Order Number: AN1856/D
Rev. 0, 5/2000

Application Note

Migrating from the ColdFire® MCF5307 to the MCF5407

This application note highlights the differences between the MCF5307B and MCF5407. Users of the MCF5307 and MCF5307A should use this document in conjunction with the *MCF5307 User's Manual Mask Set Addendum*. For additional information, see the *MCF5407 Integrated ColdFire Microprocessor Product Brief*.

This document consists of the following sections:

1.1 Overview

The MCF5407 offers an easy upgrade and more than triples the performance of the MCF5307. The MCF5407 is the first standard product of the ColdFire family to contain the Version 4 (V4) ColdFire microprocessor core. To create this next-generation, high-performance core, many advanced design features are implemented. Most notable are a Harvard memory architecture, branch cache acceleration logic, and limited superscalar dual-instruction issue support, which results in 257 MIPS at 162 MHz (Dhrystone 2.1).

Customers using the integrated peripherals of the MCF5307 can access the same features on the MCF5407 with the added advantage of increased cache and RAM memories as well as an enhanced instruction set architecture (ISA), DMA, synchronous UART, and debug functionality. Decreased voltage requirements allow designers to take advantage of other low-voltage components on the board for an integrated, low-power system.

To migrate designs from the MCF5307 to the MCF5407, note the minor differences between these code-compatible processors in the initialization code, power supplies, and clock inputs. This document describes the differences between the processors and outlines the steps to upgrade the design. Table 1 is a quick reference chart of these differences.

Table 1. Differences between MCF5307 and MCF5407

Feature	MCF5307	MCF5407	Reference
Version core	ColdFire version 3 (V3)	ColdFire version 4	—
MIPS	70 MIPS at 90-MHz core clock	257 MIPS at 162-MHz core clock	—
Instruction set	Baseline ColdFire ISA RevA is used in version 2 and version 3 core.	ColdFire ISA RevB which includes certain instruction enhancements and some instruction additions; V2/V3 ISA Rev A is upward-compatible with ISA Rev B.	Section 1.2, “Instruction Set Additions”
Caches	8-Kbyte unified cache	16-Kbyte instruction cache 8-Kbyte data cache	Section 1.3, “Enhanced Memories”
	Two cache access control registers (ACR0/ACR1)	ACR0/ACR1 configure data space; ACR2/ACR3 configure instruction space	
	4-Kbyte SRAM	Two independently configurable 2-Kbyte SRAMs	
	No cache locking	Ability to lock all or half of the instruction cache to prevent instructions from being cast out. This is useful for deterministic code.	
DMA modifications	DMA acknowledge assertion is encoded on TM[2:0].	DACK[1:0] multiplexed on TM[1:0] can be programmed as separate DMA acknowledge signals. DMA TM[2:0] encodings are different from MCF5307 DMA TM[2:0] encodings.	Section 1.4, “On-Chip DMA Modifications”
	DMA byte count register (BCR) can be programmed to be 16 or 24 bits.	BCR is 24 bits only.	
UART	Both UARTs have identical functionality. No support for synchronous mode.	UART0 is identical to the MCF5307 UARTs; UART1 has been enhanced to provide synchronous operation and a CODEC interface for soft modem support.	Section 1.5, “UART Enhancements”

Table 1. Differences between MCF5307 and MCF5407 (Continued)

Feature	MCF5307	MCF5407	Reference
Timing relationships	All signal timing with respect to BCLKO; CLKIN rise time = 5 nS.	All signal timings with respect to CLKIN (BCLKO support provides compatibility with MCF5307 designs.) Tighter negative edge bus specifications due to duty cycle; CLKIN rise time = 2 nS.	Section 1.6.1, “Phase-Locked Loop (PLL),” and Section 1.6, “Timing Differences”
Reset initialization	Need to drive D[7:0]/ AA, PS[1:0], ADDR_CONFIG, FREQ[1:0], DIVIDE[1:0]	Need to drive D[7:0]/AA, PS[1:0], ADDR_CONFIG, BE_CONFIG, DIVIDE[2:0]	Section 1.7, “Reset Initialization Modifications”
Debug module	Debug Revision B. Separate PST[3:0] and DDATA[3:0]	Debug Revision C—Adds breakpoint registers, normal interrupt request service during debug, and combines debug signals into PSTDDATA[7:0]	Section 1.8, “Revision C Debug”
Voltage input changes	Drives minimum 2.4 V; accepts 5-V input	Drives minimum 2.4 V; accepts 3.3-V input	Section 1.9, “Voltage Input Changes”
	Requires 3.3-V operating voltage	Requires 1.8-V and 3.3-V operating voltages	
Pin assignment	Standard MCF5307 pinout	Compatible with MCF5307 pinout except for power-pad input assignment	Section 1.11, “Pin-Assignment Compatibility”

1.2 Instruction Set Additions

The MCF5407 implements Revision B (Rev B) of the ColdFire instruction set, which adds instructions and enhances existing ISA Revision A (Rev A) opcodes to support byte- and word-sized operands and position-independent code. Existing MCF5307 code is completely upward compatible with the MCF5407. However, designers may incorporate the instruction set additions and enhancements, especially when upgrading 68K code that references 8- and 16-bit short operands.

The following list summarizes new and enhanced instructions of Rev B ISA:

- New instructions:
  - INTOUCH loads instructions one cache block at a time for use with cache locking.
  - MOV3Q.L moves 3-bit immediate data to the destination location.
  - MVS.{B,W} moves the sign-extended source operand to the destination register.
  - MVZ.{B,W} zero-fills the source operand and moves it to the destination register.
  - SATS.L updates bit 31 of the destination register depending on the CCR overflow bit.
  - TAS.B tests and sets byte operand being addressed.
- Enhancements to existing Revision A instructions:
  - Longword support for branch instructions (Bcc, BRA, BSR)
  - Byte and word support for compare instructions (CMP, CMPI)
  - Byte and longword support for MOVE where the source is of type #<data> and the destination is of type d16(Ax); that is, move.b #<data>, d16(Ax)

Refer to Section 1.13, “ColdFire Instruction Set Architecture Enhancements,” for details of these additions and enhancements.

1.3 Enhanced Memories

With the introduction of a Harvard memory architecture in the version 4 core design, the MCF5407 has separate instruction and data caches. The 16-Kbyte instruction cache and 8-Kbyte data cache greatly improve performance on existing systems. On-chip RAMs are also provided to work with the caches.

The MCF5307 configuration contains an 8-Kbyte unified cache with a 4-Kbyte SRAM. Configuration registers for these memories include one cache control register (CACR), two access control registers (ACR0 and ACR1), and one RAM base address register (RAMBAR). With the enhanced memory sizes of the MCF5407, more configuration registers have been provided. The new MOVEC register map for the MCF5407 memory configuration registers is given in Table 2.

Table 2. MOVEC CPU Space Register Map

Rc[1:0]	Register Definition
0x002	Cache control register (CACR)
0x004	Cache access control register 0 (ACR0; data cache)
0x005	Cache access control register 1 (ACR1; data cache)
0x006	Cache access control register 2 (ACR2; instruction cache)
0x007	Cache access control register 3 (ACR3; instruction cache)
0xC04	RAM base address register 0 (RAMBAR0) [1]
0xC05	RAM base address register 1 (RAMBAR1) [1]

[1] Either or both of the RAMBAR registers can be configured for instructions or data through an additional bit, RAMBAR*n*[D/I].

Note that the existing functionality has not changed; new registers and new bits in existing registers have been added to support the enhanced memories and control for the new branch cache. One of the two 2-Kbyte SRAMs can be dedicated to support the instruction cache, and the other can support the data cache. Many designs use one SRAM block as a system stack and the other to hold important interrupt service routines.

The SRAM can also function as a ROM by programming it as a data block while loading configuration information to it and then reprogramming it as a read-only instruction block. The two MCF5407 SRAM blocks can be programmed to provide a contiguous 4-Kbyte memory map similar to the MCF5307's single contiguous 4-Kbyte SRAM.

1.4 On-Chip DMA Modifications

The MCF5407 integrates the four-channel DMA used in the MCF5307 with changes to pin multiplexing, DMA byte transfer count, and the encoding of transfer acknowledgement. The MCF5307 provides DMA acknowledgement encodings for channels 0 and 1 through the transfer modifier pins, TM[2:1], which are multiplexed with PP[4:3]. For clarification on MCF5307 signal multiplexing, see the pinout tables in the mechanical specifications chapter of the *MCF5307 User's Manual*. The MCF5307 also indicates a DMA single address access through transfer modifier pin TM0, multiplexed with PP2.

When the pin assignment register (PAR) is programmed to enable the TM signals, the encodings listed in Table 3 and Table 4 are driven during transfers by the internal DMA channels of the MCF5307. The condition TT[1:0] = 01 indicates an access by either an internal DMA or an external device.

Table 3. TM[2:1] Encoding for MCF5307 Internal DMA as Master (TT = 01)

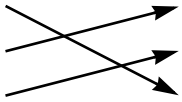
TM[2:1]	Transfer Modifier Encoding
00	DMA acknowledges negated
01	DMA acknowledge, channel 0
10	DMA acknowledge, channel 1
11	Reserved

Table 4. TM0 Encoding for MCF5307 Internal DMA as Master (TT = 01)

TM0	Transfer Modifier Encoding
0	Dual address access
1	Single address access

Although the MCF5407 provides similar encodings on TM[2:0], dedicated DMA acknowledgement pins (DACK[1:0]) have been added. Thus, DACK[1:0] are now combined with PP[3:2]/TM[1:0], resulting in a three-to-one multiplexed signal, PP[3:2]/TM[1:0]/DACK[1:0]. TM2 is still multiplexed only with PP4. For further clarification on the multiplexing, see the pinout tables in Section 1.11, "Pin-Assignment Compatibility." When properly connected, TM[2:0] can be used in MCF5407 designs as on MCF5307 designs or DACK[1:0] can be used for DMA transfers, as shown in Figure 1.

MCF5307 Function	Pin
Single/dual cycle access	TM0
DMA 0 acknowledge configuration	TM1
DMA 1 acknowledge configuration	TM2



Pin	MCF5407 Function
TM0	DMA 0 acknowledge
TM1	DMA 1 acknowledge
TM2	Single/dual cycle access

Figure 1. MCF5307 to MCF5407 TM[2:0] Pin Remapping

To enable DACK[1:0], first enable the transfer modifier signals (TM[1:0]) in the PAR and then program the interrupt assignment register (IRQPAR) in the MCF5407 SIM module to enable bits 0–1. Figure 2 defines the IRQPAR bits.

When IRQPAR[ENBDACK1] = 1 and the PAR register is also programmed to enable TM1, the DACK1 signal for DMA channel 1 is driven in place of TM1 for DMA transfers. ENBDACK1 = 0 disables this function, and only the TM1 encoding is driven. The same is true for IRQPAR[ENBDACK0]. ENBDACK0 = 1 enables DACK0 to be driven, while ENBDACK0 = 0 disables this function and drives the TM0 encoding.

	7	6	5	4	3	2	1	0
Field	IRQPAR2	IRQPAR1	IRQPAR0	Reserved			ENBDACK1	ENBDACK0
Reset	0000_0000							
R/W								
Address	Address MBAR + 0x06							

Figure 2. MCF5407 IRQPAR

Although TM[2:0] can still drive DMA access encoding, the bit positions of these encodings are different from the MCF5307. The MCF5407 encodes single-address accesses on TM2 when the PAR is set to enable the transfer modifier signal and an external master or DMA transfer is occurring. This encoding is driven by TM0 on the MCF5307.

In addition, on the MCF5407, DMA transfer acknowledgement encodings are driven on TM[1:0] (the MCF5307 uses TM[2:1]). Table 5 and Table 6 show the MCF5407 encoding for TM[2:0] when the PAR is set to enable these signals, and the IRQPAR is programmed to disable the DMA acknowledge pins DACK[1:0]. Note that when DACK[1:0] are driven, TM2 is still driven if enabled through the PAR.

Table 5. TM2 Encoding for MCF5407 Internal DMA as Master (TT = 01)

TM2	Transfer Modifier Encoding
0	Single address access negated
1	Single address access

Table 6. TM[1:0] Encoding for MCF5407 Internal DMA as Master (TT = 01)

TM[1:0]	Transfer Modifier Encoding
00	DMA acknowledges negated
01	DMA acknowledge, channel 0
10	DMA acknowledge, channel 1
11	Reserved

Table 7 summarizes MCF5407 pin configurations based on PAR and IRQPAR programming combinations.

Table 7. MCF5407 Signal Configurations for PP[4:2]/TM[2:0]/DACK[1:0]

PAR Configuration[1]	IRQPAR Configuration	PP[4:2]	TM[2:0]	DACK[1:0]
TM[2:0] disabled, PP[4:2] enabled	ENBDACK[1–0] = 0 or 1	Driven	Not driven	Not driven
TM[2:0] enabled	ENBDACK[1–0] = 0	Not driven	TM[2:0] driven	Not driven
TM[2:0] enabled	ENBDACK[1–0] = 1	Not driven	TM2 driven only	Driven

[1] Note that to enable DACK[1:0], the PAR must first be programmed to enable TM[1:0].

Designers who use MCF5307 DMA channels should also note that the MCF5407 DMA byte count registers (BCRs) for channels 0–3 exclusively support a 24-bit byte count. A 16-bit byte count register is no longer supported; therefore, MPARK[BCR24BIT] has been removed.

1.5 UART Enhancements

The MCF5407 contains two UARTs that act independently. One of the UARTs on the MCF5407 has been enhanced to provide synchronous operation and a CODEC interface for soft modem support. Each UART can be clocked by the system bus clock, eliminating the need for an external crystal.

The UART module interfaces directly to the CPU as shown in Figure 3. The UART module consists of the following major functional areas:

- Serial communication channel
- 16-bit timer for baud-rate generation
- Internal channel control logic
- Interrupt control logic

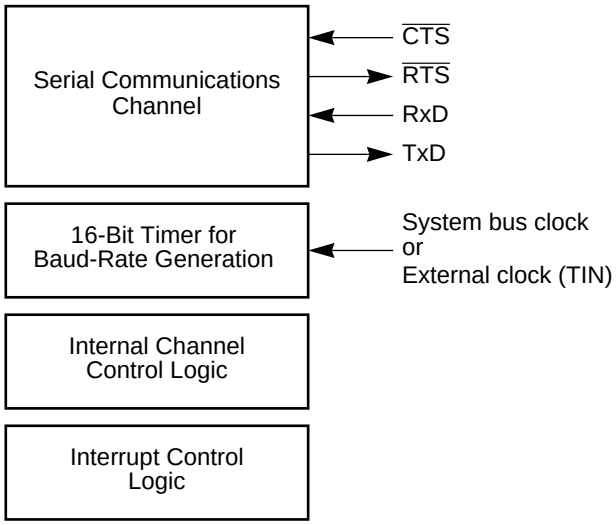


Figure 3. Simplified Block Diagram

In addition, UART1 has been enhanced to provide a CODEC interface for soft modem support. UART1 can be programmed to provide any one of the following functions:

- The original UART (identical to UART0)
- Three modem modes:
  - An 8-bit CODEC interface
  - A 16-bit CODEC interface
  - An audio CODEC 97 (AC97) digital interface controller

1.6 Timing Differences

This section explains timing relationships within phase-locked loop registers.

1.6.1 Phase-Locked Loop (PLL)

The PLL for the MCF5407 is enhanced to support faster processor clock (PCLK) frequencies. The MCF5307 supports PCLK frequencies of 66.7 and 90 MHz with a clock input (CLKIN) of 1/2 PCLK. The MCF5407 offers a larger range of clock input ratios and a higher performance processor clock.

The MCF5407 PLL module is shown in Figure 4.

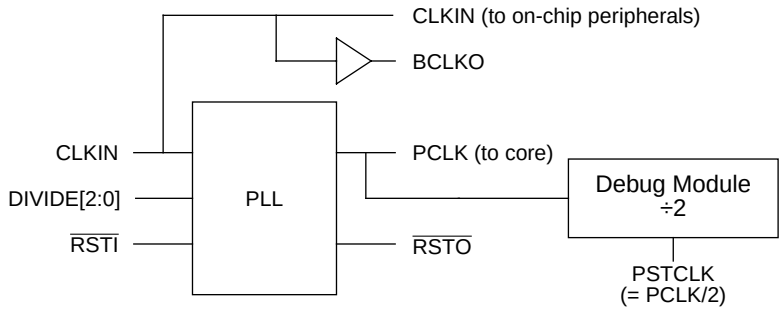


Figure 4. PLL Module

Similar to the MCF5307 functionality, the MCF5407 samples clock ratio encodings on the lower data bits of the bus at reset to determine the CLKIN-to-PCLK ratio at which the device runs. These bits are DIVIDE[1:0] on the MCF5307 and are multiplexed with data bits D[1:0]. Because the MCF5407 offers more divide ratio combinations than the MCF5307, three input bits, D[2:0]/DIVIDE[2:0], have been provided to offer more programming options at reset. Also, note that only specific CLKIN ranges are allowed for each divide ratio on the MCF5407.

Table 8 shows the new encodings. Note that they differ from the MCF5307 DIVIDE[1:0] encodings.

Table 8. Divide Ratio Encodings

D[2:0]/DIVIDE[2:0]	Input Clock (MHz)	Multiplier	Core Clock (MHz)	PSTCLK (MHz)
00x–010	Reserved			
011	40.0–54.0	3	120.0–162	60.0–81.0
100	25.0–40.5	4	100.0–162	50.0–81.0
101	25.0–32.4	5	125.0–162	67.5–81.0
110	25.0–27.0	6	150.0–162	75.0–81.0
111	Reserved			

Figure 5 correlates the CLKIN and PCLK frequencies for the 3x–6x multipliers.

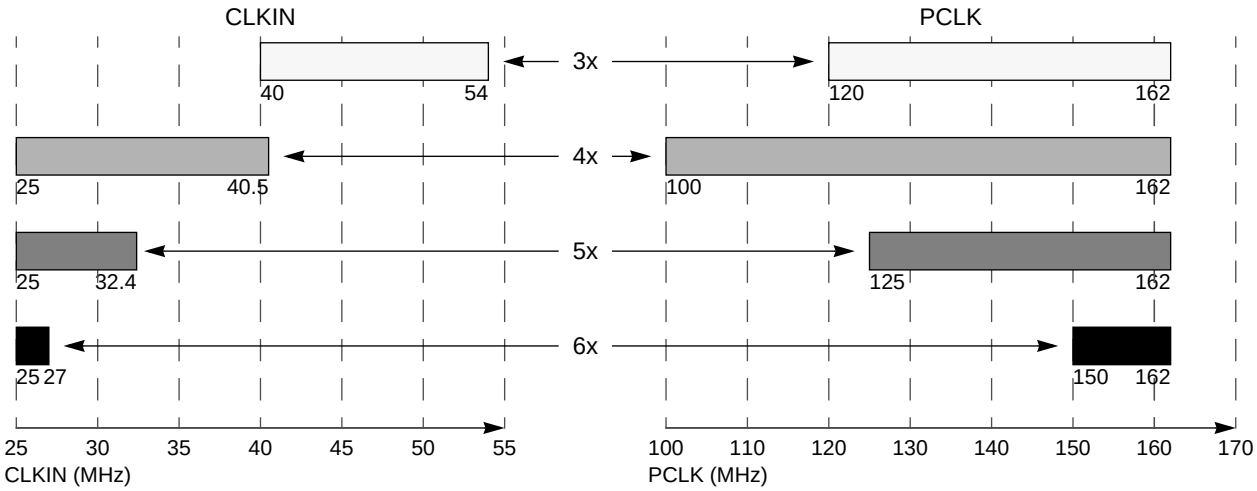


Figure 5. CLKIN-to-PCLK Frequency Ranges

1.6.2 Timing Relationships

For both the MCF5307 and MCF5407, the user provides the clock input signal (CLKIN), which is also used for on-chip peripherals, as shown in Figure 4. This signal is also the reference from which other clock frequencies are derived, including the bus clock output signal (BCLKO), which on the MCF5407 is provided for compatibility with MCF5307 designs. BCLKO is generated by the PLL and MCF5307 designs should use BCLKO as the bus timing reference for external devices; MCF5407 designs should use CLKIN. On the MCF5407, the CLKIN frequency can be 1/3, 1/4, 1/5, or 1/6 of the PCLK. Furthermore, depending on the MCF5307 configuration, the BCLKO-to-PCLK ratio may not be the same as the CLKIN-to-PCLK ratio.

On the MCF5407, the user-provided CLKIN should be used as the bus clock for the system. BCLKO runs at the same frequency as CLKIN and is offered as an optional timing reference for backwards compatibility for lower-speed MCF5307 designs.

Regardless of the CLKIN frequency driven at power-up, CLKIN and BCLKO have the same ratio value to PCLK. Although designers can use either BCLKO or CLKIN as a clock reference, recommends using CLKIN because it leaves more room to meet bus specifications than BCLKO, which is generated as a phase-aligned signal to CLKIN. An MCF5307 user should consider switching to a CLKIN reference clock when upgrading to the MCF5407 if board frequencies exceed 50 MHz.

Although the CLKIN duty cycle remains the same for the MCF5307 and MCF5407, use caution when interfacing signals on the falling edge of CLKIN with only a 4-nS window at high frequencies. Also, note that the MCF5407 input rise time is reduced to 2 nS (5 nS in the MCF5307). For designers who choose to reference signals from CLKIN only, BCLKO can be disabled to save power by setting a disable bus clock output signal (DISBCLKO) in the PLL control register (PLLCR) as shown in Figure 6.

	7	6	5	4	3	2	1	0
Field	ENBSTOP	PLLIPL2	PLLIPL1	PLLIPL0	DISBCLKO	—		
Reset	0000_0000							
R/W	R/W							
Address	Address MBAR + 0x08							

Figure 6. PLL Control Register (PLLCR)

1.7 Reset Initialization Modifications

Like the MCF5307, the MCF5407 samples a group of eight input signals, D[7:0], on the rising edge of $\overline{\text{RSTI}}$ to determine the reset configuration of the global chip select, the address bus, and PLL. However, unlike the MCF5307, the frequency range encodings are not sampled on D[3:2], which are replaced by two other reset configuration inputs. First, the CLKIN-to-PCLK ratio allows more combinations. This extra bit is now sampled on D2 so that the clock ratio programming bits encompass D[2:0]/DIVIDE[2:0].

Second, a new reset configuration bit, BE_CONFIG0, is now multiplexed with D3 in the MCF5407. This bit enables the four byte enables for the global chip select, CS0, for reads and writes or writes only, depending on the bit value sampled at reset, as shown in Table 13.

Table 9 shows the multiplexing of D[7:0] for the MCF5307 and the MCF5407.

Table 9. D[7:0] Multiplexing

Data Pins	MCF5307	MCF5407
D7	AA	
D[6:5]	PS[1:0]	
D4	ADDR_CONFIG0	
D3	FREQ1	BE_CONFIG0, BE[3:0]
D2	FREQ0	DIVIDE2
D1	DIVIDE1	
D0	DIVIDE0	

Table 10 through Table 13 list the various reset encodings for the configuration signals multiplexed with D[7:3]. See for D[2:0]/DIVIDE[2:0] encodings sampled at reset. Note that Table 10 and Table 11 configure the global, or boot, $\overline{CS0}$ that is used to access boot ROM out of reset. $\overline{CS0}$ is the only chip select active out of reset until other chip selects become valid. Both the wait states and port size of boot memory accessed by boot $\overline{CS0}$ are programmed through these bits.

Table 10. D7/AA, Automatic Acknowledge of Boot $\overline{CS0}$

D7/AA	Boot $\overline{CS0}$ AA Configuration at Reset
0	Disabled
1	Enabled with 15 wait states

Table 11 shows configurations for D[6:5]/PS[1:0].

Table 11. D[6:5]/PS[1:0], Port Size of Boot $\overline{CS0}$

D[6:5]/PS[1:0]	Boot $\overline{CS0}$ Port Size at Reset
00	32-bit port
01	8-bit port
10	16-bit port
11	16-bit port

Table 12 initializes the pin assignment register of the parallel I/O port to be either parallel I/O or to be the upper address bus bits along with various attribute and control signals at reset to give the user the option to access a broader addressing range of memory, if desired.

Table 12. D4/ADDR_CONFIG0, Address Pin Assignment

D4/ADDR_CONFIG0	Configuration Pin Assignment Register at Reset
0	PP[15:0], defaulted to inputs upon reset
1	ADDR[31:24]/$\overline{TIP}$/$\overline{DREQ}$[1:0]/TM[2:1]

Table 13 shows configurations for D3/BE_CONFIG0. Because some boot memories require byte enables to be active only during writes, the functionality of byte enables, BE[3:0], can be programmed at reset.

Table 13. D3/BE_CONFIG0, BE[3:0] Boot Configuration

D3/BE_CONFIG0	Configuration of Byte Enables for Boot CS0
0	BE[3:0] is enabled as byte write enables only
1	BE[3:0] is enabled as byte enables for reads and writes

D[2:0]/DIVIDE[2:0] configurations are shown in Table 8.

After $\overline{\text{RSTI}}$ is negated, 32 bits of CPU configuration information is loaded into data register D0 and 32 bits of internal memory information is loaded in D1. Because these registers are completely uninitialized on previous ColdFire devices, this feature allows users to identify the MCF5407 through software. Values D1 = 0x0630_0530 and D0 = 0xCF4*x*_C012 identify the MCF5407, where *x* identifies the core revision number (0x1 for the initial device).

1.8 Revision C Debug

A number of enhancements to the original ColdFire debug functions were requested by customers and third-party tool developers. As a result, an expanded set of debug functions was implemented in the version 4 ColdFire and named Revision C, or simply Debug C. Most of the enhancements are included in the MCF5407 debug module and are primarily related to improvements in the real-time debug capabilities.

1.8.1 Debug Interrupts and Interrupt Requests in Emulator Mode

In the Debug B ColdFire implementation of the MCF5307, the response to a user-defined breakpoint trigger can be configured as one of three possibilities:

- The breakpoint trigger can be displayed on the DDATA bus with no internal reaction to the trigger. The trigger state information is displayed on DDATA in all situations.
- The breakpoint trigger can force the processor to halt and allow BDM activities.
- The breakpoint trigger can generate a special debug interrupt to allow real-time systems to quickly process the interrupt quickly and return to normal system executing as rapidly as possible.

The occurrence of a debug interrupt exception is treated as a special type of interrupt. It is considered to be higher in priority than all normal interrupt requests and has special processor status values to indicate externally that this interrupt occurred.

Additionally, the execution of the debug interrupt service routine is forced to be interrupt-inhibited by the processor hardware. Optionally, it is capable of mapping all instruction and data references while in this service routine into a separate address space, so that an emulator can define the routine dynamically.

Current processor implementations include a state bit, invisible to software, that defines this emulator mode of operation. Note that the interrupt mask level is not modified during the processing of a debug interrupt.

In response to customers with real-time embedded systems asking for the ability to service normal interrupt requests while processing the debug interrupt service routine, this feature has been incorporated in the Revision C debug. To provide this function and service any number of normal interrupt requests, including the possibility of nested interrupts, the processor state signaling emulator mode is now included as part of the exception stack frame, shown in Figure 7.

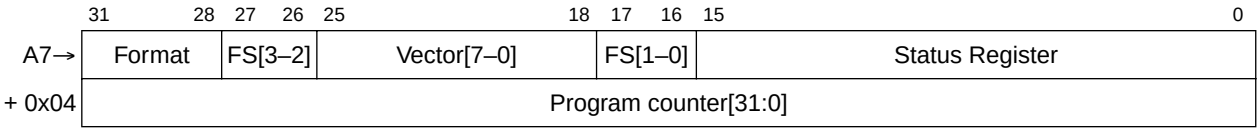


Figure 7. Exception Stack Frame Form

As part of the Debug C enhancement, the operation of the debug interrupt is modified as follows:

- The occurrence of the breakpoint trigger, configured to generate a debug interrupt, is treated exactly as before. The debug interrupt is treated as a higher priority exception relative to the normal interrupt requests encoded on the interrupt priority input signals.
- At the appropriate sample point, the processor initiates debug interrupt exception processing. This event is signaled externally by the generation of a unique PST value (PST = 0xD) asserted for multiple cycles. The processor sets the emulator mode state bit as part of this processing.
- All normal interrupt requests are evaluated and sampled once per instruction during the debug interrupt service routine. If an exception is detected, the processor takes the following steps:
  1. In response to the new exception, the processor saves a copy of the current value of the emulator mode state bit and then exits emulator mode by clearing the actual state.
  2. The new exception stack frame sets bit 1 of the fault status field, using the saved emulator mode bit, indicating that execution while the processor is in emulator mode was interrupted. This corresponds to bit [17] of the longword at the top of the system stack.
  3. Control is passed to the appropriate exception handler.
  4. When the exception handler is complete, a Return From Exception (RTE) instruction is executed. During the processing of the RTE, the FS1 bit is reloaded from the system stack. If FS1 = 1, the processor sets the emulator mode state and resumes execution of the original debug interrupt service routine. This is signaled externally by the generation of the PST value that originally identified the occurrence of a debug interrupt exception, that is, PST = 0xD.

Implementation of this revised debug interrupt handling fully supports the servicing of any number of normal interrupt requests while in a debug interrupt service routine. The emulator mode state bit is essentially changed to a program-visible value, stored into memory when the exception stack frame is created, and loaded from memory by the RTE instruction.

1.8.2 On-Chip Breakpoint Registers

The Debug B core debug module included three basic types of on-chip breakpoint registers:

- A 32-bit PC breakpoint register and a 32-bit PC breakpoint mask
- Two 32-bit address registers, which can be used to specify a single address or a range of addresses
- A 32-bit data breakpoint register and a 32-bit data breakpoint mask

The mask registers can be used to “don’t care” the equivalent bits in the breakpoint registers.

Additions to the breakpoint implementation are as follows:

- Three more 32-bit PC breakpoint registers
- Two more 32-bit address registers (ABLR1, ABHR1) plus an attribute register (AATR1) and mask register, which can be used to specify a single address or a range of addresses
- One more 32-bit data breakpoint register and a 32-bit data breakpoint mask

The addition of these new breakpoint registers also requires the appropriate control and configuration functions be added to the debug programming model. The affected BDM command and new register formats are described below. The revised BDM command is write debug module register (WDMREG).

1.8.2.1 Write Debug Module Register (WDMREG)

The operand (longword) data is written to the specified debug module register. All 32 bits of the register are altered by the write operation. The debug module's programming model can be accessed either from the serial BDM communication channel or from the processor's execution of the supervisor-mode WDEBUG instruction. DSCLK must be inactive while WDEBUG executes.

Figure 8 defines the operand data format.

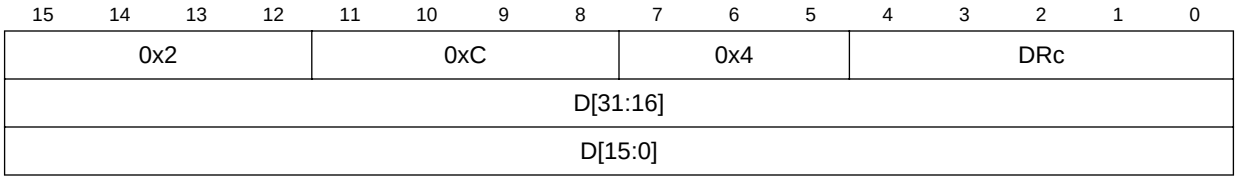


Figure 8. Write Debug Module Register Command (WDMREG)

Table 14 describes the DRc encoding for the debug registers.

Table 14. Definition of DRc Encoding—Write

DRc (hex)	Debug Register Definition	Abbreviation	Initial State (hex)
0x00	Configuration/Status	CSR	0x0000
0x01–0x04	Reserved	—	—
0x05	BDM address attributes	BAAR	0x0005
0x06	Bus attributes and mask	AATR	0x0005
0x07	Trigger definition	TDR	0x0000
0x08	PC breakpoint	PBR	—
0x09	PC breakpoint mask	PBMR	—
0x0A–0x0B	Reserved	—	—
0x0C	Operand address high breakpoint	ABHR	—
0x0D	Operand address low breakpoint	ABLR	—
0x0E	Data breakpoint	DBR	—
0x0F	Data breakpoint mask	DBMR	—
0x10–0x15	Reserved	—	—
0x16	Bus attributes and mask 1	AATR1	0x0005
0x17	Extended trigger definition	XTDR	0x0000
0x18	PC breakpoint 1	PBR1	0x0000
0x19	Reserved	—	—
0x1A	PC breakpoint 2	PBR2	0x0000
0x1B	PC breakpoint 3	PBR3	0x0000
0x1C	Operand address high breakpoint 1	ABHR1	—
0x1D	Operand address low breakpoint 1	ABLR1	—

Table 14. Definition of DRc Encoding—Write (Continued)

DRc (hex)	Debug Register Definition	Abbreviation	Initial State (hex)
0x1E	Data breakpoint 1	DBR1	—
0x1F	Data breakpoint mask 1	DBMR1	—

Command Sequence:

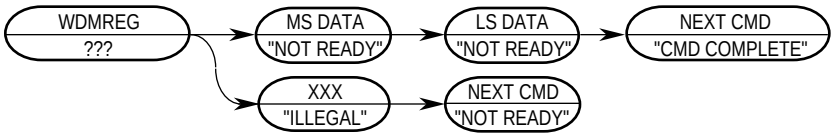


Figure 9. WDMREG Command Sequence

Operand Data:

Longword data is written into the specified debug register. Data is supplied most significant word first.

Result Data:

Command complete status (0x0FFFF) is returned when register write is complete.

1.8.3 Debug Programming Model

In addition to existing BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains a number of registers to support the required functionality. These registers are treated as 32-bit quantities, regardless of the number of bits in the implementation. The debug control registers (DRc) are addressed using a 5-bit value as part of two new BDM commands (WDREG and RDREG). These values are shown in Table 14.

These registers are also accessible from the processor's supervisor programming model through the execution of the WDEBUG instruction. Thus, the breakpoint hardware within the debug module can be accessed by the external development system using the serial interface or by the operating system running on the processor core. It is the software's responsibility to guarantee that all accesses to these resources are serialized and are logically consistent. The hardware provides a locking mechanism in the CSR to allow the external development system to disable any attempted writes by the processor to the breakpoint registers (setting IPW).

The following sections describe the newly added breakpoint registers in Debug C.

1.8.3.1 Address Breakpoint 1 Registers (ABLR1, ABHR1)

The 32-bit address breakpoint 1 registers define an upper (ABHR1) and a lower (ABLR1) boundary for a region in the operand logical address space of the processor that can be used as part of the trigger. The ABLR1 and ABHR1 values are compared with the ColdFire CPU core address signals, as defined by the setting of the trigger definition register (TDR) and the extended trigger definition register (XTDR).

1.8.3.2 Address Attribute Breakpoint Register 1 (AATR1)

The address attribute breakpoint register 1 (AATR1) defines the address attributes and a mask associated with ABLR1 and ABHR1 to be matched in the trigger. The AATR1 value is compared with the ColdFire CPU core address attribute signals, as defined by the setting of the TDR and XTDR. The format of the AATR1 is the same as the AATR register, as shown in Figure 10.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	RM	SZM		TTM		TMM			R	SZ		TT		TM		
Reset	0000_0000_0000_0101															
R/W	Write only															
DRc[4–0]	0x16 (AATR1)															

Figure 10. Address Attribute Trigger Registers (AATR, AATR1)

1.8.3.3 Program Counter Breakpoint Registers 1–3 (PBR1–PBR3)

Each of the program counter (PC) breakpoint registers (PBR, PBR1–PBR3), shown in Figure 11, defines an instruction address that can be used as part of the trigger. PBR*n* registers are compared with the processor's program counter register when the appropriate valid bit is asserted and TDR is configured appropriately.

	31	1	0
Field	Program Counter		Valid
Reset	—		0
R/W	Write		
DRc[4–0]	0x08 (PBR); 0x18 (PBR1); 0x1A (PBR2); 0x1B (PBR3)		

Figure 11. Program Counter Breakpoint Registers (PBR, PBR1, PBR2, PBR3)

The results of all PC breakpoint registers, PBR/PBMR, PBR1, PBR2, and PBR3, are logically summed to form a single PC breakpoint trigger signal.

- PBR*n*[31:1] = program counter breakpoint address
- PBR*n*[0] = valid bit

1.8.3.4 Data Breakpoint Register 1 (DBR1, DBMR1)

The data breakpoint register 1 (DBR1) defines a specific data pattern that can be used as part of a trigger. The DBR1 value is masked by DBMR1, allowing only those bits in DBR1 that have a corresponding zero in DBMR1 to be compared with the ColdFire CPU core data signals, as defined in the TDR and the XTDR.

The data breakpoint registers support both aligned and misaligned operand references. The relationship between the processor core address, the access size, and the corresponding location within the 32-bit core data bus is defined in the DBR and DBMR description.

1.8.3.5 Extended Trigger Definition Register (XTDR)

The XTDR enables the operation as defined by the new breakpoint registers, ABHR1, ABLR1, AATR1, DBR1, and DBMR1, within the debug module and operates in conjunction with the trigger definition register (TDR). The added breakpoint logic can be included as a one- or two-level trigger; XTDR[29–18] define second-level triggers and XTDR[13–2] define first-level triggers, as shown in Figure 12.

The definition of the XTDR register is exactly the same as the TDR for the control of the ABHR1, ABLR1, DBR1 and DBMR1 breakpoint registers. The XTDR is cleared on reset.

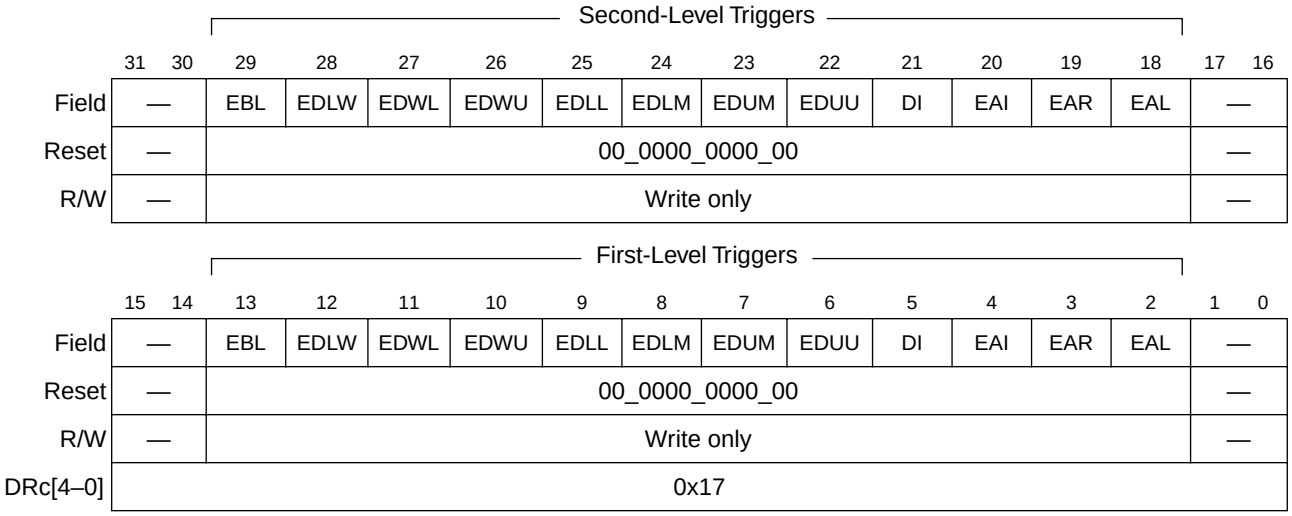


Figure 12. Extended Trigger Definition Register (XTDR)

Table 15 describes XTDR fields.

Table 15. XTDR Field Descriptions

Bits	Name	Description
29/13	EBL	Enable breakpoint level 2/enable breakpoint level 1. If set, EBL serves as the global enable for the breakpoint trigger; that is, if TDR[EBL] or XTDR[EBL] is set, a breakpoint trigger is enabled. If TDR[EBL] and XTDR[EBL] are cleared, all breakpoints are disabled.
28–22/ 12–6	ED*x*	Setting an ED bit enables the corresponding data breakpoint condition. Clearing all bits disables the data breakpoint.

Bits	Name	Breakpoint Condition
28/12	EDL	Data longword (entire data bus)
27/11	EDWL	Lower data word (low-order word)
26/10	EDWU	Upper data word (high-order word)
25/9	EDLL	Lower lower data byte (low-order byte of the low-order word)
24/8	EDLM	Lower middle data byte (high-order byte of the low-order word)
23/7	EDUM	Upper middle data byte (low-order byte of the high-order word)
22/6	EDUU	Upper upper data byte (high-order byte of the high-order word)

21/5	DI	Data breakpoint invert. This bit provides a mechanism to invert the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value not equal to the one programmed into the DBR1.
20–18/ 4–2	EA*x*	Enable address bits. Setting an EA bit enables the corresponding address breakpoint. If all three bits are cleared, this breakpoint is disabled.

Bits	Name	Breakpoint Condition
20/4	EAI	Address breakpoint inverted. Range defined by ABLR1 and ABHR1.
19/3	EAR	Address breakpoint range. Inclusive range defined by ABLR1 and ABHR1.
18/2	EAL	Address breakpoint low. Address contained in ABLR1.

The resulting set of possible breakpoint trigger combinations consists of the following options where || denotes logical OR, && denotes logical AND, and {} denotes an optional additional trigger term:

```
One-level triggers of the form:

if      (PC_breakpoint)
if      (PC_breakpoint||Address_breakpoint{&&Data_breakpoint})
if      (PC_breakpoint||Address_breakpoint{&&Data_breakpoint}
                ||      Address1_breakpoint{&&Data1_breakpoint})

if      (Address_breakpoint     {&&     Data_breakpoint})
if      ((Address_breakpoint    {&&     Data_breakpoint})
                ||      (Address1_breakpoint{&&Data1_breakpoint}))

if      (Address1_breakpoint    {&&     Data1_breakpoint})


Two-level triggers of the form:

if      (PC_breakpoint)
        then if          (Address_breakpoint{&&Data_breakpoint})

if      (PC_breakpoint)
        then if          (Address_breakpoint{&&Data_breakpoint}
                ||      Address1_breakpoint{&&Data1_breakpoint})

if      (PC_breakpoint)
        then if         (Address1_breakpoint{&&Data1_breakpoint})

if      (Address_breakpoint     {&&     Data_breakpoint})
        then if         (Address1_breakpoint{&&Data1_breakpoint})

if      (Address1_breakpoint    {&&     Data1_breakpoint})
        then if         (Address_breakpoint{&&Data_breakpoint})

if      (Address_breakpoint     {&&     Data_breakpoint})
        then if         (PC_breakpoint)

if      (Address1_breakpoint    {&&     Data1_breakpoint})
        then if         (PC_breakpoint)

if      (Address_breakpoint     {&&     Data_breakpoint})
        then if         (PC_breakpoint
                ||      Address1_breakpoint{&&Data1_breakpoint})

if      (Address1_breakpoint    {&&     Data1_breakpoint})
        then if         (PC_breakpoint
                ||      Address_breakpoint{&&Data_breakpoint})
```

In this example, PC_breakpoint is the logical summation of the PBR/PBMR, PBR1, PBR2, and PBR3 breakpoint registers; Address_breakpoint is a function of ABHR, ABLR, and AATR; Data_breakpoint is a function of DBR and DBMR; Address1_breakpoint is a function of ABHR1, ABLR1, and AATR1; and Data1_breakpoint is a function of DBR1 and DBMR1. In all cases, the data breakpoints can be included with an address breakpoint to further qualify a trigger event as an option.

1.8.4 Debug Interrupt Exception Vectors

In the Debug B revision, if the occurrence of a hardware breakpoint is configured to generate a debug interrupt, this exception is mapped to vector number 12 (0x030). The actual debug interrupts can be broadly classified into two groups—PC breakpoints and all other types. A PC breakpoint is treated in a precise manner—exception recognition and processing are initiated before the instruction at the given address is

executed. Conversely, all other breakpoint events are recognized on the given internal bus transaction, but are made pending to the processor and sampled like other interrupt conditions. As a result, these types of interrupts are imprecise by nature.

In response to a customer request that PC breakpoints be distinguishable from other type of trigger events, the debug interrupt exception vector is expanded in Debug C of the MCF5407 to two unique entries, shown in Table 16, where the occurrence of a PC breakpoint generates the 0x034 vector. In the case of a two-level trigger, the last breakpoint event determines the exception vector.

Table 16. Debug C Exception Vector Assignments

Vector	Vector Offset	Stacked Program Counter	Assignment
12	0x030	Next	Non-PC-breakpoint debug interrupt
13	0x034	Next	PC-breakpoint debug Interrupt

1.8.5 Processor Status and Debug Data Output Signals

The Debug B architecture defines processor status, PST[3:0] and debug data DDATA[3:0] signals, which provide information to support real-time trace. In the Debug B design, these signals are output at the processor frequency.

For the Debug C definition, however, the PST and DDATA are combined and redefined to operate at half the processor's operating frequency (provided by PSTCLK). Therefore, PSTDDATA[7:0] are used to output both processor status and captured debug data values.

The example in Table 17 shows PSTDDATA output for the sequential execution of single-cycle instructions A–F.

Table 17. PSTDDATA Behavior: Sequential Execution of Single-Cycle Instructions

Cycle	PSTDDATA[7:0]
T (T = processor clock period)	{PST for A, PST for B}
T+1	{PST for A, PST for B}
T+2	{PST for C, PST for D}
T+3	{PST for C, PST for D}
T+4	{PST for E, PST for F}
T+5	{PST for E, PST for F}

Cycle counts are relative to processor frequency. Consider the case where the DDATA module captures an operand on a simple load instruction, mov.l <mem>,Rx. Table 18 shows the PSTDDATA output bus for this case.

Table 18. PSTDDATA Behavior: Data Operand Captured

Cycle	PSTDDATA[7:0]
T (T = processor clock period)	{PST for mov.l, PST marker for captured operand} = {0x1, 0xB}
T+1	{0x1, 0xB}
T+2	{Operand[3:0], Operand[7:4]}
T+3	{Operand[3:0], Operand[7:4]}

Table 18. PSTDDATA Behavior: Data Operand Captured (Continued)

Cycle	PSTDDATA[7:0]
T+4	{Operand[11:8], Operand[15:12]}
T+5	{Operand[11:8], Operand[15:12]}
T+6	{Operand[19:16], Operand[23:20]}
T+7	{Operand[19:16], Operand[23:20]}
T+8	{Operand[27:24], Operand[31:28]}
T+9	{Operand[27:24], Operand[31:28]}
T+10	(PST for next instruction A fin..}
T+11	(PST for next instruction, ...}

The PST marker and the accompanying data display are guaranteed to be sent contiguously. Except for this transmission, the IDLE status (0x0) may appear at any time. There are no alignment restrictions; that is, the PST values and operands may appear on either nibble of the PSTDDATA output bus. Given that the real-time trace information appears as a sequence of 4-bit data values, the upper nibble of the output, PSTDDATA[7:4], is considered more significant, that is, occurring before the lower nibble.

In Debug B, the DDATA outputs display the status of the internal breakpoint registers when they are not displaying captured data values. For the Debug C design, any change to this breakpoint state is identified by a PST marker and then the new state value. Specifically, the marker for this breakpoint state change is a single assertion of the value 0xD. Usually, the 0xD status is asserted for multiple cycles, indicating entry into emulator mode in response to a debug interrupt exception. For Debug C, the posting of the 0xD status can signal multiple events, based on the next value.

```
if the PSTDDATA stream includes {0xD, 0x2}
        then Breakpoint state changed to Waiting for Level 1 Trigger

if the PSTDDATA stream includes {0xD, 0x4}
        then Breakpoint state changed to Level 1 Breakpoint Triggered

if the PSTDDATA stream includes {0xD, 0xA}
        then Breakpoint state changed to Waiting for Level 2 Trigger

if the PSTDDATA stream includes {0xD, 0xC}
        then Breakpoint state changed to Level 2 Breakpoint Triggered

if the PSTDDATA stream includes {0xD, 0xD}
        then Entry into Emulator Mode
```

Table 19 shows the revised definition of the processor status encodings, where the values of {0xC–0xF} are usually asserted for multiple cycles. The behavior of the 0xD value was described previously. The PSTDDATA values of 0x2 and 0x6 are formerly reserved values now needed to support the Version 4 Operand Execution Pipeline.

Table 19. Version 4 Debug C Processor Status Encodings

PSTDDATA Value	Definition
0x0	Continue execution
0x1	Begin execution of one instruction
0x2	Begin execution of two instructions

Table 19. Version 4 Debug C Processor Status Encodings (Continued)

PSTDDATA Value	Definition
0x3	Entry into user-mode
0x4	Begin execution of PULSE or WDDATA instruction
0x5	Begin execution of taken branch
0x6	Begin execution of an instruction plus a taken branch
0x7	Begin execution of RTE instruction
0x8	Begin 1-byte data transfer on PSTDDATA
0x9	Begin 2-byte data transfer on PSTDDATA
0xA	Begin 3-byte data transfer on PSTDDATA
0xB	Begin 4-byte data transfer on PSTDDATA
0xC	Exception processing
0xD	Breakpoint state change, or entry into emulator mode
0xE	Processor is stopped, waiting for interrupt
0xF	Processor is halted

1.8.6 Debug C Summary

The preceding section describes additional functionality requested by ColdFire customers and third-party developers. The Debug C enhancements are designed to retain backward compatibility with the previous definition.

1.9 Voltage Input Changes

The MCF5407 is manufactured in a 0.22-μ quad-layer-metal (QLM) technology. Whereas the MCF5407 logic operates at 1.8 V, the device pads are standard TTL-compatible and therefore can drive a 2.4-V minimum output and accepts a 3.3-V input. Thus, the MCF5407 requires both 1.8- and 3.3-V power supplies. This specification differs from the MCF5307, which operates at 3.3 V with 5-V-tolerant I/O pads. Although the power and ground pin assignment are the same for both the MCF5307 and MCF5407, the power pin allocation of the MCF5407 is divided between 1.8- and 3.3-V supply levels. Thus, two power rails are necessary to supply power to the MCF5407.

Note that the MCF5407 meets the EIA/JEDEC standard for 1.8-V power supply voltage and interface requirements. See the JEDEC standard (EIA/JESD8-7, February 1997).

1.10 PLL Power Supply Filter Circuit

To ensure PLL stability, the power supply to the PLL power pin should be filtered using a circuit similar to the one shown in Figure 13. The circuit should be as close as possible to the PLL power pin to ensure maximum noise filtering. This filter design can be used for both the MCF5307 and MCF5407.

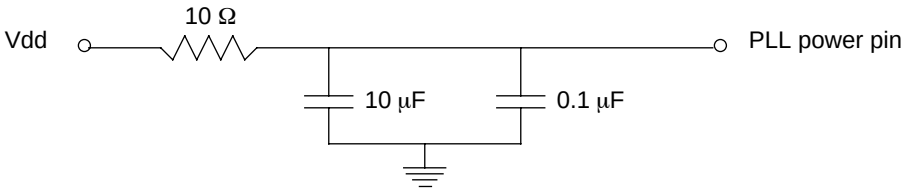


Figure 13. PLL Power Supply Filter Circuit

1.11 Pin-Assignment Compatibility

The MCF5407 pinout is identical to the MCF5307 except for the power pin allocation and PSTDDATA[7:0], which make available the contents of processor status, PST[3:0], and debug data, DDATA[3:0]. Therefore, when designing-in the MCF5407, note which power pins require 1.8 V and which require 3.3 V. The MCF5407 footprint is the same as the MCF5307, which is a 208-pin plastic quad flat pack (QFP).

1.12 Mechanical Data

This section provides a function pin listing and package diagram for the MCF5407. See URL

1.12.1 Package

The MCF5407 is assembled in a 208-pin, thermally enhanced plastic QFP package.

1.12.2 Pinout

The MCF5407 pinout is described in Table 20 through Table 23, including signal multiplexing. Additional columns indicate the direction, description, and output drive capability of each pin.

Table 20. Pins 1–52 (Left, Top-to-Bottom)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
1	IVCC	—	—	1.8-V power input	—
2	A0	—	I/O	Address bus bit	8
3	A1	—	I/O	Address bus bit	8
4	GND	—	—	Ground pin	—
5	A2	—	I/O	Address bus bit	8
6	A3	—	I/O	Address bus bit	8
7	EVCC	—	—	3.3-V power input	—
8	A4	—	I/O	Address bus bit	8
9	A5	—	I/O	Address bus bit	8
10	GND	—	—	Ground pin	—
11	A6	—	I/O	Address bus bit	8

Table 20. Pins 1–52 (Left, Top-to-Bottom) (Continued)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
12	A7	—	I/O	Address bus bit	8
13	EVCC	—	—	3.3-V power input	—
14	A8	—	I/O	Address bus bit	8
15	A9	—	I/O	Address bus bit	8
16	A10	—	I/O	Address bus bit	8
17	GND	—	—	Ground pin	—
18	A11	—	I/O	Address bus bit	8
19	A12	—	I/O	Address bus bit	8
20	A13	—	I/O	Address bus bit	8
21	EVCC	—	—	3.3-V power input	—
22	A14	—	I/O	Address bus bit	8
23	A15	—	I/O	Address bus bit	8
24	A16	—	I/O	Address bus bit	8
25	GND	—	—	Ground pin	—
26	A17	—	I/O	Address bus bit	8
27	A18	—	I/O	Address bus bit	8
28	A19	—	I/O	Address bus bit	8
29	EVCC	—	—	3.3-V power input	—
30	A20	—	I/O	Address bus bit	8
31	A21	—	I/O	Address bus bit	8
32	A22	—	I/O	Address bus bit	8
33	GND	—	—	Ground pin	—
34	A23	—	I/O	Address bus bit	8
35	PP8	A24	I/O	Parallel port bit/Address bus bit	8
36	PP9	A25	I/O	Parallel port bit/Address bus bit	8
37	EVCC	—	—	3.3-V power input	—
38	PP10	A26	I/O	Parallel port bit/Address bus bit	8
39	PP11	A27	I/O	Parallel port bit/Address bus bit	8
40	PP12	A28	I/O	Parallel port bit/Address bus bit	8
41	GND	—	—	Ground pin	—
42	PP13	A29	I/O	Parallel port bit/Address bus bit	8
43	PP14	A30	I/O	Parallel port bit/Address bus bit	8
44	PP15	A31	I/O	Parallel port bit/Address bus bit	8
45	EVCC	—	—	3.3-V power input	—

Table 20. Pins 1–52 (Left, Top-to-Bottom) (Continued)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
46	SIZ0	—	I/O	Size attribute	8
47	SIZ1	—	I/O	Size attribute	8
48	GND	—	—	Ground pin	—
49	$\overline{OE}$	—	O	Output enable for chip selects	8
50	$\overline{CS0}$	—	O	Chip select	8
51	$\overline{CS1}$	—	O	Chip select	8
52	EVCC	—	—	3.3-V power input	—

Table 21. Pins 53–104 (Bottom, Left-to-Right)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
53	GND	—	—	Ground pin	—
54	$\overline{CS2}$	—	O	Chip select	8
55	$\overline{CS3}$	—	O	Chip select	8
56	$\overline{CS4}$	—	O	Chip select	8
57	IVCC	—	—	1.8-V power input	—
58	$\overline{CS5}$	—	O	Chip select	8
59	$\overline{CS6}$	—	O	Chip select	8
60	$\overline{CS7}$	—	O	Chip select	8
61	GND	—	—	Ground pin	—
62	$\overline{AS}$	—	I/O	Address strobe	8
63	R/$\overline{W}$	—	I/O	Read/$\overline{Write}$	8
64	$\overline{TA}$	—	I/O	Transfer acknowledge	8
65	EVCC	—	—	3.3-V power input	—
66	$\overline{TS}$	—	I/O	Transfer start	8
67	$\overline{RSTI}$	—	I	Reset	—
68	$\overline{IRQ7}$	—	I	Interrupt request	—
69	GND	—	—	Ground pin	—
70	$\overline{IRQ5}$	IRQ4	I	Interrupt request	—
71	$\overline{IRQ3}$	IRQ6	I	Interrupt request	—
72	$\overline{IRQ1}$	IRQ2	I	Interrupt request	—
73	IVCC	—	—	1.8-V power input	—
74	$\overline{BR}$	—	O	Bus request	8
75	$\overline{BD}$	—	O	Bus driven	8

Table 21. Pins 53–104 (Bottom, Left-to-Right) (Continued)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
76	$\overline{BG}$	—	I	Bus grant	—
77	GND	—	—	Ground pin	—
78	TOUT1	—	O	Timer output	8
79	TOUT0	—	O	Timer output	8
80	TIN0	—	I	Timer input	—
81	EVCC	—	—	3.3-V power input	—
82	TIN1	—	I	Timer input	—
83	$\overline{RAS0}$	—	O	DRAM row address strobe	16
84	$\overline{RAS1}$	—	O	DRAM row address strobe	16
85	GND	—	—	Ground pin	—
86	$\overline{CAS0}$	—	O	DRAM column address strobe	16
87	$\overline{CAS1}$	—	O	DRAM column address strobe	16
88	$\overline{CAS2}$	—	O	DRAM column address strobe	16
89	EVCC	—	—	3.3-V power input	—
90	$\overline{CAS3}$	—	O	DRAM column address strobe	16
91	$\overline{DRAMW}$	—	O	DRAM write	16
92	$\overline{SRAS}$	—	O	SDRAM row address strobe	16
93	GND	—	—	Ground pin	—
94	$\overline{SCAS}$	—	O	SDRAM column address strobe	16
95	SCKE	—	O	SDRAM clock enable	16
96	$\overline{BE0}$	BWE0	O	Byte enable/byte write enable	8
97	EVCC	—	—	3.3-V power input	—
98	$\overline{BE1}$	BWE1	O	Byte enable/byte write enable	8
99	$\overline{BE2}$	BWE2	O	Byte enable/byte write enable	8
100	$\overline{BE3}$	BWE3	O	Byte enable/byte write enable	8
101	GND	—	—	Ground pin	—
102	SCL	—	I/OD [1]	Serial clock line	8
103	SDA	—	I/OD [1]	Serial data line	8
104	GND	—	—	Ground pin	—

[1] OD: Open-drain output

Table 22. Pins 105–156 (Right, Bottom-to-Top)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
105	IVCC	—	—	1.8-V power input	—
106	D31	—	I/O	Data bus	8
107	D30	—	I/O	Data bus	8
108	D29	—	I/O	Data bus	8
109	GND	—	—	Ground pin	—
110	D28	—	I/O	Data bus	8
111	D27	—	I/O	Data bus	8
112	D26	—	I/O	Data bus	8
113	EVCC	—	—	3.3-V power input	—
114	D25	—	I/O	Data bus	8
115	D24	—	I/O	Data bus	8
116	D23	—	I/O	Data bus	8
117	GND	—	—	Ground pin	—
118	D22	—	I/O	Data bus	8
119	D21	—	I/O	Data bus	8
120	D20	—	I./O	Data bus	8
121	EVCC	—	—	3.3-V power input	—
122	D19	—	I/O	Data bus	8
123	D18	—	I/O	Data bus	8
124	D17	—	I/O	Data bus	8
125	GND	—	—	Ground pin	—
126	D16	—	I/O	Data bus	8
127	D15	—	I/O	Data bus	8
128	D14	—	I/O	Data bus	8
129	EVCC	—	—	3.3-V power input	—
130	D13	—	I/O	Data bus	8
131	D12	—	I/O	Data bus	8
132	D11	—	I/O	Data bus	8
133	GND	—	—	Ground pin	—
134	D10	—	I/O	Data bus	8
135	D9	—	I/O	Data bus	8
136	D8	—	I/O	Data bus	8
137	EVCC	—	—	3.3-V power input	—
138	D7	CS_CONF2	I/O	Data bus/Chip select configuration	8

Table 22. Pins 105–156 (Right, Bottom-to-Top) (Continued)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
139	D6	CS_CONF1	I/O	Data bus/Chip select configuration	8
140	D5	CS_CONF0	I/O	Data bus/Chip select configuration	8
141	GND	—	—	Ground pin	—
142	D4	ADDR_CONF	I/O	Data bus/Address configuration	8
143	D3	BE_CONFIG0	I/O	Data bus/Byte enable configuration	8
144	D2	DIVIDE2	I/O	Data bus/Divide control PCLK:CLKIN	8
145	EVCC	—	—	3.3-V power input	—
146	D1	DIVIDE1	I/O	Data bus/Divide control PCLK:CLKIN	8
147	D0	DIVIDE0	I/O	Data bus/Divide control PCLK:CLKIN	8
148	GND	—	—	Ground pin	—
149	DSCLK	$\overline{\text{TRST}}$	I	Debug serial clock/JTAG Reset	—
150	TCK	TCK	I	JTAG clock	—
151	DSO	TDO	O	Debug serial out/JTAG data out	8
152	IVCC	—	—	1.8-V power input	—
153	DSI	TDI	I	Debug serial input/JTAG data in	—
154	$\overline{\text{BKPT}}$	TMS	I	Debug breakpoint/JTAG mode select	—
155	$\overline{\text{HIZ}}$	—	I	High impedance override	—
156	GND	—	—	Ground pin	—

Table 23. Pins 157–208 (Top, Right-to-Left)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
157	IVCC	—	—	1.8-V power input	—
158	CTS1	—	I	UART1 clear-to-send	—
159	RTS1	—	O	UART1 request-to-send	8
160	RXD1	—	I	UART1 receive data	—
161	TXD1	—	O	UART1 transmit data	8
162	GND	—	—	Ground pin	—
163	CTS0	—	I	UART0 clear-to-send	—
164	RTS0	—	O	UART0 request-to-send	8
165	RXD0	—	I	UART0 receive data	—
166	TXD0	—	O	UART0 transmit data	8
167	EVCC	—	—	3.3-V power input	—
168	EDGESEL	—	I	SDRAM bus clock edge select	—

Table 23. Pins 157–208 (Top, Right-to-Left) (Continued)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
169	GND	—	—	Ground pin	—
170	BCLKO	—	O	Bus clock output	16
171	IVCC	—	—	1.8-V power input	—
172	$\overline{\text{RSTO}}$	—	O	Processor reset output	8
173	GND	—	—	Ground pin	—
174	CLKIN	—	I	System bus clock input	—
175	IVCC	—	—	1.8-V power input	—
176	MTMOD0	—	I	JTAG/BDM select (Tie high or low)	—
177	MTMOD1	—	I	Tie high or low	—
178	PGND	—	—	PLL ground pin	—
179	NC	—	O	No connect	—
180	PVCC	—	—	1.8-V filter supply for PLL	—
181	MTMOD2	—	I	Tie high or low	—
182	MTMOD3	—	I	Tie high or low	—
183	GND	—	—	Ground pin	—
184	PSTCLK	—	O	Processor status clock	8
185	IVCC	—	—	1.8-V power input	—
186	PSTDDATA0	—	O	Processor status/debug data	8
187	PSTDDATA1	—	O	Processor status/debug data	8
188	GND	—	—	Ground pin	—
189	PSTDDATA2	—	O	Processor status/debug data	8
190	PSTDDATA3	—	O	Processor status/debug data	8
191	EVCC	—	—	3.3-V power input	—
192	PSTDDATA4	—	O	Processor status/debug data	8
193	PSTDDATA5	—	O	Processor status/debug data	8
194	GND	—	—	Ground pin	—
195	PSTDDATA6	—	O	Processor status/debug data	8
196	PSTDDATA7	—	O	Processor status/debug data	8
197	IVCC	—	—	1.8-V power input	—
198	PP7	TIP	I/O	Parallel port bit/transfer in progress	8
199	PP6	DREQ0	I/O	Parallel port bit/DMA request	8
200	PP5	DREQ1	I/O	Parallel port bit/DMA request	8
201	GND	—	—	Ground pin	—
202	PP4	TM2	I/O	Parallel port bit/Transfer modifier	8

Table 23. Pins 157–208 (Top, Right-to-Left) (Continued)

Pin		Alternate Function	I/O	Description	Drive (mA)
No	Name				
203	PP3	TM1/DACK1[1]	I/O	Parallel port bit/Transfer modifier/DMA acknowledge	8
204	PP2	TM0/DACK0[1]	I/O	Parallel port bit/Transfer modifier/DMA acknowledge	8
205	EVCC	—	—	3.3-V power input	—
206	PP1	TT1	I/O	Parallel port bit/Transfer type	8
207	PP0	TT0	I/O	Parallel port bit/Transfer type	8
208	GND	—	—	Ground pin	—

1 When the internal DMA is used, PP3 and PP2 (PP[3:2]/TM[1:0]) can be programmed to a third function, (DACK[1:0]), which indicates DMA acknowledge.

1.12.3 Mechanical Diagram

Figure 14 is a mechanical diagram of the 208-pin QFP MCF5407.

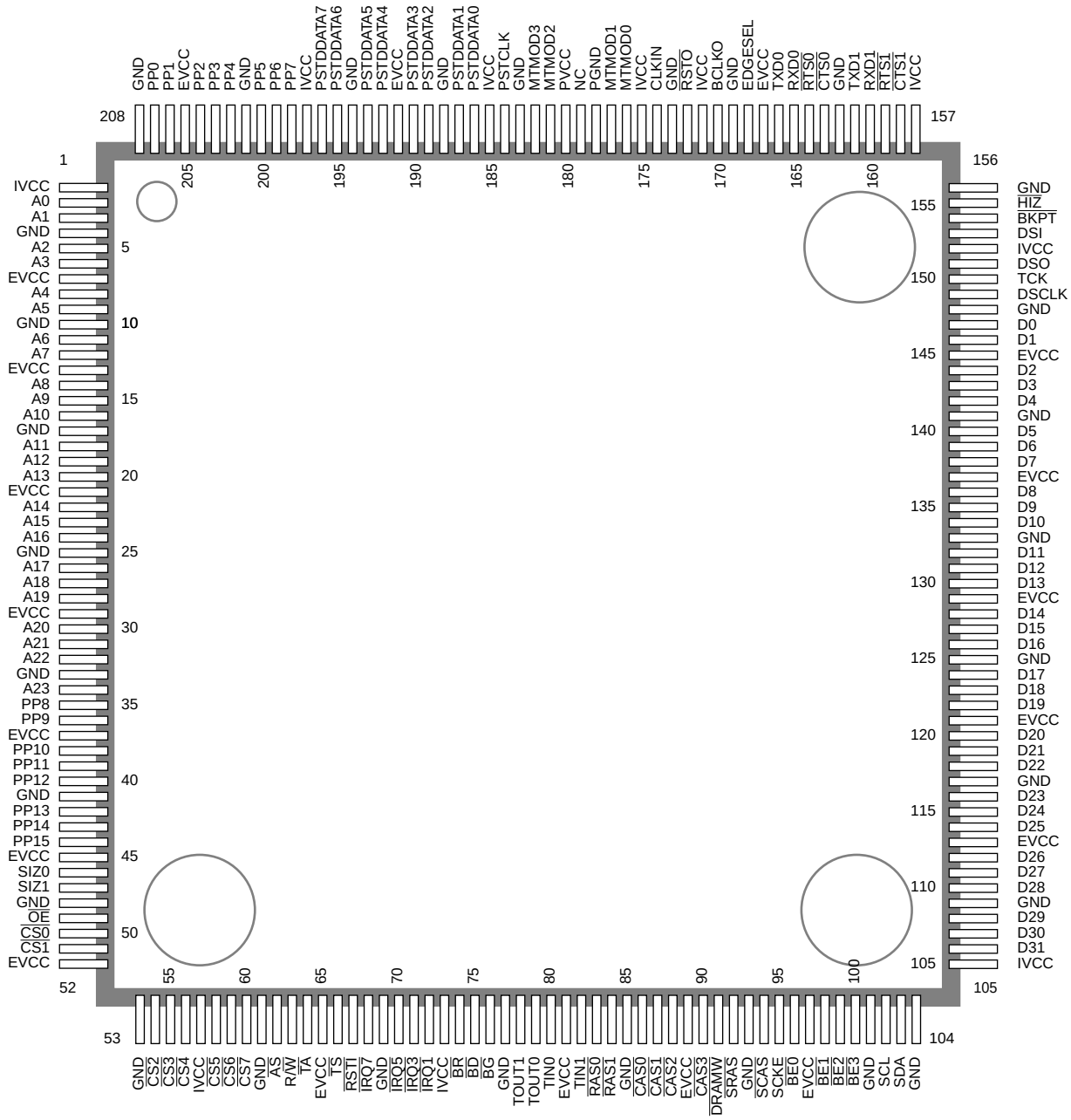


Figure 14. Mechanical Diagram

1.13 ColdFire Instruction Set Architecture Enhancements

This section describes the new opcodes implemented as part of the Revision B (ISA B) enhancements to the basic ColdFire ISA. In some cases, the opcodes represent minor enhancements to existing ColdFire functions, while in other cases, the functionality is new and not covered in the existing ISA.

Bcc

Bcc

Branch Conditionally

Operation: If Condition True
Then PC + d_n → PC

Assembler Syntax: Bcc <label>

Attributes: Size = byte, word, long

Description: If the specified condition is true, program execution continues at location (PC) + displacement. The program counter contains the address of the instruction word for the Bcc instruction, plus two. The displacement is a two's-complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is 0, a 16-bit displacement (the word immediately after the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (0xFF), the 32-bit displacement (longword immediately following the instruction) is used. Condition code cc specifies one of the following conditional tests:

Code	Condition	Code	Condition	Code	Condition	Code	Condition
CC(HI)	Carry clear	GT	Greater than	LT	Less than	VC	Overflow clear
CS(LO)	Carry set	HI	High	MI	Minus	VS	Overflow set
EQ	Equal	LE	Less or equal	NE	Not equal		
GE	Greater or equal	LS	Low or same	PL	Plus		

Condition Codes: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Condition				8-bit displacement							
16-bit displacement if 8-bit displacement = 0x00															
32-bit displacement if 8-bit displacement = 0xFF															

Instruction Fields:

- Condition field—Binary code for one of the conditions listed in the table.
- 8-bit displacement field—Two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.
- 16-bit displacement field—Used for displacement when the 8-bit displacement contains 0x00.
- 32-bit displacement field—Used for displacement when the 8-bit displacement contains 0xFF.

NOTE:

A branch to the next immediate instruction uses the 16-bit displacement format because the 8-bit displacement field contains 0x00 (zero offset).

Bcc	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.b, .w	.b, .w, .l

BRA

BRA

Branch Always

Operation: PC + d_n → PC

Assembler Syntax: BRA <label>

Attributes: Size = byte, word, long

Description: Program execution continues at location (PC) + displacement. The program counter contains the address of the instruction word of the BRA instruction, plus two. The displacement is a two's complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is 0, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (0xFF), the 32-bit displacement (longword immediately following the instruction) is used.

Condition codes: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
0	1	1	0	0	0	0	0	8-bit displacement
16-bit displacement if 8-bit displacement = 0x00								
32-bit displacement if 8-bit displacement = 0xFF								

Instruction Fields:

- 8-bit displacement field—two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.
- 16-bit displacement field—Used for displacement when the 8-bit displacement contains 0x00.
- 32-bit displacement field—Used for displacement when the 8-bit displacement contains 0xFF.

NOTE:

A branch to the next immediate instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains 0x00 (zero offset).

BRA	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.b, .w	.b, .w, .l

BSR BSR

Branch to Subroutine

Operation:SP – 4 → SP; PC → (SP); PC + d_n → PC

Assembler Syntax: BSR <label>

Attributes: Size = byte, word, long

Description: Pushes the word address of the instruction immediately following the BSR instruction onto the system stack. The program counter contains the address of the instruction word, plus two. Program execution then continues at location (PC) + displacement. The displacement is a two's complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is 0, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (0xFF), the 32-bit displacement (longword immediately following the instruction) is used.

Condition Codes: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
0	1	1	0	0	0	0	1	8-bit displacement
16-bit displacement if 8-bit displacement = 0x00								
32-bit displacement if 8-bit displacement = 0xFF								

Instruction Fields:

- 8-bit displacement field—two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.
- 16-bit displacement field—Used for displacement when the 8-bit displacement contains 0x00.
- 32-bit displacement field—Used for displacement when the 8-bit displacement contains 0xFF.

NOTE:

A branch to the next immediate instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains 0x00 (zero offset).

BSR	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.b, .w	.b, .w, .l

CMP

Compare

CMP

`Operation:Destination – Source → cc`

Assembler Syntax: CMP <ea>y, Dx

Attributes: Size = byte, word, long

Description: Subtracts the source operand from the destination operand in the data register and sets condition codes according to the result; the data register is unchanged. The operation size may be a byte, word, or longword.

CMPA is used when the destination is an address register; CMPI is used when the source is immediate data. Most assemblers automatically make this distinction.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

- Register field—specifies the destination register.
- Opmode field:

Byte	Word	Long	Operation
000	001	010	Dx - <ea>y

- Effective address field specifies the source operand; use addressing modes in the following table:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dy	000	reg. number:Dy		(d_8,Ay,Xi)	110	reg. number:Ay
Ay (word/longword operand only)	001	reg. number:Ay		(xxx).W	111	000
(Ay)	010	reg. number:Ay		(xxx).L	111	001
(Ay) +	011	reg. number:Ay		#<data>	111	100
– (Ay)	100	reg. number:Ay		(d_{16},PC)	111	010
(d_{16},Ay)	101	reg. number:Ay		(d_8,PC,Xi)	111	011

CMP	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.l	.b, .w, .l

CMPA

CMPA

Compare Address

Operation: `Destination – Source → cc`

Assembler Syntax: CMPA <ea>y, Ax

Attributes: Size = word, long

Description: Operates similarly to CMP, but is used when the destination register is an address register rather than a data register. The operation size can be word or longword. Word-length source operands are sign-extended to 32 bits for comparison.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

- Register field—specifies the destination register.
- Opmode field:

Byte	Word	Long	Operation
—	011	111	Ax - <ea>y

- Effective address field specifies the source operand; use addressing modes in the following table:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dy	000	reg. number:Dy		(d_8,Ay,Xi)	110	reg. number:Ay
Ay (word/longword operand only)	001	reg. number:Ay		(xxx).W	111	000
(Ay)	010	reg. number:Ay		(xxx).L	111	001
(Ay) +	011	reg. number:Ay		#<data>	111	100
– (Ay)	100	reg. number:Ay		(d_{16},PC)	111	010
(d_{16},Ay)	101	reg. number:Ay		(d_8,PC,Xi)	111	011

CMPA	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.l	.w, .l

CMPI

CMPI

Compare Immediate

`Operation:Destination – Immediate Data → cc`

Assembler Syntax: CMPI #<data>, Dx

Attributes: Size = byte, word, long

Description: Operates similarly to CMP, but is used when the source operand is immediate data. The size of the immediate data The size of the operation may be specified as byte, word, or longword. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	SIZE		0	0	0	REGISTER		
UPPER WORD															
LOWER WORD															

Instruction Fields:

- Register field—destination data register.
- Size field:

Byte	Word	Long	Operation
00	01	10	Dx - #<data>

NOTE:

If size = byte, the immediate is contained in bits [7:0] of the single extension word.

If size = word, the immediate is contained in bits[15:0] of the single extension word.

If size = long, the immediate is contained in the two extension words.

CMPI	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.l	.b, .w, .l

INTOUCH INTOUCH

Instruction Fetch Touch

```
Operation: If Supervisor State
               then Instruction Fetch Touch @ <Ay>
           else TRAP
```

Assembler Syntax INTOUCH <Ay>

Attributes: Unsized

Description: Generates an instruction fetch reference at address (Ay). If the referenced address space is a cacheable region, this instruction can be used to prefetch a 16-byte packet into the processor's instruction cache. If the referenced instruction address is a non-cacheable space, the instruction effectively performs no operation.

The INTOUCH instruction can be used to prefetch, and with the later setting of CACR[11], lock specific memory lines in the processor's instruction cache. This function may be desirable in systems where deterministic real-time performance is critical.

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	0	0	1	0	1	REGISTER		

Instruction Fields:

- Register field—specifies the destination address register number.

INTOUCH	V2, V3 Core	V4 Core
Opcode present	No	Yes
Operand sizes supported	—	—

MOVE MOVE

Move Data from Source to Destination

`Operation:Source → Destination`

Assembler Syntax: MOVE <ea>y, <ea>x

Attributes: Size = byte, word, long

Description: Moves the data at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or longword.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE		DESTINATION				SOURCE							
				REGISTER			MODE			MODE			REGISTER		

Instruction fields:

- Size field—specifies the size of the operand to be moved:

  01 —byte operation
  11 —word operation
  10 —long operation

- Destination effective address field—Specifies destination location; the table below lists possible data alterable addressing modes. The restrictions on combinations of source and destination addressing modes are listed in the table at the bottom of the next page.

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dx	000	reg. number:Dx		(d_8,Ax,Xi)	110	reg. number:Ax
Ax	—	—		(xxx).W	111	000
(Ax)	010	reg. number:Ax		(xxx).L	111	001
(Ax) +	011	reg. number:Ax		#<data>	—	—
– (Ax)	100	reg. number:Ax		(d_{16},PC)	—	—
(d_{16},Ax)	101	reg. number:Ax		(d_8,PC,Xi)	—	—

- Source effective address field—Specifies source operand; the table below lists possible addressing modes. The ColdFire MOVE instruction has restrictions on combinations of source and destination addressing modes. The table at the end of this instruction description outlines the restrictions.

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dy	000	reg. number:Dy		(d_8,Ay,Xi)	110	reg. number:Ay
Ay	001	reg. number:Ay		(xxx).W	111	000
(Ay)	010	reg. number:Ay		(xxx).L	111	001
(Ay) +	011	reg. number:Ay		#<data>	111	100
– (Ay)	100	reg. number:Ay		(d_{16},PC)	111	010
(d_{16},Ay)	101	reg. number:Ay		(d_8,PC,Xi)	111	011

NOTE:

Most assemblers use MOVEA when the destination is an address register.

Use MOVEQ to move an immediate 8-bit value to a data register. Use MOV3Q to move a 3-bit immediate value to any effective destination address.

Not all combinations of source/destination addressing modes are possible. The table below shows the possible combinations.

Source Addressing Mode	Destination Addressing Mode
Dy, Ay, (Ay), (Ay)+,-(Ay)	All possible
(d_{16}, Ay), (d16, PC)	All possible except (d_8, Ax, Xi), (xxx).W, (xxx).L
(d8, Ay, Xi), (d8, PC, Xi), (xxx).W, (xxx).L, #<xxx>	All possible except (d_8, Ax, Xi), (xxx).W, (xxx).L

Note: The combination of #<xxx>,d16(Ax) addressing modes can be used only on move byte and move word opcodes. Refer to the previous tables for valid source and destination addressing modes.

MOVE	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	.b, .w, .l except move.x #<data>, d16(Ax)	.b, .w, .l including move.{b,w} #<data>, d16(Ax)

MOVEA MOVEA

Move Address from Source to Destination

`Operation:Source → Destination`

Assembler Syntax: MOVEA <ea>y, Ax

Attributes: Size = word, long

Description: Moves the address at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as word or longword.

Condition Codes: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		

Instruction fields:

- Size field—specifies the size of the operand to be moved:

  11 —word operation
  10 —long operation

- Destination effective address field—Specifies the destination location; the table below lists possible addressing modes.

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dx	—	—		(d_8,Ax,Xi)	—	—
Ax	001	reg. number: Ax		(xxx).W	—	—
(Ax)	—	—		(xxx).L	—	—
(Ax) +	—	—		#<data>	—	—
– (Ax)	—	—		(d_{16},PC)	—	—
(d_{16},Ax)	—	—		(d_8,PC,Xi)	—	—

- Source effective address field—Specifies the source operand; the table below lists possible modes.

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dy	000	reg. number:Dy		(d_8,Ay,Xi)	110	reg. number:Ay
Ay	001	reg. number:Ay		(xxx).W	111	000
(Ay)	010	reg. number:Ay		(xxx).L	111	001
(Ay) +	011	reg. number:Ay		#<data>	111	100
– (Ay)	100	reg. number:Ay		(d_{16},PC)	111	010
(d_{16},Ay)	101	reg. number:Ay		(d_8,PC,Xi)	111	011

MOVEA	V2, V3 Core	V4 Core
Opcode present	Yes	Yes
Operand sizes supported	No differences	

MOV3Q

MOV3Q

Move 3-Bit Data Quick

`Operation:Immediate Data → Destination`

Assembler Syntax MOV3Q #<data>,<ea>x

Attributes: Size = long

Description: Move the immediate data to the operand at the destination location. The data range is from -1 to 7, excluding 0. The immediate data is zero-filled to a long operand and all 32 bits are transferred to the destination location.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

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

Instruction Fields:

- Data field—3 bits of data having a range {-1,1-7} where a data value of 0 represents -1.
- Effective Address field—specifies the destination operand; use only data addressing modes listed in the following table:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dx	000	reg. number:Dx		(d_8,Ax,Xi)	110	reg. number:Ax
Ax	001	reg. number:Ax		(xxx).W	111	000
(Ax)	010	reg. number:Ax		(xxx).L	111	001
(Ax) +	011	reg. number:Ax		#<data>	—	—
– (Ax)	100	reg. number:Ax		(d_{16},PC)	—	—
(d_{16},Ax)	101	reg. number:Ax		(d_8,PC,Xi)	—	—

MOV3Q	V2, V3 Core	V4 Core
Opcode present	No	Yes
Operand sizes supported	—	.l

MVS MVS

Move with Sign Extend

`Operation:(Source with sign extension) → Destination`

Assembler Syntax: MVS <ea>y,Dx

Attributes: Size = byte, word

Description: Sign-extend the source operand and move to the destination register. For the byte operation, bit 7 of the source is copied to bits 31–8 of the destination. For the word operation, bit 15 of the source is copied to bits 31–16 of the destination.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	REGISTER			1	0	SIZE	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

- Size field—specifies the size of the operation
  0 byte operation
  1 word operation
- Register field—specifies a data register as the destination.
- Effective address field—specifies the source operand; use only data addressing modes from the following table:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dy	000	reg. number:Dy		(d_8,Ay,Xi)	110	reg. number:Ay
Ay	001	reg. number:Ay		(xxx).W	111	000
(Ay)	010	reg. number:Ay		(xxx).L	111	001
(Ay) +	011	reg. number:Ay		#<data>	111	100
– (Ay)	100	reg. number:Ay		(d_{16},PC)	111	010
(d_{16},Ay)	101	reg. number:Ay		(d_8,PC,Xi)	111	011

MVS	V2, V3 Core	V4 Core
Opcode present	No	Yes
Operand sizes supported	—	.b, .w

MVZ MVZ

Move with Zero-Fill

`Operation:(Source with zero fill) → Destination`

Assembler Syntax MVZ <ea>y,Dx

Attributes: Size = byte, word

Description—Zero-fill the source operand and move to the destination register. For the byte operation, the source operand is moved to bits 7–0 of the destination and bits 31–8 are filled with zeros. For the word operation, the source operand is moved to bits 15–0 of the destination and bits 31–16 are filled with zeros.

Condition Codes:

X	N	Z	V	C
—	0	*	0	0

X Not affected
N Always cleared
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	REGISTER			1	1	SIZE	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

- Size field—specifies the size of the operation
  0 byte operation
  1 word operation
- Register field—specifies a data register as the destination.
- Effective address field—specifies the source operand; use the following data addressing modes:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dy	000	reg. number:Dy		(d_8,Ay,Xi)	110	reg. number:Ay
Ay	001	reg. number:Ay		(xxx).W	111	000
(Ay)	010	reg. number:Ay		(xxx).L	111	001
(Ay) +	011	reg. number:Ay		#<data>	111	100
– (Ay)	100	reg. number:Ay		(d_{16},PC)	111	010
(d_{16},Ay)	101	reg. number:Ay		(d_8,PC,Xi)	111	011

MVZ	V2, V3 Core	V4 Core
Opcode present	No	Yes
Operand sizes supported	—	.b, .w

SATS

Signed Saturate

SATS

```
Operation:

        if CCR.V == 1,
                then if Dx[31] == 0,
                        then Dx[31:0] = 0x80000000
                        else Dx[31:0] = 0x7FFFFFFF
                else Dx[31:0] is unchanged
```

Assembler Syntax: SATS Dx

Attributes: Size = long

Description: Update the destination register only if the overflow bit of the CCR is set. If the operand is negative, then set the result to greatest positive number; otherwise, set the result to the largest negative value. The condition codes are set according to the result.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

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

Instruction Fields:

- Register field—specifies the destination data register.

SATS	V2, V3 Core	V4 Core
Opcode present	No	Yes
Operand sizes supported	—	.l

TAS

TAS

Test and Set an Operand

Operation: Destination Tested → CCR; 1 → bit 7 of Destination

Assembler Syntax: TAS <ea>x

Attributes: Size = byte

Description: Tests and sets the byte operand addressed by the effective address field. The instruction tests the current value of the operand and sets the N and Z condition code bits appropriately. TAS also sets the high order bit of the operand. The operand uses a read-modify-write memory cycle that completes the operation without interruption. This instruction supports use of a flag or semaphore to coordinate several processors.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected
N Set if the msb of the operand is currently set; cleared otherwise
Z Set if the operand was zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

- Effective address field—specifies the destination location; the possible data alterable addressing modes are listed in the table below.

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dx	—	—		(d_8,Ax,Xi)	110	reg. number:Ax
Ax	—	—		(xxx).W	111	000
(Ax)	010	reg. number:Ax		(xxx).L	111	001
(Ax) +	011	reg. number:Ax		#<data>	—	—
– (Ax)	100	reg. number:Ax		(d_{16},PC)	—	—
(d_{16},Ax)	101	reg. number:Ax		(d_8,PC,Xi)	—	—

TAS	V2, V3 Core	V4 Core
Opcode present	No	Yes
Operand sizes supported	—	.b

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor