-1)^S \times 2^{e-16383} \times j.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($0000)
Bias of Biased Exponent	+16383 ($3FFF)
Explicit Integer Bit	0
Range of Mantissa	Nonzero
Mantissa (Explicit Integer Bit and Fraction)	0.f
Relation to Representation of Real Numbers	$(-1)^S \times 2^{-16383} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($0000)
Mantissa (Explicit Integer Bit and Fraction)	0.0
Signed Infinities	
Biased Exponent Format Maximum	32767 ($7FFF)
Explicit Integer Bit	Don't Care
Mantissa (Explicit Integer Bit and Fraction)	x.000…0000
NANs	
Sign	Don't Care
Explicit Integer Bit	Don't Care
Biased Exponent Format Maximum	32767 ($7FFF)
Mantissa	Nonzero
Representation of Mantissa Nonsignaling Signaling Nonzero Bit Pattern Created by User Mantissa When Created by FPU	 x.1xxxx…xxxx x.0xxxx…xxxx x.xxxxx…xxxx 1.11111…1111

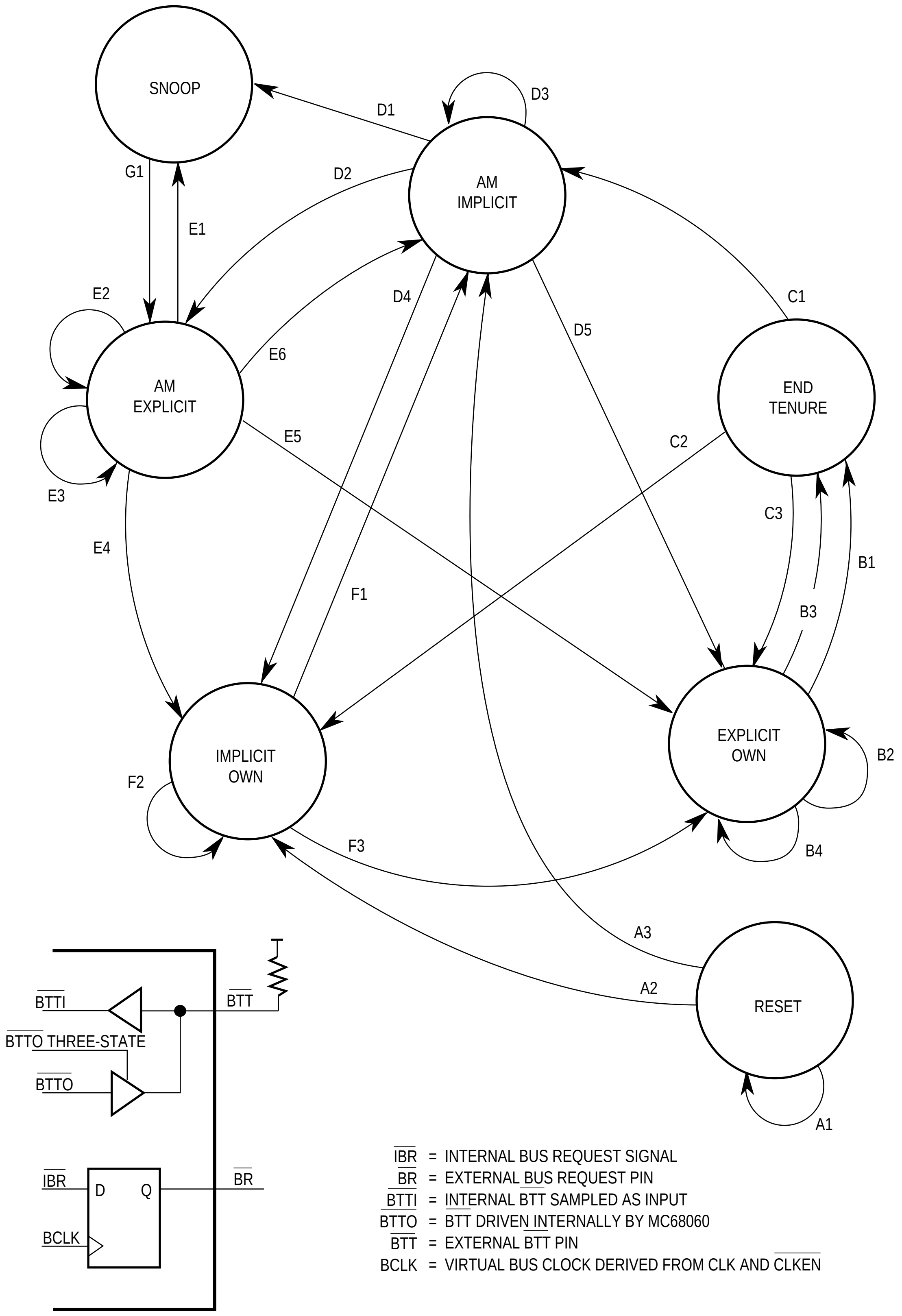
When the processor recognizes a bus error condition for an access, the access is terminated immediately. A line access that has $\overline{\text{TEA}}$ asserted for one of the four long-word transfers aborts without completing the remaining transfers, regardless of whether the line transfer uses a burst or burst-inhibited access.

When a bus cycle is terminated with a bus error, the MC68060 can enter access error exception processing immediately following the bus cycle, or it can defer processing the exception. The instruction prefetch mechanism requests instruction words from the instruction memory unit before it is ready to execute them. If a bus error occurs on an instruction fetch, the processor does not take the exception until it attempts to use the instruction. Should an intervening instruction cause a branch or should a task switch occur, the access error exception for the unused access does not occur. Similarly, if a bus error is detected on the second, third, or fourth long-word transfer for a line read access, an access error exception is taken only if the execution unit is specifically requesting that long word. The line is not placed in the cache, and the processor repeats the line access when another access references the line. If a misaligned operand spans two long words in a line, a bus error on either the first or second transfer for the line causes exception processing to begin immediately. A bus error termination for any write access or read access that reference data specifically requested by the execution unit causes the processor to begin exception processing immediately. Refer to **Section 8 Exception Processing** for details of access error exception processing.

When a bus error terminates an access, the contents of the corresponding cache can be affected in different ways, depending on the type of access. For a cache line read to replace a valid instruction or data cache line, the cache line is untouched if the replacement line read terminates with a bus error. If a dirty data cache line is being replaced, the dirty line is placed in the push buffer and is eventually written out to memory. This is done whether or not a bus error occurs during the replacement line read. If any cache push results in a bus error termination, the cache push data is lost.

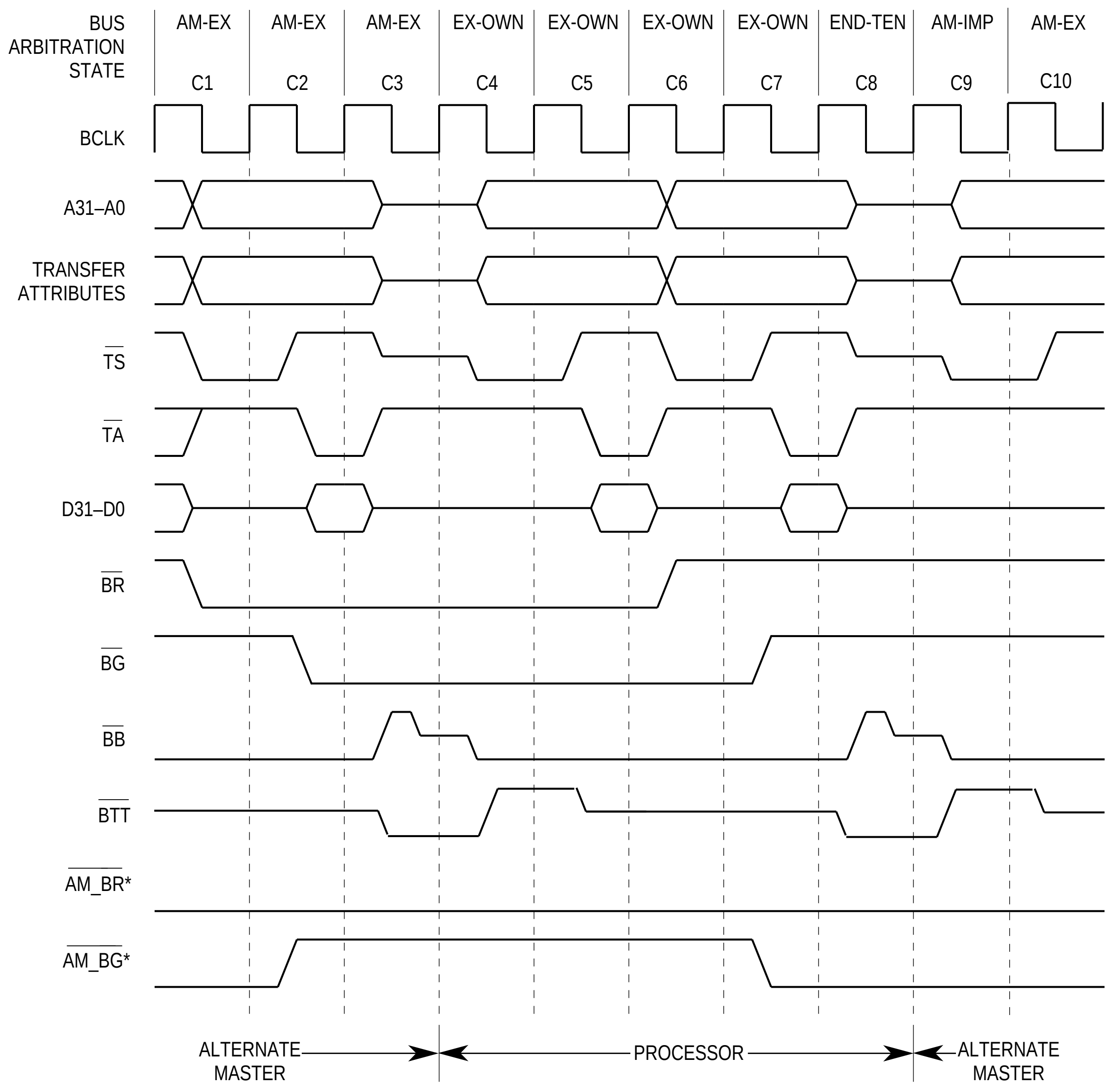
Write accesses to memory pages specified as cachable writethrough by the data memory unit update the corresponding cache line before accessing memory. If a bus error occurs during a memory access, the cache line remains valid with the new data. For noncachable precise memory pages, the cache line is not updated if the write cycle terminates with a bus error. Figure 7-37 illustrates a functional timing diagram of a bus error on a word write access causing an access error exception. Figure 7-38 illustrates a functional timing diagram of a bus error on a line read access that does not cause an access error exception.

In general, write cycles that result in bus error termination must be avoided. The MC68060 has write and push buffers to decouple the processor from the system. Before the processor writes into the write and push buffers, access errors that result from address translation cache (ATC) faults should have been reported via an access error exception and eventually fixed by the access error handler. Since the instruction that reports the bus error on the write cycle usually is not the instruction that causes the write, it is not possible to recover that write cycle via an instruction restart. Although the fault address indicates the logical address of the write cycle that incurred the bus error, the write data information is not available in the access error stack. As such, this access error case is nonrecoverable unless the system is



$\overline{IBR}$ = INTERNAL BUS REQUEST SIGNAL
$\overline{BR}$ = EXTERNAL $\overline{BUS}$ REQUEST PIN
$\overline{BTTI}$ = INTERNAL $\overline{BTT}$ SAMPLED AS INPUT
BTTO = $\overline{BTT}$ DRIVEN INTERNALLY BY MC68060
$\overline{BTT}$ = EXTERNAL $\overline{BTT}$ PIN
BCLK = VIRTUAL BUS CLOCK DERIVED FROM CLK AND $\overline{CLKEN}$

Figure 7-42. MC68060-Arbitration Protocol State Diagram



*AM indicates the alternate bus master.

Figure 7-43. Processor Bus Request Timing

Figure 7-44 illustrates a functional timing diagram for an arbitration of a relinquish and retry operation (MC68040 acknowledge termination mode). In Figure 7-44, the processor read access that begins in C1 is terminated at the end of C2 with a retry request and $\overline{BG}$ negated, forcing the processor to relinquish the bus and allow the alternate master to access the bus. Note that the processor re-asserts $\overline{BR}$ during C3 since the original access is pending again. After alternate bus master ownership, the bus is granted to the processor to allow it to retry the access beginning in C7.

Figure 7-45 is a functional timing diagram for implicit ownership of the bus.

Figure 7-46 illustrates the effect of $\overline{BGR}$ on bus arbitration activity during locked sequences. When $\overline{BGR}$ is asserted while $\overline{BG}$ is negated, locked sequences can be broken. Otherwise, the entire locked sequence of bus cycles are completed by the processor before relinquishing the bus.

9.1.2.2 LPSAMPLE. The LPSAMPLE instruction provides identical functionality to the SAMPLE/PRELOAD instruction described in 9.1.2.4 SAMPLE/PRELOAD with one exception: instead of sampling the system data and control signals present at the MC68060 input pins, the LPSAMPLE instruction forces the LPSTOP isolation transistors into isolation state so that it can be verified that they are present and interrupting the path from the signal pin to the internal logic.The LPSAMPLE instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the instruction shift register is equivalent to a $1.

9.1.2.3 Private Instructions. The set of private instructions labeled MFG-TEST1 through MFG-TEST9 are reserved by Motorola for internal manufacturing use. These instructions can change (remap) the pin I/O and pin functions as defined for system operation (some input pins may become output pins and some output pins may become input pins). Use of these instructions without proper understanding can result in potentially destructive operation of the MC68060. These instructions become active on the falling edge of TCK in the update-IR state when the data held in the instructions shift register is equivalent to values $2, $3, $8, $9, $A, $B, $C, $D, and $E.

9.1.2.4 SAMPLE/PRELOAD. The SAMPLE/PRELOAD instruction provides two separate functions. First, it provides a means to obtain a sample of the system data and control signals present at the MC68060 input pins and just prior to the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when an instruction encoding of $4 is resident in the instruction register. The user can observe this sampled data by shifting it through the boundary scan register to the output TDO by using the shift-DR state. Both the data capture and the shift operation are transparent to system operation. The user is responsible for providing some form of external synchronization to achieve meaningful results since there is no internal synchronization between TCK and the system clock, CLK.

The second function of the SAMPLE/PRELOAD instruction is to initialize the boundary scan register update cells before selecting EXTEST or CLAMP. This is accomplished by ignoring the data being shifted out of the TDO pin while shifting in initialization data. The update-DR state in conjunction with the falling edge of TCK can then be used to transfer this data to the update cells. This data will be applied to the external output pins when one of the instructions listed previously is applied.

9.1.2.5 IDCODE. The IDCODE instruction selects the 32-bit idcode register for connection as a shift path between the TDI pin and the TDO pin. This instruction allows the user to interrogate the MC68060 to determine its JTAG version number and other part identification data. The idcode register has been implemented in accordance with IEEE 1149.1 so that the least significant bit of the shift register stage is set to logic one on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the idcode register is always a logic one (this is to differentiate a part that supports an idcode register from a part that supports only the bypass register). The remaining 31-bits are also set to fixed values (see **9.1.3.1 Idcode Register**) on the rising edge of TCK following entry into the capture-DR state.

The IDCODE instruction is the default value placed in the instruction register when a JTAG reset is accomplished by, either asserting $\overline{\text{TRST}}$, or holding TMS high while clocking TCK

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

Bit	Cell Type	Pin/Cell Name	Pin Type
81	I.Pin	XTEA	Input
82	I.Pin	XTA	Input
83	O.Pin	PST0	Output
84	O.Pin	PST1	Output
85	O.Pin	PST2	Output
86	IO.Ctl	PST4–PST0, XBR ena	—
87	O.Pin	PST3	Output
88	O.Pin	PST4	Output
89	O.Pin	XSAS	Output
90	IO.Ctl	XSAS ena	—
91	O.Pin	XBTT	I/O
92	IO.Ctl	XBTT ena	—
93	I.Pin	XBTT	I/O
94	O.Pin	XTS	I/O
95	I.Pin	XTS ena	—
96	I.Pin	XTS	I/O
97	O.Pin	XTIP	Output
98	IO.Ctl	XTIP ena	—
99	I.Pin	XSNOOP	Input
100	O.Pin	XBB	I/O
101	IO.Ctl	XBB ena	—
102	I.Pin	XBB	I/O
103	O.Pin	XBR	Output
104	IO.Ctl	XLOCK, XLOCKE ena	—
105	O.Pin	XLOCK	Output
106	O.Pin	XLOCKE	Output
107	O.Pin	TLN0	Output
108	O.Pin	SIZ0	Output
109	IO.Ctl	TLN0,SIZ1–SIZ0,XR_W ena	—
110	O.Pin	SIZ1	Output
111	O.Pin	XR_W	Output
112	O.Pin	TLN1	Output
113	O.Pin	TM0	Output
114	IO.Ctl	TLN1,TM2–TM0 ena	—
115	O.Pin	TM1	Output
116	O.Pin	TM2	Output
117	O.Pin	A0	I/O
118	I.Pin	A0	I/O
119	O.Pin	A1	I/O
120	I.Pin	A1	I/O
121	IO.Ctl	A1–A0 ena	—
122	I.Pin	XCLA	—
123	O.Pin	A2	I/O
124	I.Pin	A2	I/O
125	O.Pin	A3	I/O
126	I.Pin	A3	I/O
127	IO.Ctl	A3–A2 ena	—
128	O.Pin	A4	I/O
129	I.Pin	A4	I/O

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

Bit	Cell Type	Pin/Cell Name	Pin Type
179	I.Pin	D15	I/O
180	O.Pin	D14	I/O
181	I.Pin	D14	I/O
182	IO.Ctl	D15–D12 ena	—
183	O.Pin	D13	I/O
184	I.Pin	D13	I/O
185	O.Pin	D12	I/O
186	I.Pin	D12	I/O
187	O.Pin	D11	I/O
188	I.Pin	D11	I/O
189	O.Pin	D10	I/O
190	I.Pin	D10	I/O
191	IO.Ctl	D11–D8 ena	—
192	O.Pin	D9	I/O
193	I.Pin	D9	I/O
194	O.Pin	D8	I/O
195	I.Pin	D8	I/O
196	O.Pin	D7	I/O
197	I.Pin	D7	I/O
198	O.Pin	D6	I/O
199	I.Pin	D6	I/O
200	IO.Ctl	D7–D4 ena	—
201	O.Pin	D5	I/O
202	I.Pin	D5	I/O
203	O.Pin	D4	I/O
204	I.Pin	D4	I/O
205	O.Pin	D3	I/O
206	I.Pin	D3	I/O
207	O.Pin	D2	I/O
208	I.Pin	D2	I/O
209	IO.Ctl	D3–D0 ena	—
210	O.Pin	D1	I/O
211	I.Pin	D1	I/O
212	O.Pin	D0	I/O
213	I.Pin	D0	I/O

9.2.1 Debug Command Interface

Figure 9-10 illustrates the debug command interface and Table 9-4 outlines the pins needed by the debug command interface. The debug command interface consists of a five-bit shift register and a five-bit parallel register, with each register operating independently. To activate the debug command interface, $\overline{\text{JTAG}}$ must be driven negated. This allows the debug command interface to take over the regular JTAG interface and remap $\overline{\text{JTAG}}$ pin functions. The resulting interface is fully synchronous to the CLK input.

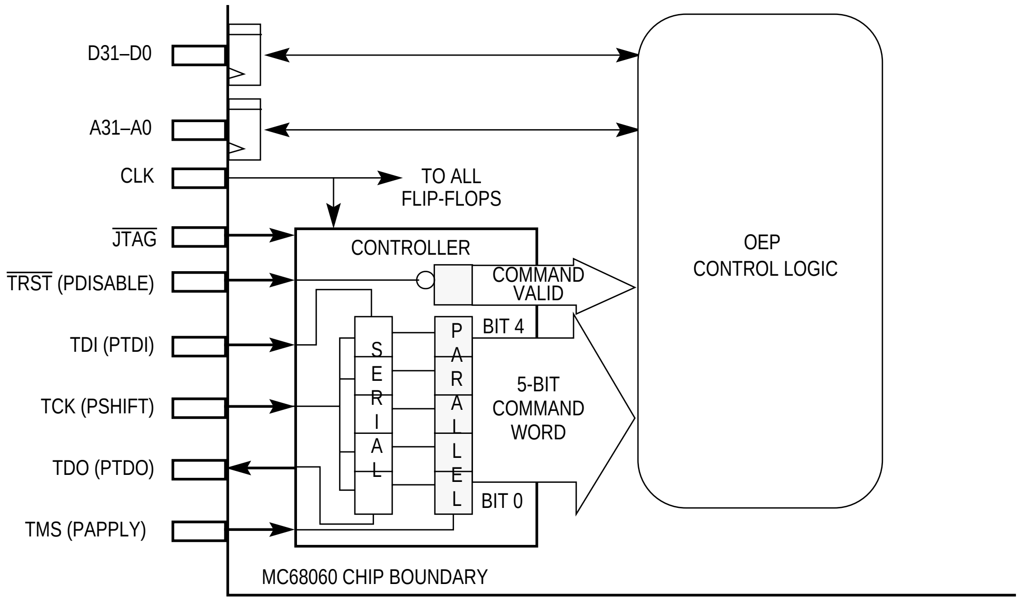


Figure 9-10. Debug Command Interface Schematic

Table 9-4. Debug Command Interface Pins

Pin Name	Alias	Description
TCK	PSHIFT	Serial Shift Enable
TMS	PAPPLY	Command Apply Enable
TDI	PTDI	Serial Command Data In
$\overline{\text{TRST}}$	PDISABLE	Debug Command Disable
TDO	PTDO	Serial Command Data Out
$\overline{\text{JTAG}}$	$\overline{\text{JTAG}}$	JTAG or Debug Select
CLK	CLK	Clock

The commands enter the debug command interface through the PTDI serial input signal into the five-bit shift register. The shift register is controlled by the PSHIFT input. The PSHIFT signal determines which rising CLK edge contains valid data on the PTDI input. When asserted the PSHIFT input causes data from the PTDI input to be latched and causes internal data bits already in the shift register to be passed on to the next shift register bit. Serial data eventually shifts out through the PTDO output. PTDO can be used as a status output and can be used to verify that the shift register is operating properly. Do not assert both PAPPLY and PSHIFT on the same CLK edge as this is interpreted as a “no operation”.

Table 10-24. Miscellaneous Instruction Execution Times (Continued)

Instruction	Size	Register	Memory	Reg -> Dest	Source -> Reg
PLPA (ATC hit)	—	15(0/0)	—	—	—
PLPA (ATC miss)	—	28(0/0)	—	—	—
PFLUSH	—	18(0/0)	—	—	—
PFLUSHN	—	18(0/0)	—	—	—
PFLUSHAN	—	33(0/0)	—	—	—
PFLUSHA	—	33(0/0)	—	—	—
RESET	—	520(0/0)	—	—	—
STOP	Word	8(0/0)	—	—	—
SWAP	Word	1(0/0)	—	—	—
TRAPF	—	1(0/0)	—	—	—
TRAPcc	—	1(0/0)	—	—	—
TRAPV	—	1(0/0)	—	—	—
UNLK	—	1(1/0)	—	—	—
UNPK	—	2(0/0)	2(1/1)	—	—

[1] For these entries, add the effective address calculation time.
[2] For the CPUSH instruction, the operand write figure refers to line-sized transfers.

10.15 FPU INSTRUCTION EXECUTION TIMES

Table 10-25 shows the number of clock cycles required for execution of the floating-point instructions, including completion of the operation and storing of the result. The number of operand read and write cycles is shown in parentheses (r/w).

Table 10-25. Floating-Point Instruction Execution Times

Instruction	Effective Address, <ea>									
	FPn	Dn	(An)	(An)+	–(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	(bd,An,XI*SF) (bd,PC,XI*SF)	(xxx).WL	#<imm>
FABS	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FDABS	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FSABS	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FADD	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FDADD	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FSADD	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FCMP	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FDIV	37(0/0)	39(0/0)	37(1/0)	37(1/0)	37(1/0)	37(1/0)	38(1/0)	39(1/0)	38(1/0)	38(0/0)
FDDIV	37(0/0)	39(0/0)	37(1/0)	37(1/0)	37(1/0)	37(1/0)	38(1/0)	39(1/0)	38(1/0)	38(0/0)
FSDIV	37(0/0)	39(0/0)	37(1/0)	37(1/0)	37(1/0)	37(1/0)	38(1/0)	39(1/0)	38(1/0)	38(0/0)
FMOVE ,FPx	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
FDMOVE ,FPx	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
FSMOVE ,FPx	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
FMOVE FPy,	—	3(0/0)	1(0/1)	1(0/1)	1(0/1)	1(1/0)	2(0/1)	3(0/1)	2(0/1)	—
FMOVE ,FPCR	—	8(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	8(1/0)	7(1/0)	7(0/0)
FMOVE FPCR,	—	4(0/0)	2(0/1)	2(0/1)	2(0/1)	2(1/0)	3(0/1)	4(0/1)	3(0/1)	—
FINT	3(0/0)	4(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	5(1/0)	3(1/0)	3(0/0)
FINTRZ	3(0/0)	4(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	5(1/0)	3(1/0)	3(0/0)

11.2.6 Clocking

For systems which have PCLK-to-BCLK skew controlled by a phase-locked-loop (PLL) clock generator such as the 88915 or 88916, it is possible to connect the PCLK of the MC68040 to the MC68060 CLK input as shown in Figure 11-8. Otherwise, the MC68060 CLK must be generated by an 88915 PLL as shown in Figure 11-9.

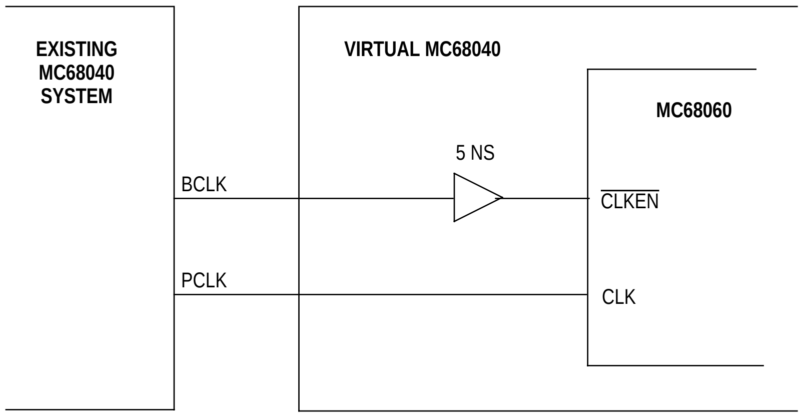


Figure 11-8. Simple CLK Generation

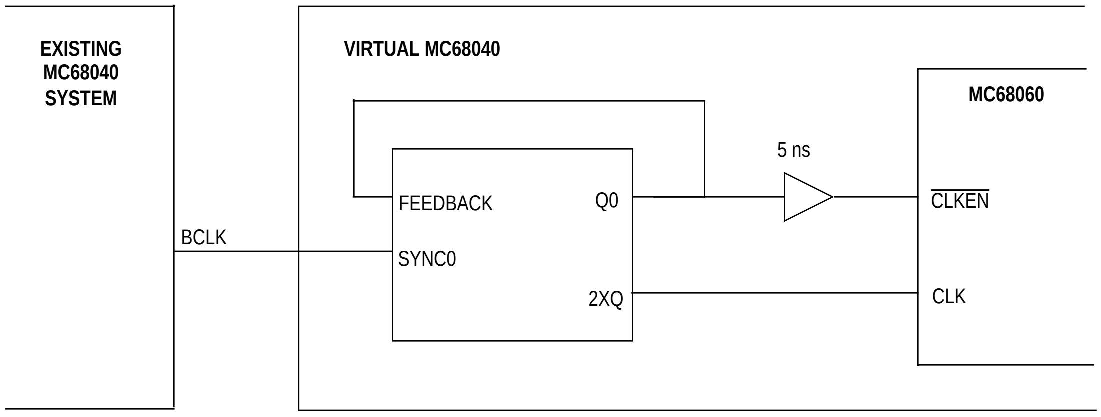


Figure 11-9. Generic CLK Generation

Appropriate generation of the $\overline{\text{CLKEN}}$ signal to enable 1/2-speed operation is easily achieved by delaying the MC68040 BCLK by 5 ns before feeding it into the $\overline{\text{CLKEN}}$ input of the MC68060.

Be aware that a clock skew exists between CLK and BCLK. The MC88915 can only control the skew to within 1 ns. Figure 11-10 shows the relationship between BCLK and $\overline{\text{CLKEN}}$.

11.2.7 PSTx Encoding

PSTx signal encoding is different between the MC68060 and MC68040. This should not affect normal applications because PSTx signals are not used for bus control logic.

```
* _real_cas(), _real_cas2(): MC68060ISP Call-out to provide choice
* of using supplied _isp_cas() and _isp_cas2() routines or to
* write an alternate routine more fitted for the system.

* _isp_cas(), _isp_cas2(): CAS and CAS2 core routine entry point that
* can be called from _real_cas() and _real_cas2() if the system wishes
* to use the CAS and CAS2 emulation code provided with the package.
* The flow is:
*      (exception) -> _isp_unimp -> _real_cas{2) -> _isp_cas{2}

* _isp_cas_inrange(): Subroutine entry point provided by the 68060ISP
* for use by the access error handler that reports if a given
* address resides within the _isp_cas() or _isp_cas2() routines.
* Inputs:
*      a0 = instruction address in question
* Outputs:
*      d0 = 0 -> success; non-zero -> failure

* _isp_cas_terminate(): Entry point provided by the MC68060ISP for
* use by an access error handler to create an access error frame for
* a process and to exit the CAS or CAS2 emulation gracefully.
* Inputs:
*      a0 = faulting address
*      d0 = Fault Status Longword

* _isp_cas_restart(): Entry point provided by the 68060ISP for use
* by an access error handler to re-start _isp_cas() and _isp_cas2()
* if a recoverable bus error occurs within the _isp_cas() and _isp_cas2()
* routines.

* _isp_cas_finish(), _isp_cas2_finish(): Entry point provided by the
* MC68060ISP for use by system-specific implementations of cas. Enter
* here to exit gracefully through the package.
* The flow is:
*       (exception) ->_isp_unimp -> _real_cas{2} -> (new code)
*       -> _isp_cas{2}_finish
* This requires close examination of the _isp_cas() and _isp_cas2() source
* code.
```

Figure C-4. CAS and CAS2 Call-Outs and Entry Points

C.2.3 Module 2: Unimplemented Integer Instruction Library (MC68060ILSP)

The M68060SP provides a library version of the following unimplemented integer instructions: 64-bit divide, 64-bit multiply, and CMP2. This version can be compiled with user applications desiring the functionality of these instructions. Using the library method, an application does not have to incur the overhead of the unimplemented integer instruction exception.

The routines are System V ABI compliant. Currently, the arguments are expected on the stack by the M68060SP library routines. For _divu64, _divs64, _mulu64, and _muls64, the results are not returned in a pair of data registers as with the actual instructions, but rather in a two-long-word memory array pointed to by a pointer argument provided by the caller.

CPUSH Push and Possibly Invalidate Cache Line CPUSH

(MC68060, MC68LC060, MC68EC060)

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	CACHE		1	SCOPE		REGISTER		

Instruction Fields:

Cache field—Specifies the Cache.
- 00—No Operation
- 01—Data Cache
- 10—Instruction Cache
- 11—Data and Instruction Caches

Scope field—Specifies the Scope of the Operation.
- 00—Illegal (causes illegal instruction trap)
- 01—Line
- 10—Page
- 11—All

Register field—Specifies the address register for line and page operations. For line operations, the low-order bits 3–0 of the address are don’t care. Bits 11–0 or 12–0 of the address are don’t care for 4K-byte or 8K-byte page operations, respectively.