

Chapter 4

Local Memory

This chapter describes the MCF5307 implementation of the ColdFire Version 3 local memory specification. It consists of two major sections.

- Section 4.2, “SRAM Overview,” describes the MCF5307 on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.
- Section 4.7, “Cache Overview,” describes the MCF5307 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interfaces with other memory structures.

4.1 Interactions between Local Memory Modules

Depending on configuration information, instruction fetches and data read accesses may be sent simultaneously to the RAM and cache controllers. This approach is required because both controllers are memory-mapped devices and the hit/miss determination is made concurrently with the read data access. Power dissipation can be minimized by configuring the RAMBARs to mask unused address spaces whenever possible.

If the access address is mapped into the region defined by the RAM (and this region is not masked), the RAM provides the data back to the processor, and the cache data is discarded. Accesses from the RAM module are never cached. The complete definition of the processor’s local bus priority scheme for read references is as follows:

```
if (RAM "hits"  
)  
    RAM supplies data to the processor  
    else if (cache "hits")  
        cache supplies data to the processor  
    else system memory reference to access data
```

For data write references, the memory mapping into the local memories is resolved before the appropriate destination memory is accessed. Accordingly, only the targeted local memory is accessed for data write transfers.

4.2 SRAM Overview

The 4-Kbyte on-chip SRAM module is connected to the internal bus and provides

SRAM Operation

pipelined, single-cycle access to memory mapped to the module. Memory can be mapped to any 0-modulo-32K location in the 4-Gbyte address space and configured to respond to either instruction or data accesses. Time-critical functions can be mapped into instruction the system stack. Other heavily-referenced data can be mapped into memory.

The following summarizes features of the MCF5307 SRAM implementation:

- 4-Kbyte SRAM, organized as 1024 x 32 bits
- Single-cycle throughput. When the pipeline is full, one access can occur per clock cycle.
- Physical location on the processor's high-speed local bus
- Memory location programmable on any 0-modulo-32K address boundary
- Byte, word, and longword address capabilities
- The RAM base address register (RAMBAR) defines the logical base address, attributes, and access types for the SRAM module.

4.3 SRAM Operation

The SRAM module provides a general-purpose memory block that the ColdFire processor can access with single-cycle throughput. The location of the memory block can be specified to any word-aligned address in the 4-Gbyte address space by RAMBAR[BA], described in Section 4.4.1, “SRAM Base Address Register (RAMBAR).” The memory is ideal for storing critical code or data structures or for use as the system stack. Because the SRAM module connects physically to the processor's high-speed local bus, it can service processor-initiated accesses or memory-referencing debug module commands.

Instruction fetches and data reads can be sent to both the cache and SRAM blocks simultaneously. If the reference is mapped into a region defined by the SRAM, the SRAM provides data to the processor and any cache data is discarded. Data accessed from the SRAM module are not cached.

Note also that the SRAM cannot be accessed by the on-chip DMAs. The on-chip system configuration allows concurrent core and DMA execution, where the core can reference code or data from the internal SRAM or cache while performing a DMA transfer.

Accesses are attempted in the following order:

1. SRAM
2. Cache (if space is defined as cacheable)
3. External access

4.4 SRAM Programming Model

The SRAM programming model consists of RAMBAR.

4.4.1 SRAM Base Address Register (RAMBAR)

The SRAM modules are configured through the RAMBAR, shown in Figure 4-1.

- RAMBAR holds the base address of the SRAM. The MOVEC instruction provides write-only access to this register from the processor.
- RAMBAR can be read or written from the debug module in a similar manner.
- All undefined RAMBAR bits are reserved. These bits are ignored during writes to the RAMBAR and return zeros when read from the debug module.
- The valid bit, RAMBAR[V], is cleared at reset, disabling the SRAM module. All other bits are unaffected.

	31	15	14	9	8	7	6	5	4	3	2	1	0		
Field	BA			—			WP	—		C/I	SC	SD	UC	UD	V
Reset	—												0		
R/W	W for CPU; R/W for debug														
Address	CPU space + 0xC04														

Figure 4-1. SRAM Base Address Register (RAMBAR)

RAMBAR fields are described in detail in Table 4-1.

Table 4-1. RAMBAR Field Description

Bits	Name	Description
31–15	BA	Base address. Defines the SRAM module's word-aligned base address. The SRAM module occupies a 4-Kbyte space defined by the contents of BA. SRAM may reside on any 32-Kbyte boundary in the 4-Gbyte address space.
14–9	—	Reserved, should be cleared.
8	WP	Write protect. Controls read/write properties of the SRAM. 0 Allows read and write accesses to the SRAM module 1 Allows only read accesses to the SRAM module. Any attempted write reference generates an access error exception to the ColdFire processor core.
7–6	—	Reserved, should be cleared.

Table 4-1. RAMBAR Field Description (Continued)

Bits	Name	Description
5-1	C/I, SC, SD, UC, UD	Address space masks (ASn). These fields allow certain types of accesses to be masked, or inhibited from accessing the SRAM module. These bits are useful for power management as described in Section 4.6, "Power Management." In particular, C/I is typically set. The address space mask bits are follows: C/I = CPU space/interrupt acknowledge cycle mask. Note that C/I must be set if BA = 0. SC = Supervisor code address space mask SD = Supervisor data address space mask UC = User code address space mask UD = User data address space mask For each ASn bit: 0 An access to the SRAM module can occur for this address space 1 Disable this address space from the SRAM module. If a reference using this address space is made, it is inhibited from accessing the SRAM module and is processed like any other non-SRAM reference.
0	V	Valid. Enables/disables the SRAM module. V is cleared at reset. 0 RAMBAR contents are not valid. 1 RAMBAR contents are valid.

The mapping of a given access into the RAM uses the following algorithm to determine if the access hits in the memory:

```

if (RAMBAR[0] = 1)
if (requested address[31:15] = RAMBAR[31:15])
    if (requested address[14:12] = 0)
        if (ASn of the requested type = 0)
            Access is mapped to the RAM module
            if (access = read)
                Read the RAM and return the data
            if (access = write)
                if (RAMBAR[8] = 0)
                    Write the data into the RAM
                else Signal a write-protect access error

```

ASn refers to the five address space mask bits: C/I, SC, SD, UC, and UD.

4.5 SRAM Initialization

After a hardware reset, the contents of the SRAM module are undefined. The valid bit, RAMBAR[V], is cleared, disabling the SRAM module. If the SRAM requires initialization with instructions or data, the following steps should be performed:

1. Read the source data and write it to the SRAM. Various instructions support this function, including memory-to-memory move instructions and the move multiple instruction (MOVEM). MOVEM is optimized to generate line-sized burst fetches on line-aligned addresses, so it generally provides maximum performance.
2. After the data is loaded into the SRAM, it may be appropriate to revise the RAMBAR attribute bits, including the write-protect and address space mask fields.

Remember that the SRAM cannot be accessed by the on-chip DMAs. The on-chip system configuration allows concurrent core and DMA execution where the core can execute code

out of internal SRAM or cache during DMA access.

The ColdFire processor or an external emulator using the debug module can perform these initialization functions.

4.5.1 SRAM Initialization Code

The code segment below initializes the SRAM. The code sets the base address of the SRAM at 0x2000_0000 and then initializes the RAM to zeros.

```
RAMBASE      EQU      0x20000000      ;set this variable to 0x20000000
RAMVALID     EQU      0x00000035
move.l       #RAMBASE+RAMVALID,D0    ;load RAMBASE + valid bit into D0
movec.l      D0, RAMBAR               ;load RAMBAR and enable SRAM
```

The following loop initializes the entire SRAM to zero:

```
lea.l        RAMBASE,A0              ;load pointer to SRAM
move.l       #1024,D0                ;load loop counter into D0

SRAM_INIT_LOOP:
clr.l        (A0)+                    ;clear 4 bytes of SRAM
subq.l       #1,D0                    ;decrement loop counter
bne.b        SRAM_INIT_LOOP          ;exit if done; else continue looping
```

The following function copies the number of bytesToMove from the source (*src) to the processor's local RAM at an offset relative to the SRAM base address defined by destinationOffset. The bytesToMove must be a multiple of 16. For best performance, source and destination SRAM addresses should be line-aligned (0-modulo-16).

```
; copyToCpuRam (*src, destinationOffset, bytesToMove)

RAMBASE      EQU      0x20000000      ;SRAM base address
RAMFLAGS     EQU      0x00000035      ;RAMBAR valid + mask bits

        lea.l       -12(a7),a7        ;allocate temporary space
        movem.l     #0x1c,(a7)        ;store D2/D3/D4 registers

; stack arguments and locations
; +0   saved d2
; +4   saved d3
; +8   saved d4
; +12  returnPc
; +16  pointer to source operand
; +20  destinationOffset
; +24  bytesToMove

        move.l      RAMBASE+RAMFLAGS,a0 ;define RAMBAR contents
        movec.l     a0,rambar           ;load it

        move.l      16(a7),a0          ;load argument defining *src

        lea.l      RAMBASE,a1         ;memory pointer to RAM base
        add.l       20(a7),a1         ;include destinationOffset
```

Power Management

```
        move.l    24(a7),d4          ;load byte count
        asr.l    #4,d4              ;divide by 16 to convert to loop count

loop:   .align    4                  ;force loop on 0-mod-4 address
        movem.l  (a0),#0xf          ;read 16 bytes from source
        movem.l  #0xf,(a1)         ;store into RAM destination
        lea.l    16(a0),a0         ;increment source pointer
        lea.l    16(a1),a1         ;increment destination pointer
        subq.l   #1,d4              ;decrement loop counter
        bne.b    loop              ;if done, then exit, else continue

        movem.l  (a7),#0x1c        ;restore d2/d3/d4 registers
        lea.l    12(a7),a7         ;deallocate temporary space
        rts
```

4.6 Power Management

Because processor memory references may be simultaneously sent to an SRAM module and cache, power can be minimized by configuring RAMBAR address space masks as precisely as possible. For example, if an SRAM is mapped to the internal instruction bus and contains instruction data, setting the ASn mask bits associated with operand references can decrease power dissipation. Similarly, if the SRAM contains data, setting ASn bits associated with instruction fetches minimizes power.

Table 4-2 shows typical RAMBAR configurations.

Table 4-2. Examples of Typical RAMBAR Settings

Data Contained in SRAM	RAMBAR[5-0]
Code only	0x2B
Data only	0x35
Both code and data	0x21

4.7 Cache Overview

This section describes the MCF5307 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

The MCF5307 processor contains a nonblocking, 8-Kbyte, 4-way set-associative, unified (instruction and data) cache with a 16-byte line size. The cache improves system performance by providing low-latency access to the instruction and data pipelines. This decouples processor performance from system memory performance, increasing bus availability for on-chip DMA or external devices. Figure 4-2 shows the organization and integration of the data cache.

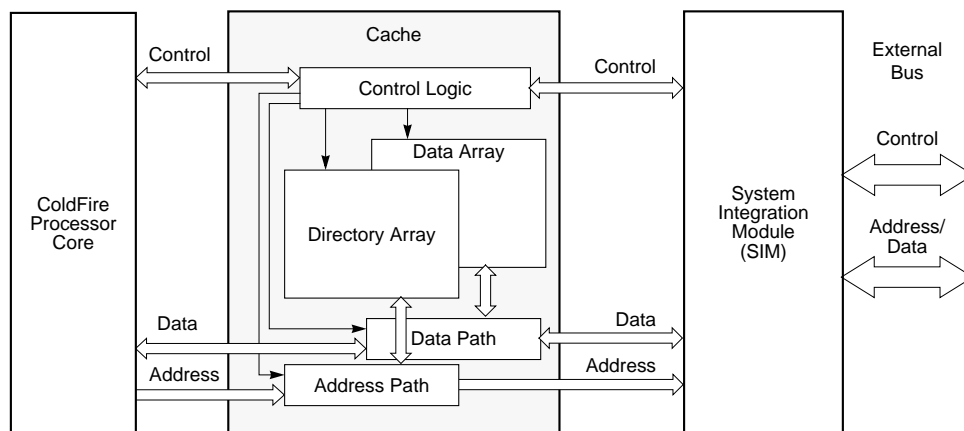


Figure 4-2. Unified Cache Organization

The cache supports operation of copyback, write-through, or cache-inhibited modes. The cache lock feature can be used to guarantee deterministic response for critical code or data areas.

A nonblocking cache services read hits or write hits from the processor while a fill (caused by a cache allocation) is in progress. As Figure 4-2 shows, instruction and data accesses use a single bus connected to the cache.

All addresses from the processor to the cache are physical addresses. A cache hit occurs when an address matches a cache entry. For a read, the cache supplies data to the processor. For a write, the processor updates the cache. If an access does not match a cache entry (misses the cache) or if a write access must be written through to memory, the cache performs a bus cycle on the internal bus and correspondingly on the external bus by way of the system integration module (SIM).

The SRAM module does not implement bus snooping; cache coherency with other possible bus masters must be maintained in software.

4.8 Cache Organization

A four-way set associative cache is organized as four ways (levels). There are 128 sets in the 8-Kbyte cache with each line containing 16 bytes (4 longwords). Entire cache lines are loaded from memory by burst-mode accesses that cache 4 longwords of data or instructions. All 4 longwords must be loaded for the cache line to be valid.

Figure 4-3 shows cache organization as well as terminology used.

Cache Organization

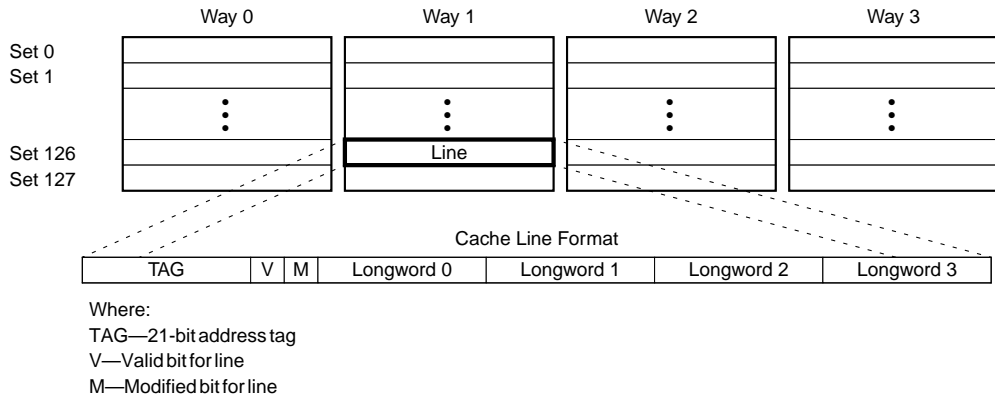


Figure 4-3. Cache Organization and Line Format

A set is a group of four lines (one from each level, or way), corresponding to the same index into the cache array.

4.8.1 Cache Line States: Invalid, Valid-Unmodified, and Valid-Modified

As shown in Table 4-3, a cache line can be invalid, valid-unmodified (often called exclusive), or valid-modified.

Table 4-3. Valid and Modified Bit Settings

V	M	Description
0	x	Invalid. Invalid lines are ignored during lookups.
1	0	Valid, unmodified. Cache line has valid data that matches system memory.
1	1	Valid, modified. Cache line contains most recent data, data at system memory location is stale.

A valid line can be explicitly invalidated by executing a CPUSHL instruction.

4.8.2 The Cache at Start-Up

As Figure 4-4 (A) shows, after power-up, cache contents are undefined; V and M may be set on some lines even though the cache may not contain the appropriate data for start up. Because reset and power-up do not invalidate cache lines automatically, the cache should be cleared explicitly by setting CACR[CINVA] before the cache is enabled (B).

After the entire cache is flushed, cacheable entries are loaded first in way 0. If way 0 is occupied, the cacheable entry is loaded into the same set in way 1, as shown in Figure 4-4 (D). This process is described in detail in Section 4.9, “Cache Operation.”

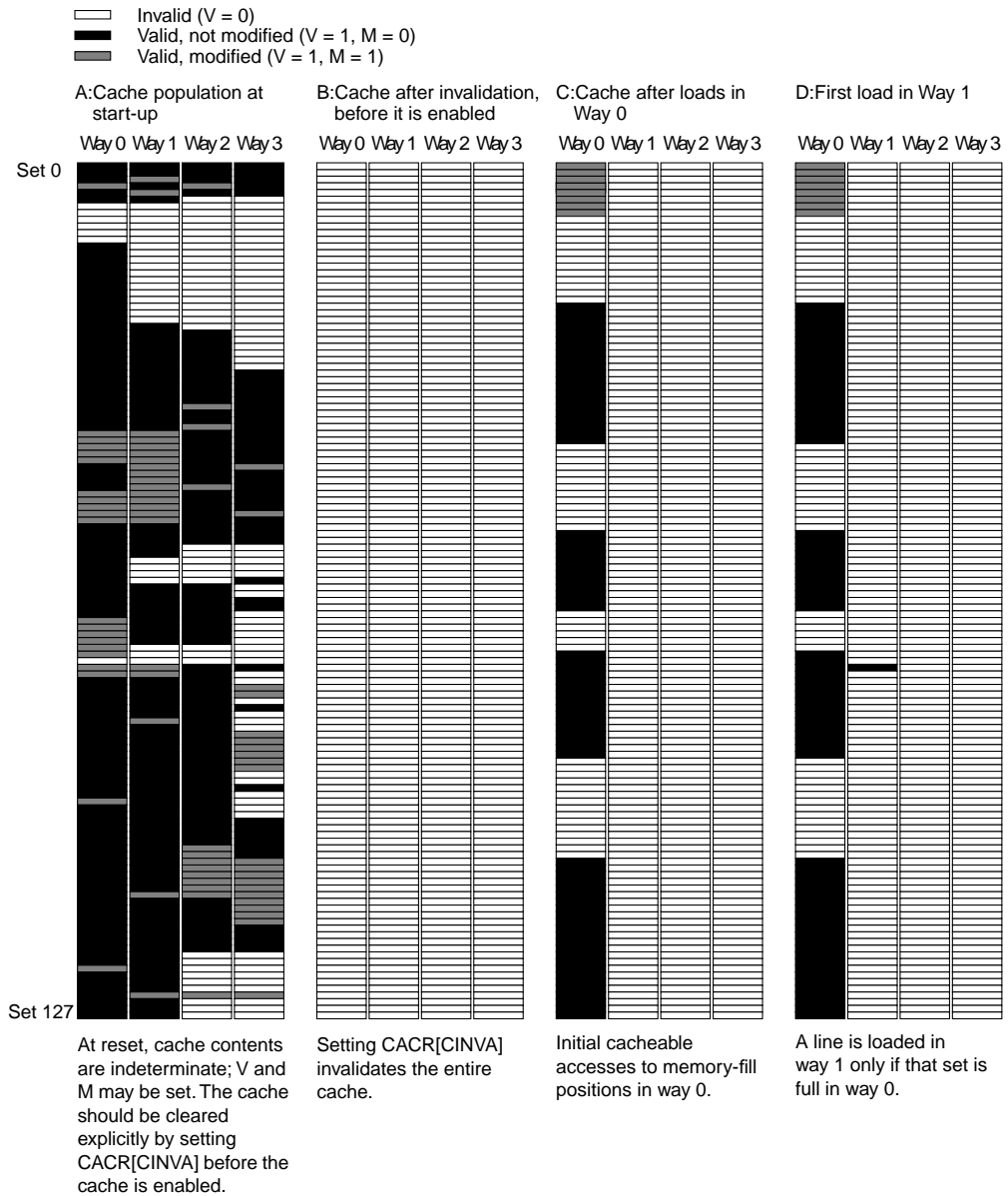


Figure 4-4. Cache—A: at Reset, B: after Invalidation, C and D: Loading Pattern

4.9 Cache Operation

Figure 4-5 shows the general flow of a caching operation.

Cache Operation

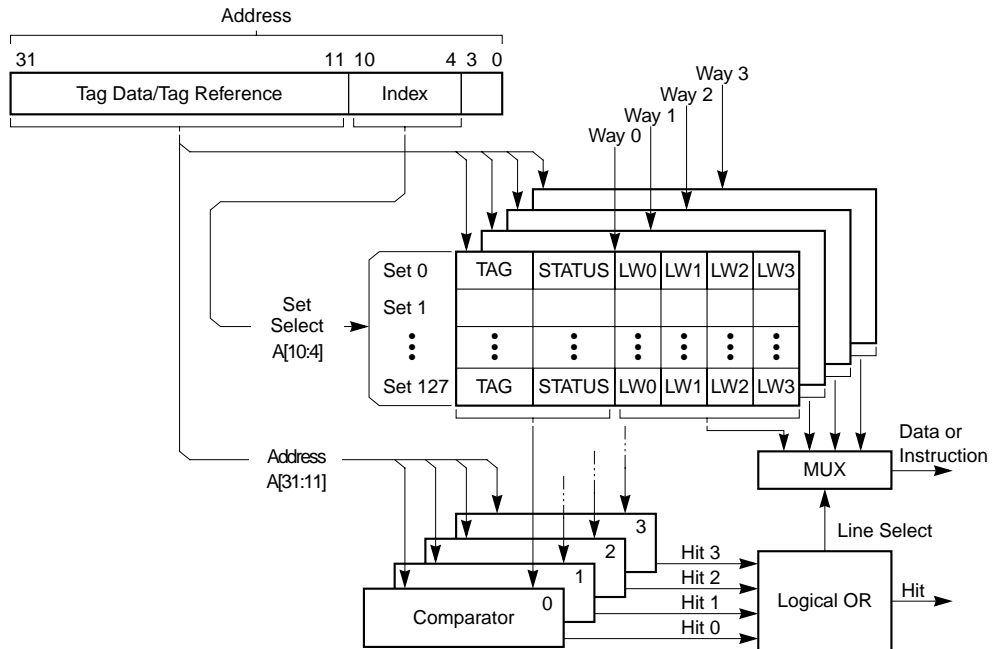


Figure 4-5. Caching Operation

The following steps determine if a cache line is allocated for a given address:

1. The cache set index, $A[10:4]$, selects one cache set.
2. $A[31:11]$ and the cache set index are used as a tag reference or are used to update the cache line tag field. Note that $A[31:11]$ can specify 21 possible addresses that can be mapped to one of the four ways.
3. The four tags from the selected cache set are compared with the tag reference. A cache hit occurs if a tag matches the tag reference and the V bit is set, indicating that the cache line contains valid data. If a cacheable write access hits in a valid cache line, the write can occur to the cache line without having to load it from memory.

If the memory space is copyback, the updated cache line is marked modified ($M = 1$), because the new data has made the data in memory out of date. If the memory location is write-through, the write is passed on to system memory and the M bit is never used. Note that the tag does not have TT or TM bits.

To allocate a cache entry, the cache set index selects one of the cache's 128 sets. The cache control logic looks for an invalid cache line to use for the new entry. If none is available, the cache controller uses a pseudo-round-robin replacement algorithm to choose the line to be deallocated and replaced. First the cache controller looks for an invalid line, with way 0 the highest priority. If all lines have valid data, a 2-bit replacement counter is used to choose the way. After a line is allocated, the pointer increments to point to the next way.

Cache lines from ways 0 and 1 can be protected from deallocation by enabling half-cache locking. If $\text{CACR}[\text{HLCK}] = 1$, the replacement pointer is restricted to way 2 or 3.

As part of deallocation, a valid, unmodified cache line is invalidated. It is consistent with system memory, so memory does not need to be updated. To deallocate a modified cache line, data is placed in a push buffer (for an external cache line push) before being invalidated. After invalidation, the new entry can replace it. The old cache line may be written after the new line is read.

When a cache line is selected to host a new cache entry, the following three things happen:

1. The new address tag bits $A[31:11]$ are written to the tag.
2. The cache line is updated with the new memory data.
3. The cache line status changes to a valid state ($V = 1$).

Read cycles that miss in the cache allocate normally as previously described.

Write cycles that miss in the cache do not allocate on a cacheable write-through region, but do allocate for addresses in a cacheable copyback region.

A copyback byte, word, longword, or line write miss causes the following:

1. The cache initiates a line fill or flush.
2. Space is allocated for a new line.
3. V and M are both set to indicate valid and modified.
4. Data is written in the allocated space. No write to memory occurs.

Note the following:

- Read hits cannot change the status bits and no deallocation or replacement occurs; the data or instructions are read from the cache.
- If the cache hits on a write access, data is written to the appropriate portion of the accessed cache line. Write hits in cacheable, write-through regions generate an external write cycle and the cache line is marked valid, but is never marked modified. Write hits in cacheable copyback regions do not perform an external write cycle; the cache line is marked valid and modified ($V = 1$ and $M = 1$).
- Misaligned accesses are broken into at least two cache accesses.
- Validity is provided only on a line basis. Unless a whole line is loaded on a cache miss, the cache controller does not validate data in the cache line.

Write accesses designated as cache-inhibited by the CACR or ACR bypass the cache and perform a corresponding external write.

Normally, cache-inhibited reads bypass the cache and are performed on the external bus. The exception to this normal operation occurs when all of the following conditions are true during a cache-inhibited read:

- The cache-inhibited fill buffer bit, $\text{CACR}[\text{DNFB}]$, is set.

Cache Operation

- The access is an instruction read.
- The access is normal (that is, transfer type (TT) equals 0).

In this case, an entire line is fetched and stored in the fill buffer. It remains valid there, and the cache can service additional read accesses from this buffer until either another fill or a cache-invalidate-all operation occurs.

Valid cache entries that match during cache-inhibited address accesses are neither pushed nor invalidated. Such a scenario suggests that the associated cache mode for this address space was changed. To avoid this, it is generally recommended to use the CPUSHL instruction to push or invalidate the cache entry or set CACR[CINVA] to invalidate the cache before switching cache modes.

4.9.1 Caching Modes

For every memory reference generated by the processor or debug module, a set of effective attributes is determined based on the address and the ACRs. Caching modes determine how the cache handles an access. An access can be cacheable in either write-through or copyback mode; it can be cache-inhibited in precise or imprecise modes. For normal accesses, the ACR n [CM] bit corresponding to the address of the access specifies the caching modes. If an address does not match an ACR, the default caching mode is defined by CACR[DCM]. The specific algorithm is as follows:

```
if (address == ACR0-address including mask)
    effective attributes = ACR0 attributes
else if (address == ACR1-address including mask)
    effective attributes = ACR1 attributes
else effective attributes = CACR default attributes
```

Addresses matching an ACR can also be write-protected using ACR[W]. Addresses that do not match either ACR can be write-protected using CACR[DW].

Reset disables the cache and clears all CACR bits. As shown in Figure 4-4, reset does not automatically invalidate cache entries; they must be invalidated through software.

The ACRs allow the defaults selected in the CACR to be overridden. In addition, some instructions (for example, CPUSHL) and processor core operations perform accesses that have an implicit caching mode associated with them. The following sections discuss the different caching accesses and their associated cache modes.

4.9.1.1 Cacheable Accesses

If ACR n [CM] or the default field of the CACR indicates write-through or copyback, the access is cacheable. A read access to a write-through or copyback region is read from the cache if matching data is found. Otherwise, the data is read from memory and the cache is updated. When a line is being read from memory for either a write-through or copyback read miss, the longword within the line that contains the core-requested data is loaded first and the requested data is given immediately to the processor, without waiting for the three

remaining longwords to reach the cache.

The following sections describe write-through and copyback modes in detail.

4.9.1.2 Write-Through Mode

Write accesses to regions specified as write-through are always passed on to the external bus, although the cycle can be buffered, depending on the state of CACR[ESB]. Writes in write-through mode are handled with a no-write-allocate policy—that is, writes that miss in the cache are written to the external bus but do not cause the corresponding line in memory to be loaded into the cache. Write accesses that hit always write through to memory and update matching cache lines. The cache supplies data to data-read accesses that hit in the cache; read misses cause a new cache line to be loaded into the cache.

4.9.1.3 Copyback Mode

Copyback regions are typically used for local data structures or stacks to minimize external bus use and reduce write-access latency. Write accesses to regions specified as copyback that hit in the cache update the cache line and set the corresponding M bit without an external bus access.

Be sure to flush the cache using the CPUSHL instruction before invalidating the cache in copyback mode. Modified cache data is written to memory only if the line is replaced because of a miss or a CPUSHL instruction pushes the line. If a byte, word, longword, or line write access misses in the cache, the required cache line is read from memory, thereby updating the cache. When a miss selects a modified cache line for replacement, the modified cache data moves to the push buffer. The replacement line is read into the cache and the push buffer contents are then written to memory.

4.9.2 Cache-Inhibited Accesses

Memory regions can be designated as cache-inhibited, which is useful for memory containing targets such as I/O devices and shared data structures in multiprocessing systems. It is also important to not cache the MCF5307 memory mapped registers. If the corresponding $ACR_n[CM]$ or $CACR[DCM]$ indicates cache-inhibited, precise or imprecise, the access is cache-inhibited. The caching operation is identical for both cache-inhibited modes, which differ only regarding recovery from an external bus error.

In determining whether a memory location is cacheable or cache-inhibited, the CPU checks memory-control registers in the following order:

1. RAMBAR
2. ACR0
3. ACR1
4. If an access does not hit in the RAMBAR or the ACRs, the default is provided for all accesses in CACR.

Cache Operation

Cache-inhibited write accesses bypass the cache and a corresponding external write is performed. Cache-inhibited reads bypass the cache and are performed on the external bus, except when all of the following conditions are true:

- The cache-inhibited fill-buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (that is, TT = 0).

In this case, a fetched line is stored in the fill buffer and remains valid there; the cache can service additional read accesses from this buffer until another fill occurs or a cache-invalidate-all operation occurs.

If ACR n [CM] indicates cache-inhibited mode, precise or imprecise, the controller bypasses the cache and performs an external transfer. If a line in the cache matches the address and the mode is cache-inhibited, the cache does not automatically push the line if it is modified, nor does it invalidate the line if it is valid. Before switching cache mode, execute a CPUSHL instruction or set CACR[CINVA] to invalidate the entire cache.

If ACR n [CM] indicates precise mode, the sequence of read and write accesses to the region is guaranteed to match the instruction sequence. In imprecise mode, the processor core allows read accesses that hit in the cache to occur before completion of a pending write from a previous instruction. Writes are not deferred past data-read accesses that miss the cache (that is, that must be read from the bus).

Precise operation forces data-read accesses for an instruction to occur only once by preventing the instruction from being interrupted after data is fetched. Otherwise, if the processor is not in precise mode, an exception aborts the instruction and the data may be accessed again when the instruction is restarted. These guarantees apply only when ACR n [CM] indicates precise mode and aligned accesses.

CPU space-register accesses, such as MOVEC, are treated as cache-inhibited and precise.

4.9.3 Cache Protocol

The following sections describe the cache protocol for processor accesses and assumes that the data is cacheable (that is, write-through or copyback).

4.9.3.1 Read Miss

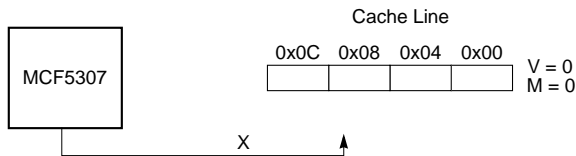
A processor read that misses in the cache requests the cache controller to generate a bus transaction. This bus transaction reads the needed line from memory and supplies the required data to the processor core. The line is placed in the cache in the valid state.

4.9.3.2 Write Miss

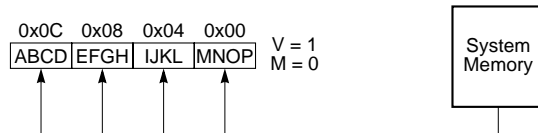
The cache controller handles processor writes that miss in the cache differently for write-through and copyback regions. Write misses to copyback regions cause the cache line

to be read from system memory, as shown in Figure 4-6.

- Writing character X to 0x0B generates a write miss. Data cannot be written to an invalid line.



- The cache line (characters A–P) is updated from system memory, and line is marked valid.



- After the cache line is filled, the write that initiated the write miss (the character X) completes to 0x0B.

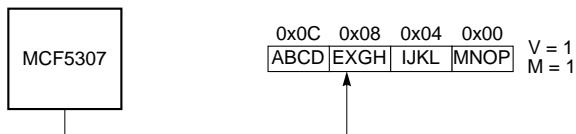


Figure 4-6. Write-Miss in Copyback Mode

The new cache line is then updated with write data and the M bit is set for the line, leaving it in modified state. Write misses to write-through regions write directly to memory without loading the corresponding cache line into the cache.

4.9.3.3 Read Hit

On a read hit, the cache provides the data to the processor core and the cache line state remains unchanged. If the cache mode changes for a specific region of address space, lines in the cache corresponding to that region that contain modified data are not pushed out to memory when a read hit occurs within that line. First execute a CPUSHL instruction or set CACR[CINVA] before switching the cache mode.

4.9.3.4 Write Hit

The cache controller handles processor writes that hit in the cache differently for write-through and copyback regions. For write hits to a write-through region, portions of cache lines corresponding to the size of the access are updated with the data. The data is also written to external memory. The cache line state is unchanged. For copyback accesses, the cache controller updates the cache line and sets the M bit for the line. An external write is not performed and the cache line state changes to (or remains in) the modified state.

4.9.4 Cache Coherency

The MCF5307 provides limited cache coherency support in multiple-master environments. Both write-through and copyback memory update techniques are supported to maintain coherency between the cache and memory.

The cache does not support snooping (that is, cache coherency is not supported while external or DMA masters are using the bus). Therefore, on-chip DMAs of the MCF5307 cannot access local memory and do not maintain coherency with the unified cache.

4.9.5 Memory Accesses for Cache Maintenance

The cache controller performs all maintenance activities that supply data from the cache to the core, including requests to the SIM for reading new cache lines and writing modified lines to memory. The following sections describe memory accesses resulting from cache fill and push operations. Chapter 18, “Bus Operation,” describes required bus cycles in detail.

4.9.5.1 Cache Filling

When a new cache line is required, a line read is requested from the SIM, which generates a burst-read transfer by indicating a line access with the size signals, `SIZ[1:0]`.

The responding device supplies 4 consecutive longwords of data. Burst operations can be inhibited or enabled through the burst read/write enable bits (`BSTR/BSTW`) in the chip-select control registers (`CSCR0–CSCR7`).

SIM line accesses implicitly request burst-mode operations from memory. For more information regarding external bus burst-mode accesses, see Chapter 18, “Bus Operation.”

The first cycle of a cache-line read loads the longword entry corresponding to the requested address. Subsequent transfers load the remaining longword entries.

A burst operation is aborted by a write-protection fault, which is the only possible access error. Exception processing proceeds immediately. Because the write cycle can be decoupled from the processor’s issuing of the operation, error signaling appears to be decoupled from the instruction that generated the write. Accordingly, the PC in the exception stack frame represents the program location when the access error was signaled. See Section 2.8.2, “Processor Exceptions.”

4.9.5.2 Cache Pushes

Cache pushes occur for line replacement and as required for the execution of the `CPUSHL` instruction. To reduce the requested data’s latency in the new line, the modified line being replaced is temporarily placed in the push buffer while the new line is fetched from memory. After the bus transfer for the new line completes, the modified cache line is written back to memory and the push buffer is invalidated.

4.9.5.2.1 Push and Store Buffers

The 16-byte push buffer reduces latency for requested new data on a cache miss by holding a displaced modified cache line while the new data is read from memory.

If a cache miss displaces a modified line, a miss read reference is immediately generated. While waiting for the response, the current contents of the cache location load into the push buffer. When the burst-read bus transaction completes, the cache controller can generate the appropriate line-write bus transaction to write the push buffer contents into memory.

In imprecise mode, the FIFO store buffer can defer pending writes to maximize performance. The store buffer can support as many as four entries (16 bytes maximum) for this purpose.

Data writes destined for the store buffer cannot stall the core. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (one per cycle maximum) and the external bus's ability to retire those writes. In imprecise mode, writes stall only if the store buffer is full and a write operation is on the internal bus. The internal write cycle is held, stalling the data execution pipeline.

If the store buffer is not used (that is, store buffer disabled or cache-inhibited precise mode), external bus cycles are generated directly for each pipeline write operation. The instruction is held in the pipeline until external bus transfer termination is received. Therefore, each write is stalled for 5 cycles, making the minimum write time equal to 6 cycles when the store buffer is not used. See Section 2.1.2.2, “Operand Execution Pipeline (OEP).”

The store buffer enable bit, CACR[ESB], controls the enabling of the store buffer. This bit can be set and cleared by the MOVEC instruction. ESB is zero at reset and all writes are performed in order (precise mode). ACRn[CM] or CACR[DCM] generates the mode used when ESB is set. Cacheable write-through and cache-inhibited imprecise modes use the store buffer.

The store buffer can queue data as much as 4 bytes wide per entry. Each entry matches the corresponding bus cycle it generates; therefore, a misaligned longword write to a write-through region creates two entries if the address is to an odd-word boundary. It creates three entries if it is to an odd-byte boundary—one per bus cycle.

4.9.5.2.2 Push and Store Buffer Bus Operation

As soon as the push or store buffer has valid data, the internal bus controller uses the next available external bus cycle to generate the appropriate write cycles. In the event that another cache fill is required (for example, cache miss to process) during the continued instruction execution by the processor pipeline, the pipeline stalls until the push and store buffers are empty, then generate the required external bus transaction.

Supervisor instructions, the NOP instruction, and exception processing synchronize the processor core and guarantee the push and store buffers are empty before proceeding. Note that the NOP instruction should be used only to synchronize the pipeline. The preferred

Cache Operation




no-operation function is the TPF instruction.

4.9.6 Cache Locking

Ways 0 and 1 of the cache can be locked by setting CACR[HLCK]. If the cache is locked, cache lines in ways 0 and 1 are not subject to being deallocated by normal cache operations.

As Figure 4-7 (B and C) shows, the algorithm for updating the cache and for identifying cache lines to be deallocated is otherwise unchanged. If ways 2 and 3 are entirely invalid, cacheable accesses are first allocated in way 2. Way 3 is not used until the location in way 2 is occupied.

Ways 0 and 1 are still updated on write hits (D in Figure 4-7) and may be pushed or cleared only explicitly by using specific cache push/invalidate instructions. However, new cache lines cannot be allocated in ways 0 and 1.

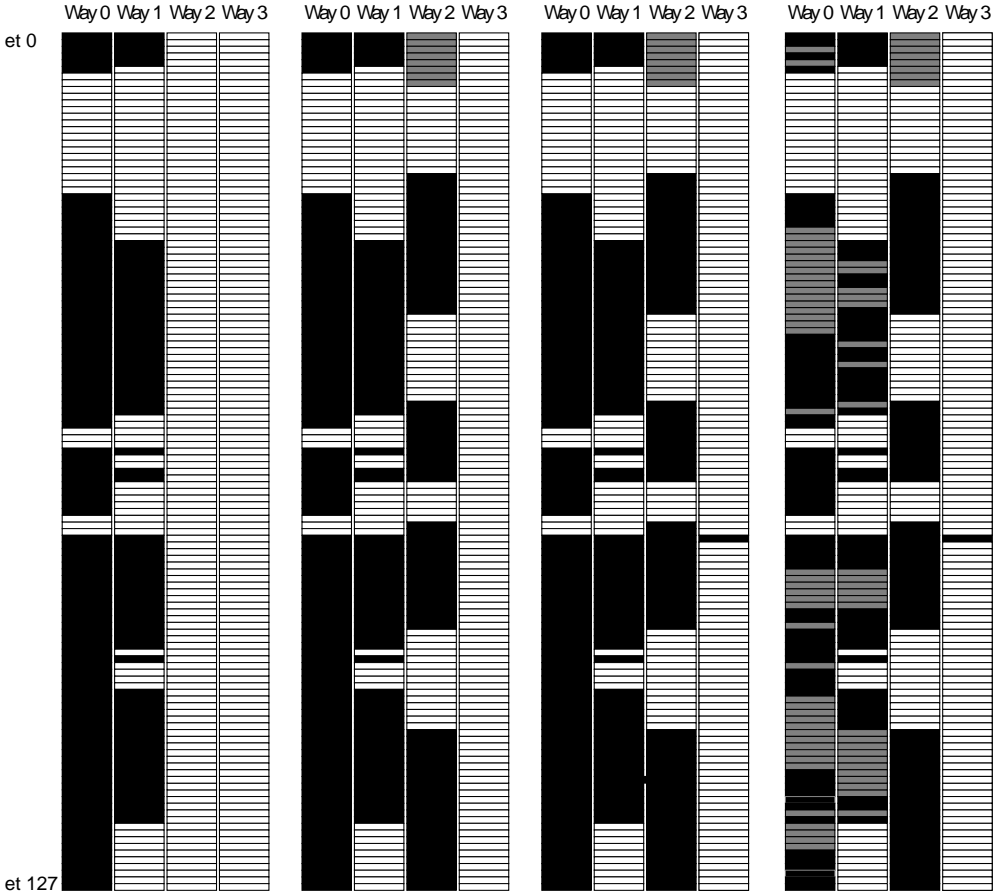
-  Invalid ($V = 0$)
-  Valid, not modified ($V = 1, M = 0$)
-  Valid, modified ($V = 1, M = 1$)

A: Ways 0 and 1 are filled. Ways 2 and 3 are invalid.

B: CACR[DH LCK] is set, locking ways 0 and 1.

C: When a set in Way 2 is occupied, the set in way 3 is used for a cacheable access.

D: Write hits to ways 0 and 1 update cache lines.



After reset, the cache is invalidated, ways 0 and 1 are then written with data that should not be deallocated.

After CACR[HLCK] is set, subsequent cache accesses go to ways 2 and 3.

While the cache is locked and after a position in ways is full, the set in Way 3 is updated.

While the cache is locked, ways 0 and 1 can be updated by write hits. In this example, memory is configured as copyback, so updated cache lines are marked modified.

Figure 4-7. Cache Locking

4.10 Cache Registers

This section describes the MCF5307 implementation of the Version 3 cache registers.

4.10.1 Cache Control Register (CACR)

The CACR in Figure 4-8 contains bits for configuring the cache. It can be written by the MOVEC register instruction and can be read or written from the debug facility. A hardware reset clears CACR, which disables the cache; however, reset does not affect the tags, state information, or data in the cache.

	31	30	29	28	27	26	25	24	23	20	19	18	17	16
Field	EC	—	ESB	DPI	HLCK	—	CINVA	—						
Reset	0000_0000_0000_0000													
R/W	Write (R/W by debug module)													
	15	14	13	12	11	10	9	8	7	0				
Field	—					DNFB	DCM	—	DW	—				
Reset	0000_0000_0000_0000													
R/W	Write (R/W by debug module)													
Rc	0x002													

Figure 4-8. Cache Control Register (CACR)

Table 4-4 describes CACR fields.

Table 4-4. CACR Field Descriptions

Bits	Name	Description
31	EC	Enable cache. 0 Cache disabled. The cache is not operational, but data and tags are preserved. 1 Cache enabled.
30	—	Reserved, should be cleared.
29	ESB	Enable store buffer. 0 Writes to write-through or noncachable in imprecise mode bypass the store buffer and generate bus cycles directly. Section 4.9.5.2.1, "Push and Store Buffers," describes the performance penalty for this. 1 The four-entry FIFO store buffer is enabled; when imprecise mode is used, this buffer defers pending writes to write-through or cache-inhibited regions to maximize performance. Cache-inhibited, precise-mode accesses always bypass the store buffer.
28	DPI	Disable CPUSHL invalidation. 0 Normal operation. A CPUSHL instruction causes the selected line to be pushed if modified and then invalidated. 1 No clear operation. A CPUSHL instruction causes the selected line to be pushed if modified, then left valid.

Table 4-4. CACR Field Descriptions (Continued)

Bits	Name	Description
27	HLCK	Half-cache lock mode 0 Normal operation. The cache allocates the lowest invalid way. If all ways are valid, the cache allocates the way pointed at by the counter and then increments this counter modulo-4. 1 Half-cache operation. The cache allocates to the lower invalid way of levels 2 and 3; if both are valid, the cache allocates to way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates way 3 and increments the round-robin counter modulo-2. This locks the content of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions. This implementation allows maximum use of available cache memory and provides the flexibility of setting HLCK before, during, or after allocations occur.
26–25	—	Reserved, should be cleared.
24	CINVA	Cache invalidate all. Writing a 1 to this bit initiates entire cache invalidation. Once invalidation is complete, this bit automatically returns to 0; it is not necessary to clear it explicitly. Note the caches are not cleared on power-up or normal reset, as shown in Figure 4-4. 0 No invalidation is performed. 1 Initiate invalidation of the entire cache. The cache controller sequentially clears V and M bits in all sets. Subsequent accesses stall until the invalidation is finished, at which point, this bit is automatically cleared. In copyback mode, the cache should be flushed using a CPUSHL instruction before setting this bit.
23–11	—	Reserved, should be cleared.
10	DNFB	Default noncacheable fill buffer. Determines if the fill buffer can store noncacheable accesses 0 Fill buffer not used to store noncacheable instruction accesses (16 or 32 bits). 1 Fill buffer used to store noncacheable accesses. The fill buffer is used only for normal (TT = 0) instruction reads of a noncacheable region. Instructions are loaded into the fill buffer by a burst access (same as a line fill). They stay in the buffer until they are displaced, so subsequent accesses may not appear on the external bus. Note that this feature can cause a coherency problem for self-modifying code. If DNFB = 1 and a cache-inhibited access uses the fill buffer, instructions remain valid in the fill buffer until a cache-invalidate-all instruction, another cache-inhibited burst, or a miss that initiates a fill. A write to the line in the fill buffer goes to the external bus without updating or invalidating the buffer. Subsequent reads of that written data are serviced by the fill buffer and receive stale information.
9–8	DCM	Default cache mode. Selects the default cache mode and access precision as follows: 00 Cacheable, write-through 01 Cacheable, copy-back 10 Cache-inhibited, precise exception model 11 Cache-inhibited, imprecise exception model. Precise and imprecise modes are described in Section 4.9.2, “Cache-Inhibited Accesses.”
7–6	—	Reserved, should be cleared.
5	DW	Default write protect. Use of this bit is described in Section 4.9.1, “Caching Modes.” 0 Read and write accesses permitted 1 Write accesses not permitted
4–0	—	Reserved, should be cleared.

4.10.2 Access Control Registers (ACR0–ACR1)

The ACRs, Figure 4-9, assign control attributes, such as cache mode and write protection, to specified memory regions. Registers are accessed with the MOVEC instruction with the Rc encodings in Figure 4-9.

For overlapping regions, ACR0 takes priority. Data transfers to and from these registers are

Cache Registers

longword transfers. Bits 12–7, 4, 3, 1, and 0 are always read as zeros.

NOTE:

The SIM MBAR region should be mapped as cache-inhibited through an ACR.

	31	24 23	16 15 14 13 12	7 6 5 4 3 2 1 0									
Field	Address Base		Address Mask			E	S	—		CM	—	W	—
R/W	Uninitialized					0	Uninitialized						
R/W	Write (R/W by debug module)												
Rc	ACR0: 0x004; ACR1: 0x005												

Figure 4-9. Access Control Register Format (ACR_n)

Table 4-5 describes ACR_n fields.

Table 4-5. ACR_n Field Descriptions

Bits	Name	Description
31–24	Address base	Address base. Compared with address bits A[31:24]. Eligible addresses that match are assigned the access control attributes of this register.
23–16	Address mask	Address mask. Setting a mask bit causes the corresponding address base bit to be ignored. The low-order mask bits can be set to define contiguous regions larger than 16 Mbytes. The mask can define multiple noncontiguous regions of memory.
15	E	Enable. Enables or disables the other ACR _n bits. 0 Access control attributes disabled 1 Access control attributes enabled
14–13	S	Supervisor mode. Specifies whether only user or supervisor accesses are allowed in this address range or if the type of access is a don't care. 00 Match addresses only in user mode 01 Match addresses only in supervisor mode 1x Execute cache matching on all accesses
12–7	—	Reserved; should be cleared.
6–5	CM	Cache mode. Selects the cache mode and access precision. Precise and imprecise modes are described in Section 4.9.2, "Cache-Inhibited Accesses." 00 Cacheable, write-through 01 Cacheable, copyback 10 Cache-inhibited, precise 11 Cache-inhibited, imprecise
4–3	—	Reserved; should be cleared.
2	W	Write protect. Selects the write privilege of the memory region. 0 Read and write accesses permitted 1 Write accesses not permitted
1–0	—	Reserved; should be cleared.

4.11 Cache Management

The cache can be enabled and configured by using a MOVEC instruction to access CACR. A hardware reset clears CACR, disabling the cache and removing all configuration information; however, reset does not affect the tags, state information, and data in the cache.

Set CACR[CINVA] to invalidate the cache before enabling it.

The privileged CPUSHL instruction supports cache management by selectively pushing and invalidating cache lines. The address register used with CPUSHL directly addresses the cache's directory array. The CPUSHL instruction flushes a cache line.

The value of CACR[DPI] determines whether CPUSHL invalidates a cache line after it is pushed. To push the entire cache, implement a software loop to index through all sets and through each of the four lines within each set (a total of 512 lines). The state of CACR[EC] does not affect the operation of CPUSHL or CACR[CINVA]. Disabling the cache by setting CACR[EC] makes the cache nonoperational without affecting tags, state information, or contents.

The contents of An used with CPUSHL specify cache row and line indexes. This differs from the MC68040 where a physical address is specified. Figure 4-10 shows the An format.

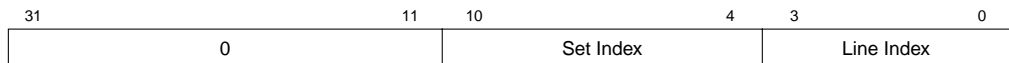


Figure 4-10. An Format

Bits A[10:4] specify a set index and bits A[3:0] specify the cache way. The following code example flushes the entire cache:

```

_cache_disable:
    nop
    move.w      #0x2700,SR      ;mask off IRQs
    jsr        _cache_flush   ;flush the cache completely
    clr.l      d0
    movec     d0,ACR0         ;ACR0 off
    movec     d0,ACR1         ;ACR1 off
    move.l    #0x01000000,d0   ;Invalidate and disable cache
    movec     d0,CACR
    rts

_cache_flush:
    nop                    ;synchronize—flush store buffer
    moveq.l   #0,d0         ;initialize way counter
    moveq.l   #0,d1         ;initialize set counter
    move.l    d0,a0         ;initialize cpushl pointer

setloop:
    cpushl   bc,(a0)        ;push cache line a0
    add.l    #0x0010,a0     ;increment set index by 1
    addq.l   #1,d1          ;increment set counter
    cmpi.l   #128,d1        ;are sets for this way done?
    bne
    moveq.l   #0,d1         ;set counter to zero again

```

Cache Operation Summary

```
addq.l    #1,d0           ;increment to next way
move.l    d0,a0           ;set = 0, way = d0
cmpi.l    #4,d0           ;flushed all the ways?
bne       setloop
rts
```

The following CACR loads assume the default cache mode is copyback.

CacheLoadAndLock:

```
move.l    #0xA1000100,d0; enable and invalidate cache ...
movec     d0,cacr ; ... in the CACR
```

The following code preloads half of the cache (4 Kbytes). It assumes a contiguous block of data is to be mapped into the cache, starting at a 0-modulo-4K address.

```
move.l    #256,d0         ;256 16-byte lines in 4K space
lea       data_,a0        ; load pointer defining data area
CacheLoop:
tst.b     (a0)             ;touch location + load into data cache
lea       16(a0),a0        ;increment address to next line
subq.l    #1,d0           ;decrement loop counter
bne.b     CacheLoop       ;if done, then exit, else continue

; A 4K region has been loaded into levels 0 and 1 of the 8K cache. lock it!
move.l    #0xA8000100,d0 ;set the cache lock bit ...
movec     d0,cacr         ; ... in the CACR
rts

align    16
```

4.12 Cache Operation Summary

This section gives operational details for the cache and presents cache-line state diagrams.

4.12.1 Cache State Transitions

Using the V and M bits, the cache supports a line-based protocol allowing individual cache lines to be invalid, valid, or modified. To maintain memory coherency, the cache supports both write-through and copyback modes, specified by the corresponding ACR[CM], or CACR[DCM] if no ACR matches.

Read or write misses to copyback regions cause the cache controller to read a cache line from memory into the cache. If available, tag and data from memory update an invalid line in the selected set. The line state then changes from invalid to valid by setting the V bit. If all lines in the row are already valid or modified, the pseudo-round-robin replacement algorithm selects one of the four lines and replaces the tag and data. Before replacement, modified lines are temporarily buffered and later copied back to memory after the new line has been read from memory.

Figure 4-11 shows the three possible cache line states and possible processor-initiated transitions for memory configured as copyback. Transitions are labeled with a capital letter indicating the previous state and a number indicating the specific case listed in Table 4-7.

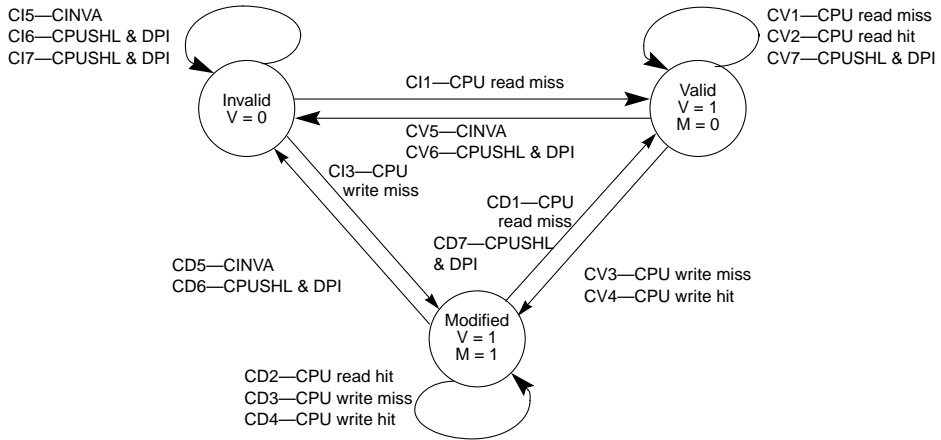


Figure 4-11. Cache Line State Diagram—Copyback Mode

Figure 4-13 shows the two possible states for a cache line in write-through mode.

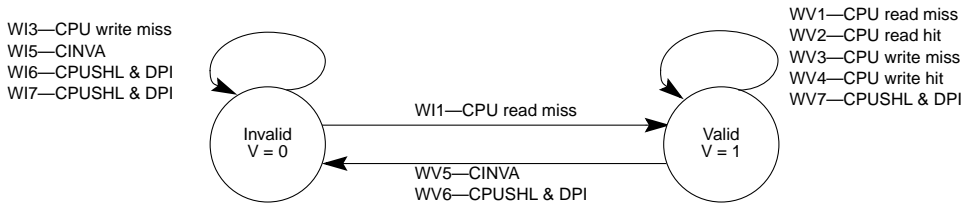


Figure 4-12. Cache Line State Diagram—Write-Through Mode

Table 4-7 describes cache line transitions and the accesses that cause them.

Table 4-6. Cache Line State Transitions

Access	Current State					
	Invalid (V = 0)		Valid (V = 1, M = 0)		Modified (V = 1, M = 1)	
Read miss	(C,W)I1	Read line from memory and update cache; supply data to processor; go to valid state.	(C,W)V1	Read new line from memory and update cache; supply data to processor; stay in valid state.	CD1	Push modified line to buffer; read new line from memory and update cache; supply data to processor; write push buffer contents to memory; go to valid state.
Read hit	(C,W)I2	Not possible.	(C,W)V2	Supply data to processor; stay in valid state.	CD2	Supply data to processor; stay in modified state.

Table 4-6. Cache Line State Transitions (Continued)

Access	Current State					
	Invalid (V = 0)		Valid (V = 1, M = 0)		Modified (V = 1, M = 1)	
Write miss (copy-back)	CI3	Read line from memory and update cache; write data to cache; go to modified state.	CV3	Read new line from memory and update cache; write data to cache; go to modified state.	CD3	Push modified line to buffer; read new line from memory and update cache; write push buffer contents to memory; stay in modified state.
Write miss (write-through)	WI3	Write data to memory; stay in invalid state.	WV3	Write data to memory; stay in valid state.	WD3	Write data to memory; stay in modified state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[CINVA] before switching modes.
Write hit (copy-back)	CI4	Not possible.	CV4	Write data to cache; go to modified state.	CD4	Write data to cache; stay in modified state.
Write hit (write-through)	WI4	Not possible.	WV4	Write data to memory and to cache; stay in valid state.	WD4	Write data to memory and to cache; go to valid state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[CINVA] before switching modes.
Cache invalidate	(C,W)I5	No action; stay in invalid state.	(C,W)V5	No action; go to invalid state.	CD5	No action (modified data lost); go to invalid state.
Cache push	(C,W)I6 (C,W)I7	No action; stay in invalid state.	(C,W)V6	No action; go to invalid state.	CD6	Push modified line to memory; go to invalid state.
			(C,W)V7	No action; stay in valid state.	CD7	Push modified line to memory; go to valid state.

The following tables present the same information as Table 4-7, organized by the current state of the cache line. In Table 4-8 the current state is invalid.

Table 4-7. Cache Line State Transitions (Current State Invalid)

Access	Response	
Read miss	(C,W)I1	Read line from memory and update cache; supply data to processor; go to valid state.
Read hit	(C,W)I2	Not possible

Table 4-7. Cache Line State Transitions (Current State Invalid) (Continued)

Access	Response	
Write miss (copyback)	CI3	Read line from memory and update cache; write data to cache; go to modified state.
Write miss (write-through)	WI3	Write data to memory; stay in invalid state.
Write hit (copyback)	CI4	Not possible
Write hit (write-through)	WI4	Not possible
Cache invalidate	(C,W)I5	No action; stay in invalid state.
Cache push	(C,W)I6	No action; stay in invalid state.
Cache push	(C,W)I7	No action; stay in invalid state.

In Table 4-9 the current state is valid.

Table 4-8. Cache Line State Transitions (Current State Valid)

Access	Response	
Read miss	(C,W)V1	Read new line from memory and update cache; supply data to processor; stay in valid state.
Read hit	(C,W)V2	Supply data to processor; stay in valid state.
Write miss (copyback)	CV3	Read new line from memory and update cache; write data to cache; go to modified state.
Write miss (write-through)	WV3	Write data to memory; stay in valid state.
Write hit (copyback)	CV4	Write data to cache; go to modified state.
Write hit (write-through)	WV4	Write data to memory and to cache; stay in valid state.
Cache invalidate	(C,W)V5	No action; go to invalid state.
Cache push	(C,W)V6	No action; go to invalid state.
Cache push	(C,W)V7	No action; stay in valid state.

In Table 4-10 the current state is modified.

Table 4-9. Cache Line State Transitions (Current State Modified)

Access	Response	
Read miss	CD1	Push modified line to buffer; read new line from memory and update cache; supply data to processor; write push buffer contents to memory; go to valid state.
Read hit	CD2	Supply data to processor; stay in modified state.
Write miss (copyback)	CD3	Push modified line to buffer; read new line from memory and update cache; write push buffer contents to memory; stay in modified state.
Write miss (write-through)	WD3	Write data to memory; stay in modified state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[CINVA] before switching modes.
Write hit (copyback)	CD4	Write data to cache; stay in modified state.
Write hit (write-through)	WD4	Write data to memory and to cache; go to valid state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[CINVA] before switching modes.
Cache invalidate	CD5	No action (modified data lost); go to invalid state.
Cache push	CD6	Push modified line to memory; go to invalid state.
Cache push	CD7	Push modified line to memory; go to valid state.

4.13 Cache Initialization Code

The following example sets up the cache for FLASH or ROM space only.

```

move.l #0x81000300,D0 //enable cache, invalidate it,
                        //default mode is cache-inhibited imprecise
movecD0, CACR

move.l #0xFF00C000,D0 //cache FLASH space, enable,
                        //ignore FC2, cacheable, writethrough
movecD0, ACR0

```