MCF5407

Errata to MCF5407

Preface

This document identifies implementation differences between the MCF5407 processor and the description contained in the MCF5407 User's Manual. Please check the WWW at http://www.motorola.com/ColdFire for the latest updates. This errata lists differences from the following documents:

- MCF5407 User's Manual
- *ColdFire Microprocessor Family Programmer's Reference Manual*

Table 1 summarizes MCF5407 errata.

Table 1. Summary of MCF5407 Errata

Errata ID	Module Affected	Date Errata Added	Errata Title
1.	Core	24 Apr 2000	Variant JSR in upper instruction word + RTS in lower instruction word can result in erroneous processor execution
2.	Core	24 Apr 2000	CSR indicator fails to correctly report source of processor core halt
3.	Core	24 Apr 2000	Fault-on-fault does not always halt
4.	Core	24 Apr 2000	TAS instruction does not check write-protect
5.	Core	24 Apr 2000	Write-protect fault hangs
6.	Core	24 Apr 2000	During exceptions, references to data memory may occur in the wrong privilege mode
7.	Core	24 Apr 2000	Misdirection of final write cycles of misaligned read-modify-write followed by a MOVEC/CPUSHL
8.	Core	24 Apr 2000	CPUSHL instruction ignores cancellation
9.	Core	24 Apr 2000	Trace of STOP instruction does not work
10.	Core (MAC)	24 Apr 2000	MAC Fractional -1 * -1 overflow detection erratic

Table 1. Summary of MCF5407 Errata (Continued)

Errata ID	Module Affected	Date Errata Added	Errata Title
11.	Core (Debug)	24 Apr 2000	DBG Trigger state machine not sequencing correctly
12.	Core (Debug)	24 Apr 2000	Possible PSTDDATA errors while RMW is capturing both R&W data
13.	Core (Debug)	24 Apr 2000	Data inversion triggers are incorrect for word and longword accesses
14.	Core (Debug)	24 Apr 2000	Second-level breakpoint trigger missed if it occurs on the cycle immediately after assertion of first-level trigger
15.	Core (Instruction Cache)	24 Apr 2000	CACR bits for instruction cache default cache mode and cache freeze do not work properly
16.	DMA	24 Apr 2000	DMA writes to UART cause transmission errors
17.	DMA	24 Apr 2000	DMA single-address access mode and cycle-steal mode do not work together
18.	DMA arbiter	24 Apr 2000	DMA channel arbiter hangs in round-robin mode
19.	DRAM Controller	24 Apr 2000	DRAM Controller: External master termination is not correct

1. Variant JSR in upper instruction word + RTS in lower instruction word can result in erroneous processor execution

1.1 Description

The MCF5407 contains a hardware return stack to accelerate RTS instruction performance. An RTS acceleration bug exists for the scenario where a variant JSR instruction (such as JSR a0) is in the upper 16-bits and an RTS is in the lower 16-bits of the same longword of instruction memory. Attempted processor execution of this sequence could cause generation of a bogus instruction fetch address which could result in possible failures such as processor hang, an unexpected exception, or data corruption.

There are multiple copies of the return stack in the processor hardware to accelerate RTS performance. These copies can lose synchronization when presented with a variant JSR (such as JSR %A0) whose target is unknown during prefetch. The simultaneous detection of both a return stack push and a pop causes a subsequent JSR to overwrite entry 1 of the return stack instead of entry 0.

Later, when an attempt is made to resynchronize the internal copies, the uninitialized entry 0 is erroneously validated. Subsequent use of this uninitialized target causes a spurious instruction fetch address.

1.2 Workarounds

1.2.1 Software

Initialize the hardware return stack to known good target values during system initialization. This can be done with a short piece of assembly code, such as the following:

make sure the code matches the syntax used in the UM

```
move.l  cacr.w,d0#get CACR value
move.l  d0,d1   #save the original value
ori.l   #0xC0000,d0#ensure branch cache is enabled/cleared
move.l  d0,cacr #load CACR enabling branch cache
bsr.w   .+4     #initialize return stack entry
bsr.w   .+4     #initialize return stack entry
bsr.w   .+4     #initialize return stack entry
bsr.w   .+4     #initialize return stack entry
lea     16(a7),a7#pop stack of unneeded return addresses
move.l  d1,cacr #reload CACR to original value
```

1.2.2 Hardware

Disable the hardware return stack, CACR[16] = 1

2. CSR indicator fails to correctly report source of processor core halt

2.1 Description

The halt indicators in the debug module's configuration/status register (CSR[27–24]) fail to report the reason the processor core is halted. CSR[26] is supposed to be set when a debug breakpoint register has triggered and halted the processor. Although this bit operates correctly if the breakpoint involves an address (and an optional data) breakpoint, it fails if a PC breakpoint register causes the trigger event.

An error also occurs when the processor is restarted with a BDM GO command. In this situation, the 4-bit halt field is supposed to be cleared. This behavior of the halt status field was one of the specific enhancements in the Debug Revision C. Formerly, the halt status field was cleared if the CSR was read. As part of the Debug C changes, this function is no longer supported, and the halt status field is cleared only when the processor is restarted. However, if the trigger definition registers (TDR and XTDR) are not cleared while halted, then CSR[26] is negated for only a single processor cycle before being set again. This operation of CSR[26] is independent of the type of breakpoint trigger (address {and optional data}, or PC) event.

Note: In all cases, the processor core's behavior in response to the hardware breakpoint is correct; it is only the CSR reporting mechanism that is not operating correctly.

2.2 Software Workarounds

There are two potential workarounds:

- To work around the problem associated with the setting of CSR[26], it is recommended that PC breakpoint triggers be configured to generate debug interrupts with a HALT instruction included as the first instruction of the debug interrupt service routine. If the processor halts due to the execution of a HALT instruction and the program counter address matches the location in the debug interrupt service routine, a PC breakpoint trigger has occurred. The exception stack frame in memory defines the PC of the actual trigger instruction.
- To prevent the extraneous setting of CSR[26] after a BDM GO command, the trigger definition registers should be cleared while the processor is halted. Clearing both the TDR and XTDR while the processor is halted guarantees that CSR[26] is not set again when the BDM GO command is issued to restart the processor.

3. Fault-on-fault does not always halt

3.1 Description

The ColdFire architecture specifies that detection of any fault condition during exception processing should generate a catastrophic fault-on-fault condition and halt the processor. Only a processor reset can exit this condition. Essentially three potential fault-on-fault conditions are examined during exception processing: two access errors during the exception stack frame writes, and an address error on the exception vector defining the service routine location. In the case where an address error occurs during the exception vector fetch – that is when an exception vector table entry contains an odd address – the processor does not halt even though the address error condition is successfully detected. This happens because the recognition occurs too late – outside the hardware's window of time defining the fault-on-fault condition. As a result, the processor takes a second address error exception. Unfortunately, the second exception frame is not logically coherent because it reports the original PC value as the faulting value while the status register indicates supervisor mode. While it is possible – depending on the exact behavior of the address error service routine – to correctly restart these two exception frames, this is not guaranteed.

The ColdFire architecture specifies that detection of any fault condition during exception processing generates a catastrophic fault-on-fault and halts the processor. Only a processor reset can exit this condition. There are essentially three potential error conditions that are examined during exception processing: two access errors during the exception stack frame writes and an address error on the exception vector defining the location of the service routine.

An address error condition is detected successfully, but outside of the window of time that defines the fault-on-fault condition. As a result, the processor takes a second address error exception after the initial exception. Unfortunately, the second exception frame is not logically coherent because it reports the original PC as the faulting program counter, but the status register indicates supervisor mode.

It is not guaranteed that these two exception frames could be restarted correctly, although this is very dependent on the exact behavior of the address error service routine. The correct hardware response is that the processor halts with the fault-on-fault condition.

3.2 Software Workarounds

There are two potential workarounds to this error:

- Guarantee that no exception vector table includes pointers with odd addresses.
- If option 1 cannot be guaranteed, check the address error service routine to determine if there is an exception stack frame immediately before the address error frame. If so, the address error service routine should pass control to that exception handler after completing execution.

4. TAS instruction does not check write-protect

4.1 Description

TAS is a locked, read-modify-write instruction that can be used to build task synchronization routines. It has an implicit cache mode of non-cacheable precise.

A memory region can be marked as write protected by setting the appropriate write protect bit in the RAMBARs, ACRs, or the CACR.

Due to this error, if a memory region is marked as write-protected in the ACRs and a TAS instruction reference is made to the memory space defined by that ACR, a write-protect fault is not taken.

The 5407 provides two ACRs dedicated to the data memory (ACR0 and ACR1).

The effective cache mode is calculated as follows:

The local data memory address is compared with, from highest to lowest RAMBAR0, RAMBAR1, ACR0 and ACR1. If no match is found, the default access attributes in the CACR for the data cache are used for that access.

4.2 Software Workarounds

- Do not use ACR0 or ACR1 to mark a memory region as write protected.

  or

- Do not execute a TAS instruction that references a memory region marked write protected by ACR0 or ACR1.

5. Write-protect fault hangs

5.1 Description

A memory region can be marked as write protected by setting the appropriate write protect bit in the RAMBARs, ACRs, or CACR.

Write references made to the memory space marked write protected in the ACRs or the CACR may hang. The exact circumstances are based on instruction local memory bus activity and are therefore unpredictable.

The effective cache mode for the data local memory bus is calculated as follows:

The operand local memory bus address is compared with, from highest-to-lowest RAMBAR0, RAMBAR1, ACR0 and ACR1. If no match is found, the default access attributes in the CACR for the data cache are used for the access.

5.2 Software Workaround

Note that the 5407 provides two ACRs dedicated to the local data bus memories (ACR0 and ACR1). Do not mark a memory region as write protected in ACR0, ACR1, or CACR.

6. During exceptions, references to data memory may occur in the wrong privilege mode

6.1 Description

When a ColdFire processor initiates any type of exception processing, the privilege level is switched to supervisor mode. There is a brief window during exception processing when the program-visible supervisor bit of the status register (SR[13]) may differ from the operating mode of the processor. This is required because the program state at the time of the fault (including the SR contents) must first be saved in the exception stack frame, before SR[13] is set.

During this window, the processor generates three data memory references: the two longword exception stack frame writes and the longword read of the exception vector. Although all three references should be treated as supervisor-mode accesses, they are attempted in the operating mode at the time of the exception. Therefore, if the processor is in user mode at the time of the exception, these three accesses are incorrectly made as user mode references.

6.2 Software Workaround

Guarantee that the attributes of the memory regions containing the system stack and the exception vector table are equivalent between supervisor and user modes. The memory attributes include: the local RAM/non-local RAM mapping, cacheability definition (non-cacheable, copyback, write-through) and write protection. Given equivalent attributes, this design error is invisible.

Conversely, if memory attributes differ between supervisor and user modes for these regions, the exception stack frame writes and exception vector fetch may reference incorrect memory regions, corrupting memory locations on the exception writes and/or returning incorrect data on the vector read.

7. Misdirection of final write cycles of misaligned read-modify-write followed by a MOVEC/CPUSHL

7.1 Description

If the privileged instructions, MOVEC or CPUSHL, immediately follow a misaligned operand read-modify-write instruction, the processor may incorrectly handle the last portion(s) of the misaligned write causing data corruption or a system hang. Specifically, the MOVEC and CPUSHL instructions specify unique encodings for the operand transfer type and transfer modifier fields, and in certain situations, the processor may prematurely switch from the default values to those required by MOVEC and CPUSHL during the last phases of the misaligned write. The net effect is that the last write cycle associated with the misaligned read-modify-write instruction may be mapped to an incorrect address space.

The affected read-modify-write instructions include the following:

- add.l Dy,<mem>x
- addq.l #imm,<mem>x
- and.l Dy,<mem>x
- eor.l Dy,<mem>x
- or.l Dy,<mem>x
- sub.l Dy,<mem>x
- subq.l #imm,<mem>x

7.2 Software Workaround

Most supervisor code using MOVEC and CPUSHL instructions follows a two-instruction template, where a register load instruction immediately precedes the MOVEC/CPUSHL, of the following form:

```
move.l #imm,Rx  ; load operand value for control register
movec Rx,Rc     ; move it into control register

move.l #imm,Ax  ; define address for cache operation
cpushl ,Ax      ; perform cache operation
```

CPUSHL instructions also typically appear in loop constructs, often with LEA or ADDQ.L instructions immediately preceding the push opcode. Systems making exclusive use of these types of templates are not affected by this errata.

For other situations, if the instruction immediately preceding every MOVEC or CPUSHL instruction is not a read-modify-write opcode, or if it is a read-modify-write without operand misalignment, this problem is completely avoided. If the system does not decode TM, TT signals, this design error is invisible.

8. CPUSHL instruction ignores cancellation

8.1 Description

A CPUSHL instruction is not cancelled correctly if it is the target of an incorrectly predicted return or the target of a debug PC breakpoint that causes a halt. If either of these scenarios occur, the MCF5407 processor may execute an incorrect instruction (unexpected exception or data corruption possible) or the processor instruction local bus may hang (resulting in processor hang).

CPUSHL instructions are correctly cancelled in the data cache. However, in the instruction cache, a CPUSHL is not cancelled correctly if it is the target of an incorrectly predicted return or the target of a debug PC breakpoint that causes a halt. If either of these scenarios occur, an incorrect instruction may be executed by the MCF5407 processor or the processor instruction local bus may hang, which in turn hangs the processor.

8.2 Software Workarounds

Do not let the CPUSHL instruction be the target of a debug PC breakpoint that causes a halt or of an incorrectly predicted return. To do this, do not set a PC breakpoint to halt on a CPUSHL instruction and do either of the following:

- Do not have a CPUSHL instruction as the target of any RTS

  or
- Disable the return stack, CACR[16] = 1

9. Trace of STOP instruction does not work

9.1 Description

Instruction-by-instruction tracing is controlled by the T-bit in the Status Register (SR[15]). This bit is loaded during the execution of three privileged instructions, one of which is the STOP instruction. If the MCF5407 is operating in trace mode, or if T-bit is being set to a "1" during a STOP instruction's load, the specified processor action upon executing the STOP instruction is to take a trace exception. Due to a processor error, the actual result upon executing a STOP in any of these trace scenarios is that the processor stops (no trace exception).

The STOP instruction stops when in trace mode with no trace exception taken and causes the test to time out.

When executing a STOP instruction while in trace mode, regardless of whether the STOP instruction itself enables or disables trace mode, a trace exception is taken. Also, if a STOP instruction is executed while not in trace mode, but the STOP instruction itself enables trace mode, a trace exception is taken. Whenever the trace exception is taken, the STOP instruction is effectively a MOVE-to-SR instruction, though the PST lines will indicate that a STOP instruction has been executed (PST = e).

9.2 Software Workaround

Don't use the STOP instruction in trace mode.

While there is no direct workaround for this error, the following sequence can be used to emulate a STOP instruction without actually stopping the processor.

```
      move.w #data,sr
      .align 4

loop:

      trapf
      bne.b loop
```

10. MAC Fractional -1 * -1 overflow detection erratic

10.1 Description

In fractional mode, 32-bit results in the range [-1,+1) are generated in the accumulator of the MAC unit. The bracket indicates that -1 is included and the parenthesis indicates that +1 is not. The product itself of two inputs will always fall in this range except in the case of -1 × -1 yielding +1. The +1 product is handled as though no overflow has occurred and added to the accumulator. Only if the accumulator is initially non-negative will overflow be signalled by setting the V condition code bit in the MACSR, and by saturating the accumulator if the OMC bit (saturation mode) is set. If the product is to be subtracted from the accumulator (MSAC instruction) then overflow will occur if the accumulator is initially negative.

There is an error in the detection of the overflow in the special case when the two 16-bit operands are obtained from different halves of CPU registers. There is no problem with MAC instructions with 32-bit operands or with MAC instructions with both 16-bit operands residing in the lower halves or both in the upper halves of CPU registers. The execution of the following instructions may result in an erroneous V (overflow) bit state if both operands = -1 (8000 hexadecimal):

```
mac.w   Rx:u,Ry:l
mac.w   Rx:l,Ry:u
msac.w  Rx:u,Ry:l
msac.w  Rx:l,Ry:u
```

If saturation mode is not enabled, only the setting of the V bit may be in error. If saturation mode is enabled, the result in the accumulator may also be wrong.

10.2 Software Workarounds

If 16-bit operands MAC instruction operands are being used from different halves of different CPU registers (y != x), and if the V bit is being checked or if the MAC unit is operating in saturation mode, then any of the above instructions can be replaced with a three instruction sequence. For instance, the following instruction:

```
mac.w   Rx:u,Ry:l
```

can be replaced with the following equivalent sequence:

```
swap    Rx
mac.w   Rx:l,Ry:l
swap    Rx
```

And the instruction:

```
mac.w   Rx:l,Ry:u
```

can be replaced with:

```
swap    Rx
mac.w   Rx:u,Ry:u
swap    Rx
```

Of course, the second swap instruction in each case should be removed if the MAC instruction specifies a memory operand that is to be loaded into the Rx register.

Similar sequences can be substituted for the two failing MSAC.W instructions.

The case for 16-bit operands from different halves of the same CPU register (y == x) is slightly more complicated. The instruction:

```
mac.w   Rx:u,Rx:l
```

can be replaced with:

```
      cmp.l     x,&0x80008000
      bne.b     label1
      mac.w     Rx:l,Rx:l
      bra.b     label2

label1:

      mac.w     Rx:u,Rx:l

label2:
```

Similar sequences can be used for the other failing instructions.

11. DBG Trigger state machine not sequencing correctly

11.1 Description:

Additional testing of the Version 4 debug functionality has uncovered problems with the 3-bit trigger state machine. The debug module maintains a 3-bit state machine to track the activity of hardware breakpoint triggers. The value of this trigger state machine is reported to the external development system (emulator) via two distinct paths: as the upper nibble of the configuration/status register (CSR) and as the default value on the PSTDDATA outputs when captured data is not being displayed.

Specifically, the trigger state machine outputs are defined in Table 2.

Table 2. Trigger State Machine Output Definitions

CSR[31–28]	PSTDDATA[3–0]	State description
0x0	0x0	No hardware breakpoints enabled
0x1	0x2	Waiting for level-1 trigger
0x2	0x4	Level-1 triggered
0x5	0xA	Waiting for level-2 trigger
0x6	0xC	Level-2 triggered

There are two problems, both associated with 2-level trigger conditions. In the first error, the final state for a level-2 trigger is incorrect. The machine is supposed to remain in the CSR=0x6, PSTDDATA=0xC state once the level-2 trigger has occurred. This state is retained until the breakpoint configuration is reprogrammed. In the current design, the machine reaches this state, but only remains there for a single processor cycle before incorrectly changing back to the CSR=0x5, PSTDDATA=0xA state. The sequence of state machine transitions are shown in Figure 1.

CSR =	0x1	0x2	0x5	0x6	0x5

Figure 1. Actual State Machine Transition Sequence

The second error involves the reporting of the trigger states on the PSTDDATA output. Specifically, the current design may miss certain trigger state changes, depending on the exact timing and sequence of their arrival. As an example, if the level-1 trigger and level-2 triggers occur with only a few cycles between them, the resulting trigger states displayed are shown in Figure 2.

CSR =	0x1	0x2	0x5	0x6	0x5
PSTDDATA =	0x2		0xA		0xA

Figure 2. Trigger States on CSR and PSTDDATA

Here, the PSTDDATA output loses both the level-1 triggered (CSR=0x2) and level-2 triggered (CSR=0x6) indications. It should also be noted that the CSR and PSTDDATA do *not* transition on the same machine cycle. Rather, this table is meant to simply show the sequence of transitions — not their cycle-accurate timing.

This failure is present in the 1.4* versions of the CF4Ref design, as well as the 2.* versions of the core.

11.2 Software Workarounds

Note in all cases, the processor core's behavior in response to the hardware breakpoint is correct; it is only the CSR/PSTDDATA reporting mechanism that is not operating correctly. There are two potential workarounds to these errors:

1. Do not directly use the debug module's two-level trigger capabilities. Rather, program the combined conditions as two 1-level triggers, i.e., setup the first condition as a level-1 breakpoint, then when it triggers, program the second condition as another level-1 breakpoint.
2. Given that the processor core correctly handles the one- and two- level breakpoint triggers, use the debug interrupt service routine to specifically output a value to the PST/PSTDDATA outputs (using the WPSTDDATA instruction) as a substitute for the PSTDDATA trigger state change indicator.

12. Possible PSTDDATA errors while RMW is capturing both R&W data

12.1 Description

The 5407 debug module includes a 4-entry, first-in, first-out (FIFO) buffer to store read, write, and/or address captures for output on PSTDDATA[7–0]. Depending upon the amount and sequencing of captures and the latency in processing them for output, the debug module sources a “stall” signal to the operand execution pipeline (OEP). This signal includes terms related to the number of entries in the data FIFO as well as the current pipeline state of data captures in the OEP. A stall is asserted when the data FIFO is filled and remains asserted until it can accept more capture data.

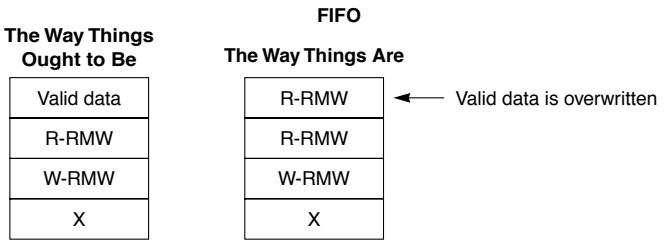

Figure 3. Read Data from Read-Modify-Write Operation Erroneously Overwrites Valid Data

If both read and write data captures are enabled, i.e., CSR[12–11] = 11, then the execution of single-cycle read-modify-write instructions result in two data operands being captured: first the read data, and then the write data. In the current design, the generation of the pipeline stall signal does not correctly account for this condition; thus depending upon the amount and sequencing of previous captures, one FIFO location can be overwritten. Thus, one incorrect data word is output from the core. The valid data that should have been output will be overwritten in the FIFO by the RMW instruction. This data will be followed by the RMW data. The read-modify-write instruction captures will be output correctly as will all subsequent instructions and data captures.

This “one incorrect PSTDDATA output” scenario can occur anytime a read-modify-write instruction occurs with only one entry in the PSTDDATA FIFO being available.

12.2 Software Workarounds

Do not enable both read and write captures; i.e. don't set CSR[12–11] = 11. Enabling read or write data captures separately functions correctly.

Another possible workaround would be to guarantee no single-cycle RMW instructions reference addresses mapped into cache or on-chip memories.

The single-cycle RMWs include the following instructions:

```
add.l   Dy,<mem>x
addq.l  #imm,<mem>x
and.l   Dy,<mem>x
eor.l   Dy,<mem>x
or.l    Dy,<mem>x
sub.l   Dy,<mem>x
subq.l  #imm,<mem>x
```

13. Data inversion triggers are incorrect for word and longword accesses

13.1 Description

The 5407 allows debug breakpoint triggers to be set on PC, and/or address, and optionally data compares. The trigger definition register (TDR), and extended trigger definition register (XTDR) register in the 5407, configures the breakpoint logic. In particular, TDR/XTDR[12:5] configures a data breakpoint. TDR/XTDR[12:6] configures the size of the data to be compared to the data breakpoint register DBR (and/or DBR1 in MCF5407) data value. Data breakpoints can be configured for byte-, word- or longword reference sizes. However, the data inversion bit, TDR/XTDR[5], does not work as advertised. When asserted, this bit is specified to enable data breakpoints to trigger for any access which does not match the specified data value. The data inversion feature works correctly for byte-size accesses, but is incorrect for 16- and 32-bit references. There are scenarios where triggers should occur, but don't, for 16- and 32-bit references.

13.2 Software Workaround

Do not use data inversion triggers/do not set TDR/XTDR[5] for word or longword breakpoints.

14. Second-level breakpoint trigger missed if it occurs on the cycle immediately after assertion of first-level trigger

14.1 Description

The 5407 allows debug breakpoint triggers to be set correlating to PC, and/or address, and optionally data compares. The configuration/status register bits, CSR[31–28], report breakpoint status, as follows:

- 0000 = No breakpoints enabled
- 0010 = Waiting for level 1 breakpoint
- 0100 = Level 1 breakpoint triggered
- 1010 = Waiting for level 2 breakpoint
- 1100 = Level 2 breakpoint triggered

A two-level breakpoint configuration means that the breakpoint cannot occur until the first-level trigger has been matched and then after that the second-level trigger matches.

A boundary condition bug exists for a two-level breakpoint configuration:

If a two-level breakpoint is configured, with both trigger conditions occurring in adjacent processor clock cycles, the second level trigger is missed and the trigger state incorrectly remains in the “waiting for trigger 2” state.

14.2 Software Workaround

Do not configure a two-level breakpoint; instead, configure two separate one-level breakpoints.

15. CACR bits for instruction cache default cache mode and cache freeze do not work properly

15.1 Description

Cache Control Register (CACR) bits [11:10] are defined as:

CACR[11] - instruction cache lock enable (0 = not locked, 1 = 1/2 instruction cache locked)
CACR[10] - instruction cache default cache mode (0 = cacheable, 1 = non-cacheable)

Because of this error, the CACR[11:10] do not work independently. For the current MCF5407, CACR[11] controls both the default instruction cache mode and lock enable. This results in only 2 of 4 combinations usable:

CACR[11] = 1 - 1/2 instruction cache locked; default mode non-cacheable
CACR[11] = 0 - instruction cache not locked; default mode cacheable

Because of this error, it is impossible to set the default instruction cache mode to cacheable and have the instruction cache locked; or to set the default instruction cache mode to non-cacheable and have the instruction cache not locked, (that is, four-way set associative).

The CACR[10, 11] do not work independently. CACR[11] is the instruction cache lock enable (0 = not locked, 1 = 1/2 instruction cache locked) and CACR[10] is the instruction cache default cache mode encode (0 = cacheable, 1 = non-cacheable).

The two combinations of CACR[11] and CACR[10] that are selectable with this error work as specified. That is, CACR[11] = CACR[10] = 0 gives a default instruction cache mode of cacheable and has the instruction cache not locked; CACR[11] = CACR[10] = 1 gives a default instruction cache mode as non-cacheable and has the instruction cache locked (that is, four-way set associative plus two locked levels).

However, it is impossible to have the default instruction cache mode as cacheable and have the instruction cache locked or to have the default instruction cache mode as non-cacheable and have the instruction cache not locked (that is, four-way set associative).

15.2 Software Workarounds

Several potential workarounds involve using access control registers (ACRs) to override the default cache mode for the instruction cache. Note that the 5407 defines ACR2 and ACR3 for instruction memory. The effective cache mode for instruction space as follows:

The instruction local memory bus address is compared with, from highest to lowest RAMBAR0, RAMBAR1, ACR2, and ACR3. If no match is found, the default access attributes in the CACR for the instruction cache are used for that access.

1. To have the functionality of CACR[11] equal 0 and CACR[10] equal 1 (that is, default instruction cache mode as non-cacheable and instruction cache not locked), do the following:
   a) Set CACR[11] equal 0 and CACR[10] equal X (don't care). This gives a default instruction cache mode as cacheable and leaves the instruction cache unlocked. This will get the cache lock function to the desired state.
   b) Set ACR3 to 0x00FFC060. This indicates that any instruction address hit ACR3, and ACR3 has a cache mode of non-cacheable. This overrides the default cache mode of cacheable for all addresses.

2. To have the functionality of CACR[11] equal 1 and CACR[10] equal 0 (default instruction cache mode as cacheable and have the instruction cache locked)
   a) Set CACR[11] equal 1 and CACR[10] equal X (don't care). This gives a default instruction cache mode as non-cacheable and locks the instruction cache. This gets the desired cache lock function.
   b) Set ACR3 to 0x00FFC000. This indicates that instruction addresses hit ACR3, and ACR3 has a cache mode of cacheable. This overrides the default cache mode of non-cacheable for all addresses.

16. DMA writes to UART cause transmission errors

16.1 Description

DMA transfers from memory to the UART do not work properly. The DMA module configured for DMA transfers to the UART:

DCR = 0x6052

Table 3. DCR = 0x6052

Bits	Name	Setting	Description
15	INT	0	No interrupt is generated at the completion of a transfer
14	EEXT	1	Enable external request
13	CS	1	Enable cycle steal
12	AA	0	No accesses are auto-aligned
11–9	BWC	000	DMA has priority
8	SAA	0	Dual address mode
7	S_RW	0	Not valid when SAA=0
6	SINC	1	Source address increments
5–4	SSIZE	01	Size of source bus cycle = byte
3	DINC	0	Destination address does not increment
2–1	DSIZE	01	Size of destination bus cycle = byte
0	START	0	The UART $\overline{\text{DREQ}}$ kicks off the DMA transfer

The UART is configured to assert an internal interrupt when the transmitter is ready (UIMR = 0x01 and UISR = 0x01). $\overline{\text{DREQ}}$ is connected internally to the UART interrupt pin. When the UART asserts an internal interrupt, the DMA initiates a single read/write cycle. The problem stems from $\overline{\text{DREQ}}$ not being negated fast enough thus allowing a second DMA transfer right after the first when the transmit buffer (UTB) of the UART isn't ready yet.

If the first byte hasn't been completely transmitted and the UTB (transmitter buffer) is still full, the second byte transfer has nowhere to go and is discarded. For example if a string = “0123456789” is transferred via the DMA to the transmit buffer of the UART, the UTB receives only “0248” and consequently only 0248 is sent out of the UART.

16.2 Software Workaround

Use the CPU to write to the UART transmitter buffer.

17. DMA single-address access mode and cycle-steal mode do not work together

17.1 Description

An enabled DMA channel performing a single address access and a cycle steal, interprets a request for another channel as a request for itself, performing an errant access. The intended access does occur after the errant access.

17.2 Software Workaround

Don't use single address access mode in cycle-steal mode.

18. DMA channel arbiter hangs in round-robin mode

In the following scenario, the DMA channel arbiter may hang:

- Round-robin mode (MPARK[PARK] = 00)
- DCR[BWC] = 000
- Simultaneous access by DMA and core while the arbiter is selecting the external master (master 0).

When the bus grant is removed or when the SDRAM is performing a PAL (precharge), a hold is asserted to the arbiter, putting it in a park on external master mode. When the hold is removed, if both core and DMA want the bus and DCR[BWC] is set to have priority, the arbiter hangs, causing the external bus to hang.

18.1 Software Workaround

- Set PARK = 01 (park on ColdFire core) or 10 (park on DMA).
- Set BWC bits ≠ 000.

19. DRAM Controller: External master termination is not correct

19.1 Description

The $\overline{TA}$ output from the MCF5407 does not assert properly when an external master is accessing SDRAM address space. For back-to-back reads from the same page, only the first read and last read will be terminated. For a write cycle by an external master, the $\overline{TA}$ asserts 1 bus clock early for all CASL[1:0] encodings.

19.2 Workarounds:

- Disallow external master access to SDRAM using the MCF5407 DRAM controller.
- Provide external termination for SDRAM bus cycles by an external master.

Mfax is a trademark of Motorola, Inc.

Motorola Literature Distribution Centers:

USA/EUROPE: Motorola Literature Distribution; P.O. Box 5405; Denver, Colorado 80217; Tel.: 1-800-441-2447 or 1-303-675-2140;
World Wide Web Address: http://ldc.nmd.com/

JAPAN: Nippon Motorola Ltd SPD, Strategic Planning Office 4-32-1, Nishi-Gotanda Shinagawa-ku, Tokyo 141, Japan Tel.: 81-3-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd Silicon Harbour Centre 2, Dai King Street Tai Po Industrial Estate Tai Po, New Territories, Hong Kong

Mfax™: RMFAX0@email.sps.mot.com; TOUCHTONE 1-602-244-6609; US & Canada ONLY (800) 774-1848;
World Wide Web Address: http://sps.motorola.com/mfax

INTERNET: http://motorola.com/sps

Technical Information: Motorola Inc. SPS Customer Support Center 1-800-521-6274; electronic mail address: crc@wmkmail.sps.mot.com.

Document Comments: FAX (512) 895-2638, Attn: RISC Applications Engineering.

World Wide Web Addresses: http://www.motorola.com/PowerPC
http://www.motorola.com/netcomm
http://www.motorola.com/Coldfire

MOTOROLA

MCF5407