

Motorola Semiconductor Engineering Bulletin

EB310

Using Bus Error Stack Frames to Diagnose CPU32 Released Write Faults

By Charles Melear
Austin, Texas

Introduction

This engineering bulletin describes a methodology that uses bus error (BERR) stack frames to diagnose problems in a memory transaction in the write unit of the CPU32.

When an instruction must perform a write to external memory, the instruction finishes execution as soon as the instruction delivers the address and data of the memory transaction to the write unit. Once the transaction is handed to the write unit, the CPU will begin execution of the next instruction.

If the memory transaction in the write unit faults – for example, is terminated by a BERR – a bus error stack frame will be generated. The BERR will cause the write unit to abort the write cycle. This aborted cycle is referred to as a released write fault. The address of the current instruction will be put into the stack frame, although the current instruction generally is not the instruction that caused the fault.



What to Do When a Bus Error Occurs

When the CPU32 executes an instruction that writes data to external memory, the last part of the instruction is used to actually perform the write. An effective address (EA) is calculated along with the data and is then delivered to the write unit. As soon as the CPU32 has delivered the effective address and data to the write unit, the CPU32 can begin execution of another instruction. As long as the write unit is not needed for a subsequent instruction, the CPU32 can continue executing new instructions while the transaction in the write unit is pending.

The transaction in the write unit can get delayed indefinitely if the transaction is waiting on an external DSACK that never gets asserted. Obviously, this would be an errant condition. In the absence of some type of bus monitor, such a condition could permanently stall the transaction. Usually, the internal bus monitor of the CPU32 derivative is set up to catch excessive external data transfer times. If the bus monitor times out, a BERR will be signaled.

If a bus error occurs, the transaction in the write unit will be released, thus the term released write fault. Once the transaction is released, the transaction cannot be restarted.

There is no guaranteed way to recover from a released write fault. This is because the current instruction being executed may not and probably will not be the instruction that generated the faulted transaction in the write unit.

The occurrence of a released write fault is determined inside the BERR exception handler where the special status word is examined. Depending upon the actual cause of the BERR, one of three sets of information will be put on the stack. This information is detailed in the figures that follow. The exception handler begins by examining bits 15 and 14 of the special status word. These bits will be %00, %01, or %10, depending upon the type of fault that causes the BERR.

The three different types of BERR stack frames are shown in [Figure 1](#), [Figure 2](#), and [Figure 3](#). All BERR stack frames are 12 words long.

```

STATUS REGISTER
RETURN PROGRAM COUNTER HIGH
RETURN PROGRAM COUNTER LOW
1 1 0 0 + VECTOR OFFSET
FAULTED ADDRESS HIGH
FAULTED ADDRESS LOW
DBUF HIGH
DBUF LOW
CURRENT INSTRUCTION PROGRAM COUNTER HIGH
CURRENT INSTRUCTION PROGRAM COUNTER LOW
INTERNAL TRANSFER COUNT REGISTER
0 0 + SPECIAL STATUS WORD
    
```

Figure 1. FORMAT \$C — BERR Stack for Prefetches and Operands

```

STATUS REGISTER
RETURN PROGRAM COUNTER HIGH
RETURN PROGRAM COUNTER LOW
1 1 0 0 + VECTOR OFFSET
FAULTED ADDRESS HIGH
FAULTED ADDRESS LOW
DBUF HIGH
DBUF LOW
CURRENT INSTRUCTION PROGRAM COUNTER HIGH
CURRENT INSTRUCTION PROGRAM COUNTER LOW
INTERNAL TRANSFER COUNT REGISTER
0 1 + SPECIAL STATUS WORD
    
```

Figure 2. FORMAT \$C — BERR Stack on MOVEM Operand

```

STATUS REGISTER
NEXT INSTRUCTION PROGRAM COUNTER HIGH
NEXT INSTRUCTION PROGRAM COUNTER LOW
1 1 0 0 + VECTOR OFFSET
FAULTED ADDRESS HIGH
FAULTED ADDRESS LOW
PRE-EXCEPTION STATUS REGISTER
FAULTED EXECPTION FORMAT/VECTOR WORD
FAULTED INSTRUCTION PROGRAM COUNTER HIGH
FAULTED INSTRUCTION PROGRAM COUNTER LOW
INTERNAL TRANSFER COUNT REGISTER
1 0 + SPECIAL STATUS WORD
    
```

Figure 3. FORMAT \$C — 4- and 6-Word BERR Stack

The example program in **Figure 4** was created to deliberately cause a BERR with the MOVE instruction at location 4114. Specifically, when the CPU32 attempted to write to \$E00000, a bus monitor timeout occurred because no DSACK was returned to indicate a response from the memory system.

When the BERR occurred, a BERR exception was taken. The bus error exception vector was fetched and program execution began at the bus error exception handler. In this case, the BERR exception handler was an infinite loop created by having a BRA instruction simply branching to itself.

```

4100  MOVE.L #$12345678,D0
4106  MOVE.L #$87654321,D1
410C  ADD.L D0,D1
410E  SUBI.L #$48,D1
4114  MOVE.W D0,($E00000).L
411A  CMP.W D0,D1
411C  MOVE.L D1,D1
411E  CLR.B D0
4120  ASL.W #$4,D2
4122  BRA.W $4122
    
```

Figure 4. Program Example 1

The data in **Figure 5** was placed on the stack in response to the BERR.

Stack Data

2FE4	2700	Status Register
2FE6	0000	Return Program Counter High
2FE8	411A	Return Program Counter Low
2FEA	C008	Vector Offset
2FEC	00E0	Faulted Address High
2FEE	0000	Faulted Address Low
2FF0	1234	Data Buffer High
2FF2	5678	Data Buffer Low
2FF4	0000	Current Instruction Pointer High
2FF6	4114	Current Instruction Pointer Low
2FF8	0000	Internal Transfer Count
2FFA	0615	Special Status Word

Figure 5. BERR Stack Frame Data for Program Example 1

In normal practice, the BERR handler will examine bits 15 and 14 of the special status word to determine the type of BERR. According to the three stack frames previously listed, the BERR handler will know what information is on the stack. In the present case, the special status word at location \$2FFA is \$0615 (binary 0000 0110 0001 0101).

The definitions for the bits in the special status word are:

- Bit 15, TP — BERR frame type
- Bit 14, MV — MOVEM in progress
- Bit 13, 0 — Not used
- Bit 12, TR — Trace pending
- Bit 11, B1 — Breakpoint on channel 1
- Bit 10, B0 — Breakpoint on channel 0
- Bit 9, RR — Rerun write cycle after RTE
- Bit 8, RM — Faulted cycle was read-modify-write
- Bit 7, IN — Instruction
- Bit 6, RW — Faulted cycle was a read (1) or a write (0)
- Bit 5, LG — Original operand size was longword
- Bits 4 and 3, SIZ — Remaining size of faulted bus cycle
- Bits 2, 1, and 0, FUNC — Function code of faulted bus cycle

Bit 9 indicates that the faulted transaction was a released write fault. There will be enough information on the stack to determine how to complete the transaction. However, as previously stated, it may not be possible to determine which instruction was responsible for starting the memory transaction.

In the present case, bits 15 and 14 of the special status word are %00. This indicates that this stack frame is for a prefetch or operand fetch. Upon further examination, the location of the current instruction is \$0000 4114, and the data is in locations \$2FF4 and \$2FF6. The data to be written to memory in the released write was \$12345678 and is located in locations \$2FF0 and \$2FF2 on the stack. The faulted address is \$0E0000 and is located in locations \$2FEC and \$2FEE.

In the simple example offered here, it is somewhat obvious (although not absolute) that the MOVE.W D0,(\$E0000).L at location \$4114 was responsible for creating the problematic address in the write unit. The data in D0 was \$12345678 and the effective destination address was \$E0 0000.

In more complex programs, it may not be possible to determine which instruction caused the errant transaction which was delivered to the write unit.

The current instruction program counter high/low points to the MOVE instruction at \$4114. In this case, this is the instruction that generated the errant transaction delivered to the write unit. The return program counter points to the instruction following the MOVE at \$411A.

The same program was run with the same initial conditions except that the number of wait states for the RAM where the program and the stack memory was located was changed from zero wait states to 13 wait states.

Now, the stack data is:

2708	Status Register
0000	Return Program Counter High
411e	Return Program Counter Low
c008	Vector Offset
00E0	Faulted Address High
0000	Faulted Address Low
1234	Data Buffer High
5678	Data Buffer Low
0000	Current Instruction Pointer High
411C	Current Instruction Pointer Low
0000	Internal Transfer Count
0215	Special Status Word

Figure 6. BERR Stack Frame Data for Programming Example 1 Using 13 Wait States

The important differences between this and the first example are:

- The return program counter is \$0000411E instead of \$0000411A.
- The current instruction program counter is \$0000411C instead of \$00004114.

This shows that wait states can affect the number of instructions that get executed between the time that the memory transaction is delivered to the write unit and when the BERR actually occurs. By increasing the number of wait states from 0 to 13, the CPU32 was able to run additional instructions before the BERR occurred from the memory transaction in the write unit.

A significant number of instructions get executed between the instruction generating the memory transaction and the occurrence of the BERR for the faulted transaction.

Consider this program:

```

4100      MOVE.L #$12345678,D0
4106      MOVE.L #$87654321,D1
410C      ADD.L D0,D1
410E      SUBI.L #$48,D1
4114      MOVE.W D0,($E00000).L
411A      BRA $4400
.....
.....
4400      ADD.L D0,D1
4402      SUB.L D2,D3
4404      MOVE.L #$5A5A5A5A,D2
440A      MOVE.L #$A5A5A5A5,D3
  
```

Figure 7. Programming Example 2

Depending upon the number of wait states per memory transaction and the number of cycles allowed before a bus monitor timeout, the BERR stack frame for **Figure 7** would be identical to the stack frame in **Figure 6** except that the address of the current instruction pointer and the return program counter would be different. If the return program counter contained \$0000 4404, it would be impossible to determine that the instruction at location \$0000 4114 is the instruction that is ultimately responsible for the BERR. In fact, the same is true for **Figure 6**, that is, it is not absolute that the instruction at \$0000 4114 is the offending instruction.

However, for debugging purposes, the use of BERR stack frames can yield significant clues as to where problematic areas of a program are located. For instance, many times the offending address and data in the write unit can be generated only by a few (or possibly one) instructions. When mechanical problems, such as worn sockets or cold solder joints, are causing problems, generally, the return program counter will not show any repeatable pattern. Thus, the data in many stack frames will all be different.


Conclusion

From the data in the BERR stack frame, it is possible to reconstruct an errant memory access. A BERR handler routine can attempt to retry (or modify and then retry) the memory transaction.

It is generally not possible to determine absolutely which instruction generated the memory transaction that caused the BERR generated by a released write fault.

By studying BERR stack frames, the location and cause of programming/hardware problems can be localized and sometimes pinpointed to certain areas.

Once the user is proficient in using BERR stack frames, the same knowledge can be used to determine other types of problems. For instance, faults in interrupt routines can cause stack overflows, which many times will result in a BERR. Software errors that cause invalid effective address calculations can also result in a memory transaction that will cause a bus error. Because the BERR stack frames give the faulted memory address and data, etc., effective program and hardware debugging can be implemented to locate problematic areas.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217.
1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Motorola Japan Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinagawa-Ku, Tokyo, Japan, 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate,
Tai Po, New Territories, Hong Kong, 852-26629298

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; <http://sps.motorola.com/mfax/>;
TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: <http://motorola.com/sps/>

Mfax is a trademark of Motorola, Inc.



MOTOROLA

© Motorola, Inc., 1999

EB310/D

**For More Information On This Product,
Go to: www.freescale.com**