Case Study: Debugging Traps on TriCore™ AURIX™

2025-09-15 - Comments (0) - Debug

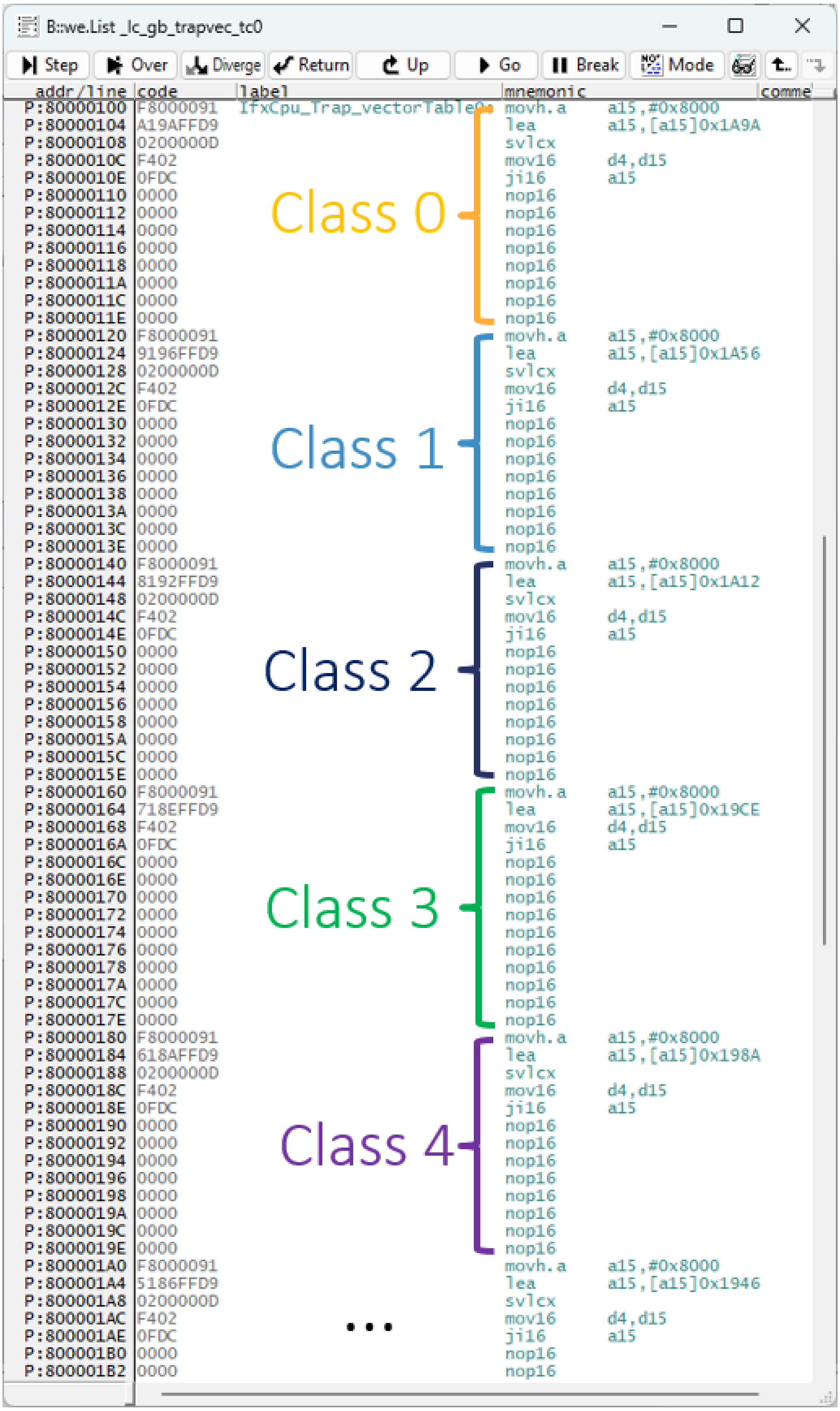
Background Information: Traps in TriCore™ AURIX™

Traps in the TriCore™ AURIX™ architecture are exceptional events that can be caused by conditions such as:

- Instruction Exception
- Illegal Memory accesses, e.g.:
  - Result from attempts to access non mapped memory regions or peripherals not yet initialized
  - Bus transactions that lead to ECC faults
- Non-Maskable Interrupt (NMI)

Traps are always active; they cannot be masked or disabled by software. The trap vector handler is stored in code memory, and the **BTV (Base Trap Vector)** register specifies the base address of the trap vector table. The contents of this register can be inspected in the `Register.view` window.

The TriCore architecture defines **eight general trap classes**, each with its own dedicated handler. The trap class determines the offset of the corresponding trap handler in program memory, relative to the base address specified in the BTV register.



```
  addr/line  code      label                    mnemonic
 P:80000100 F8000091  IfxCpu_Trap_vectorTable0: movh.a    a15,#0x8000
 P:80000104 A19AFFD9                            lea       a15,[a15]0x1A9A
 P:80000108 0200000D                            svlcx
 P:8000010C F402                                mov16     d4,d15
 P:8000010E 0FDC                                ji16      a15
 P:80000110 0000                                nop16
 P:80000112 0000                                nop16
 P:80000114 0000                                nop16
 P:80000116 0000                                nop16
 P:80000118 0000                                nop16
 P:8000011A 0000                                nop16
 P:8000011C 0000                                nop16
 P:8000011E 0000                                nop16
 P:80000120 F8000091                            movh.a    a15,#0x8000
 P:80000124 9196FFD9                            lea       a15,[a15]0x1A56
 P:80000128 0200000D                            svlcx
 P:8000012C F402                                mov16     d4,d15
 P:8000012E 0FDC                                ji16      a15
 P:80000130 0000                                nop16
 P:80000132 0000                                nop16
 P:80000134 0000                                nop16
 P:80000136 0000                                nop16
 P:80000138 0000                                nop16
 P:8000013A 0000                                nop16
 P:8000013C 0000                                nop16
 P:8000013E 0000                                nop16
 P:80000140 F8000091                            movh.a    a15,#0x8000
 P:80000144 8192FFD9                            lea       a15,[a15]0x1A12
 P:80000148 0200000D                            svlcx
 P:8000014C F402                                mov16     d4,d15
 P:8000014E 0FDC                                ji16      a15
 P:80000150 0000                                nop16
 P:80000152 0000                                nop16
 P:80000154 0000                                nop16
 P:80000156 0000                                nop16
 P:80000158 0000                                nop16
 P:8000015A 0000                                nop16
 P:8000015C 0000                                nop16
 P:8000015E 0000                                nop16
 P:80000160 F8000091                            movh.a    a15,#0x8000
 P:80000164 718EFFD9                            lea       a15,[a15]0x19CE
 P:80000168 F402                                mov16     d4,d15
 P:8000016A 0FDC                                ji16      a15
 P:8000016C 0000                                nop16
 P:8000016E 0000                                nop16
 P:80000170 0000                                nop16
 P:80000172 0000                                nop16
 P:80000174 0000                                nop16
 P:80000176 0000                                nop16
 P:80000178 0000                                nop16
 P:8000017A 0000                                nop16
 P:8000017C 0000                                nop16
 P:8000017E 0000                                nop16
 P:80000180 F8000091                            movh.a    a15,#0x8000
 P:80000184 618AFFD9                            lea       a15,[a15]0x198A
 P:80000188 0200000D                            svlcx
 P:8000018C F402                                mov16     d4,d15
 P:8000018E 0FDC                                ji16      a15
 P:80000190 0000                                nop16
 P:80000192 0000                                nop16
 P:80000194 0000                                nop16
 P:80000196 0000                                nop16
 P:80000198 0000                                nop16
 P:8000019A 0000                                nop16
 P:8000019C 0000                                nop16
 P:8000019E 0000                                nop16
 P:800001A0 F8000091                            movh.a    a15,#0x8000
 P:800001A4 5186FFD9                            lea       a15,[a15]0x1946
 P:800001A8 0200000D                            svlcx
 P:800001AC F402                                mov16     d4,d15
 P:800001AE 0FDC                                ji16      a15
 P:800001B0 0000                                nop16
 P:800001B2 0000                                nop16
```

Each trap is assigned a unique **Trap Identification Number (TIN)**. When a trap occurs, the TIN is automatically stored in register **D15**, allowing the Trap Service Routine (TSR) to identify the trap and take appropriate action in the application software.

Trap Types:

- **Source Classification:**
    - **Hardware traps**: Generated in response to exception conditions detected by the hardware (e.g., illegal instruction or memory protection traps).
    - **Software traps**: Intentionally generated by executing a system call or an assertion instruction.
- **Timing Classification:**
    - **Synchronous traps**: Occur during the execution (or attempted execution) of a specific instruction. The causing instruction is known precisely, and the trap is serviced immediately before execution continues.
    - **Asynchronous traps**: Triggered by hardware conditions detected externally and signaled back to the core. The exact instruction that caused the condition may not be identifiable since the CPU stops at a random location and the displayed instruction is thus not related to the trap.

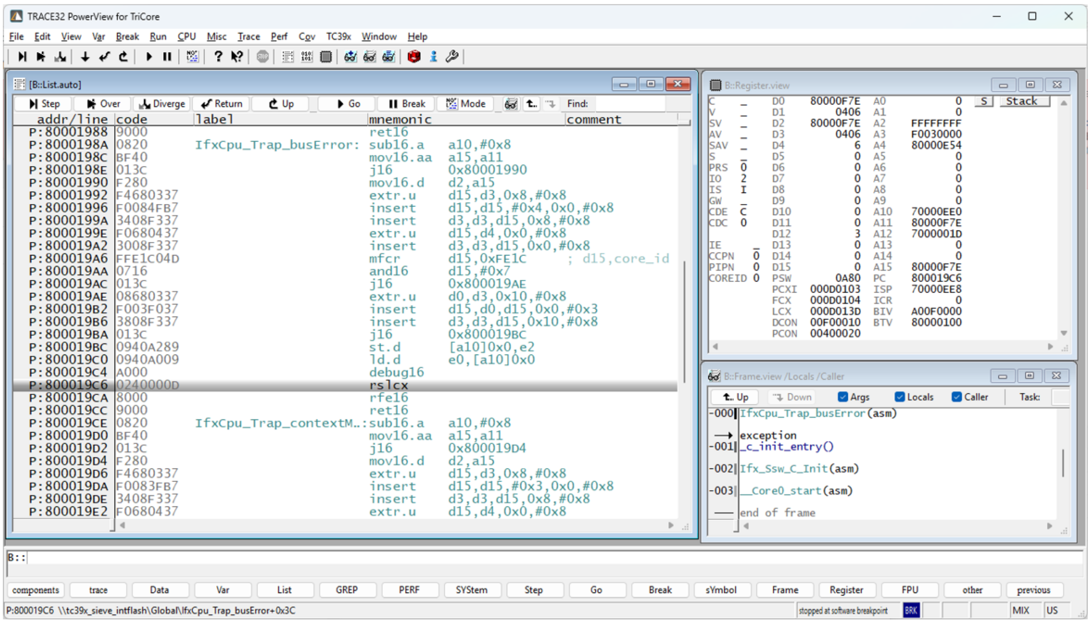
For more information, refer to the *Core Architecture Manual*.

Case Study - Trap Debugging

In this case study, we demonstrate how to debug a trap using **TRACE32 PowerView** on a **TriBoard equipped with a TC397XE**.

Initial Observations

We start with a trap condition where the symbolic information shows that the application is stopped at a **Bus Error Trap**.



TRACE32 PowerView for TriCore
File Edit View Var Break Run CPU Misc Trace Perf Cov TC39x Window Help
[B::List.auto]
Step Over Diverge Return Up Go Break Mode Find:
B::Register.view
S Stack
B::Frame.view /Locals /Caller
Up Down
Task:
components trace Data Var List GREP PERF SYStem Step Go Break sYmbol Frame Register FPU other previous

```
  addr/line code      label                     mnemonic                         comment
  P:80001988 9000                                ret16
  P:8000198A 0820      IfxCpu_Trap_busError:     sub16.a   a10,#0x8
  P:8000198C BF40                                mov16.aa  a15,a11
  P:8000198E 013C                                j16       0x80001990
  P:80001990 F280                                mov16.d   d2,a15
  P:80001992 F4680337                            extr.u    d15,d3,0x8,#0x8
  P:80001996 F0084FB7                            insert    d15,d15,#0x4,0x0,#0x8
  P:8000199A 3408F337                            insert    d3,d3,d15,0x8,#0x8
  P:8000199E F0680437                            extr.u    d15,d4,0x0,#0x8
  P:800019A2 3008F337                            insert    d3,d3,d15,0x0,#0x8
  P:800019A6 FFE1C04D                            mfcr      d15,0xFE1C       ; d15,core_id
  P:800019AA 0716                                and16     d15,#0x7
  P:800019AC 013C                                j16       0x800019AE
  P:800019AE 08680337                            extr.u    d0,d3,0x10,#0x8
  P:800019B2 F003F037                            insert    d15,d0,d15,0x0,#0x3
  P:800019B6 3808F337                            insert    d3,d3,d15,0x10,#0x8
  P:800019BA 013C                                j16       0x800019BC
  P:800019BC 0940A289                            st.d      [a10]0x0,e2
  P:800019C0 0940A009                            ld.d      e0,[a10]0x0
  P:800019C4 A000                                debug16
  P:800019C6 0240000D                            rslcx
  P:800019CA 8000                                rfe16
  P:800019CC 9000                                ret16
  P:800019CE 0820      IfxCpu_Trap_contextM..:sub16.a   a10,#0x8
  P:800019D0 BF40                                mov16.aa  a15,a11
  P:800019D2 013C                                j16       0x800019D4
  P:800019D4 F280                                mov16.d   d2,a15
  P:800019D6 F4680337                            extr.u    d15,d3,0x8,#0x8
  P:800019DA F0083FB7                            insert    d15,d15,#0x3,0x0,#0x8
  P:800019DE 3408F337                            insert    d3,d3,d15,0x8,#0x8
  P:800019E2 F0680437                            extr.u    d15,d4,0x0,#0x8
```

```
C       _    D0    80000F7E  A0          0
V       _    D1        0406  A1          0
SV      _    D2    80000F7E  A2   FFFFFFFF
AV      _    D3        0406  A3   F0030000
SAV     _    D4           6  A4   80000E54
S       _    D5           0  A5          0
PRS     0    D6           0  A6          0
IO      2    D7           0  A7          0
IS      I    D8           0  A8          0
GW      _    D9           0  A9          0
CDE     C    D10          0  A10  70000EE0
CDC     0    D11          0  A11  80000F7E
             D12          3  A12  7000001D
IE        _  D13          0  A13         0
CCPN      0  D14          0  A14         0
PIPN      0  D15          0  A15  80000F7E
COREID    0  PSW       0A80  PC   800019C6
             PCXI  000D0103  ISP  70000EE8
             FCX   000D0104  ICR         0
             LCX   000D013D  BIV  A00F0000
             DCON  00F00010  BTV  80000100
             PCON  00400020
```

☑ Args ☑ Locals ☑ Caller

```
-000 IfxCpu_Trap_busError(asm)
 →  exception
-001 _c_init_entry()
-002 Ifx_Ssw_C_Init(asm)
-003 __Core0_start(asm)
 ──  end of frame
```

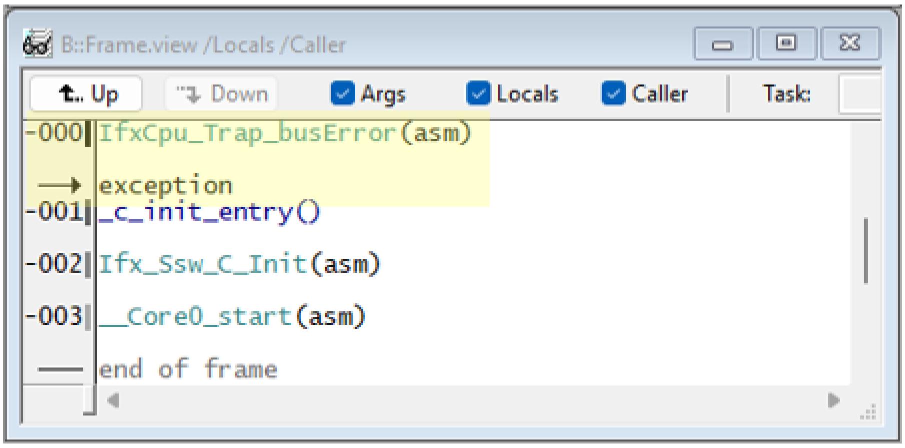
B::

P:800019C6 \\tc39x_sieve_intflash\Global\IfxCpu_Trap_busError+0x3C   stopped at software breakpoint   BRK   MIX   US

The TRACE32 PowerView status bar indicates that the target is *“stopped at a software breakpoint.”* The **`List` window** confirms that the program counter points to an instruction immediately after debug16.

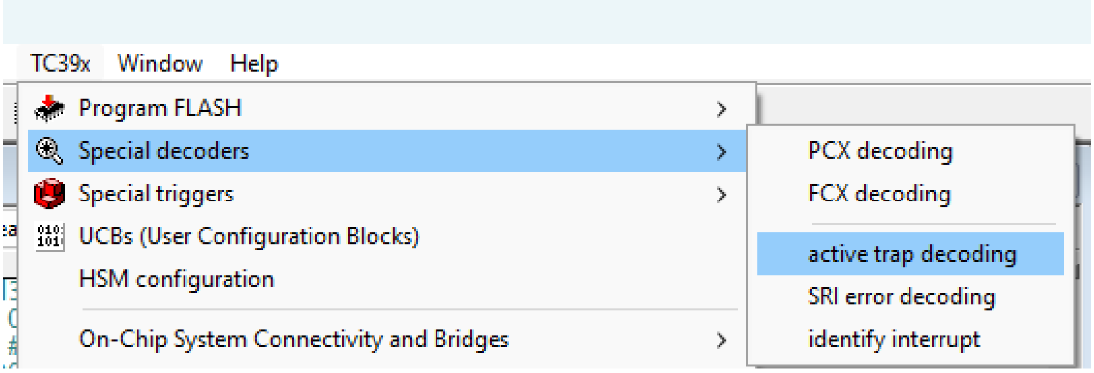
Note
Debug instructions are typically inserted into error-handling routines. When a debugger is connected, they halt the core for inspection. Without a debugger, they behave as NOP instructions.

Examining the **`Frame.view`** window reveals an exception followed by a call to a trap-handling function.



```
-000 IfxCpu_Trap_busError(asm)

 ⟶  exception
-001 _c_init_entry()

-002 Ifx_Ssw_C_Init(asm)

-003 __Core0_start(asm)

 ── end of frame
```

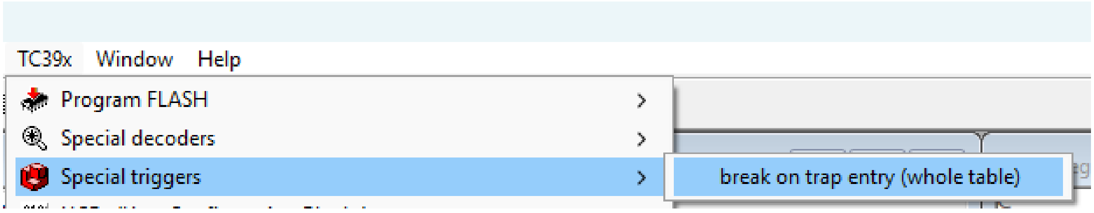
Using TRACE32 Trap Decoding
TRACE32 PowerView provides a menu for identifying the reason for a trap:
TC39x > Special decoders > active trap decoding.



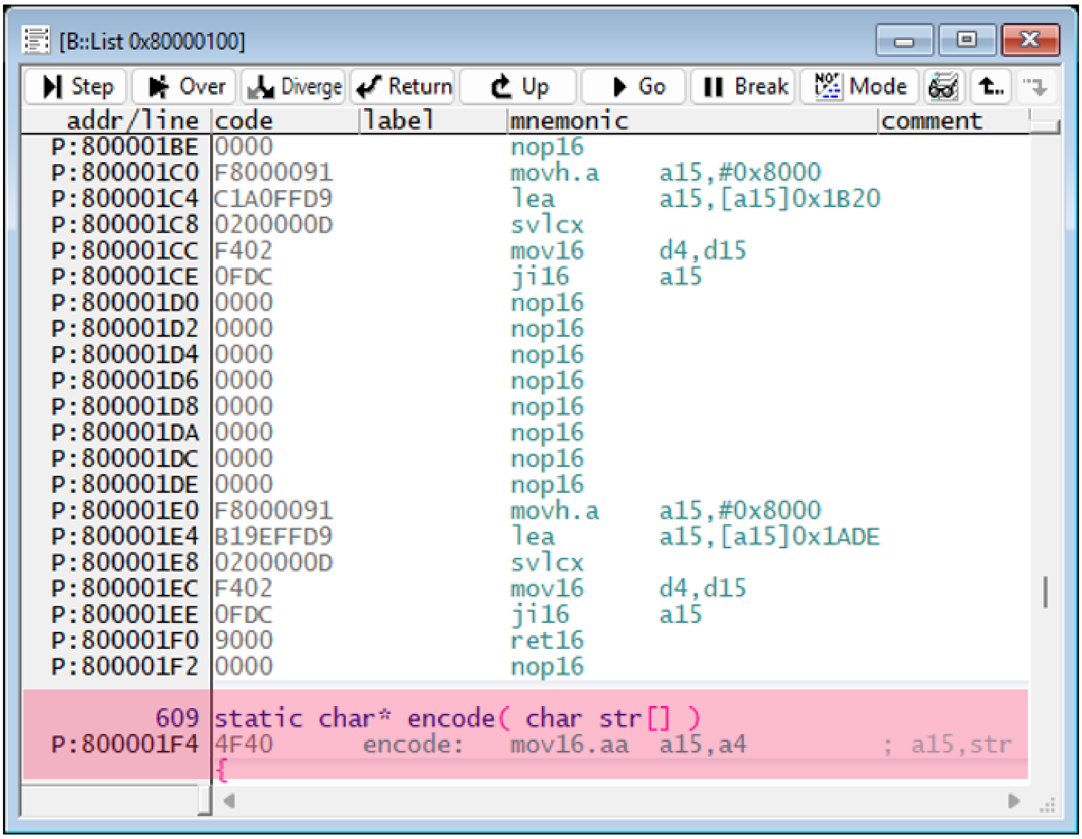
If the core is halted *inside* the Trap Service Routine (not at the trap vector itself), the AREA window may display *“No exception detected.”* This is expected because the core has already moved past the trap vector.

To decode the trap reason, the target must be halted **at the trap vector itself**. TRACE32 PowerView provides a menu for setting a program breakpoint across the trap vector range:

TC39x > Special triggers > break on trap entry (whole table).



Care must be taken, as compilers may insert regular application code (e.g., the encode function) into unused bytes of the trap vector.

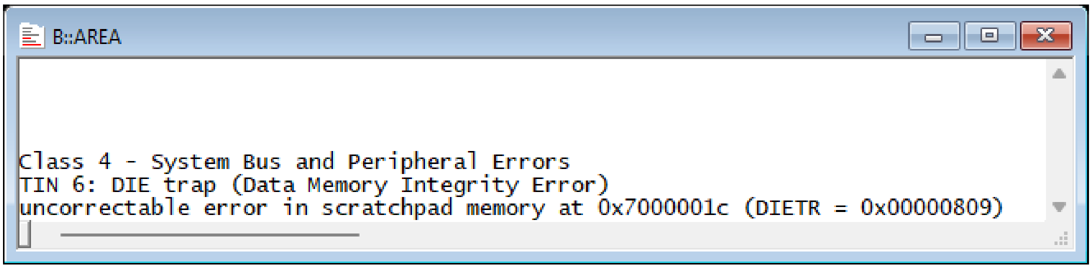


```
  addr/line  code       label      mnemonic                comment
 P:800001BE  0000                  nop16
 P:800001C0  F8000091              movh.a    a15,#0x8000
 P:800001C4  C1A0FFD9              lea       a15,[a15]0x1B20
 P:800001C8  0200000D              svlcx
 P:800001CC  F402                  mov16     d4,d15
 P:800001CE  0FDC                  ji16      a15
 P:800001D0  0000                  nop16
 P:800001D2  0000                  nop16
 P:800001D4  0000                  nop16
 P:800001D6  0000                  nop16
 P:800001D8  0000                  nop16
 P:800001DA  0000                  nop16
 P:800001DC  0000                  nop16
 P:800001DE  0000                  nop16
 P:800001E0  F8000091              movh.a    a15,#0x8000
 P:800001E4  B19EFFD9              lea       a15,[a15]0x1ADE
 P:800001E8  0200000D              svlcx
 P:800001EC  F402                  mov16     d4,d15
 P:800001EE  0FDC                  ji16      a15
 P:800001F0  9000                  ret16
 P:800001F2  0000                  nop16

        609  static char* encode( char str[] )
 P:800001F4  4F40       encode:    mov16.aa  a15,a4        ; a15,str
             {
```

In this example, the program breakpoint must be adapted accordingly:

`Break.Set IfxCpu_Trap_vectorTable0++0xF3 /Program /Onchip`

When the core is halted correctly within the trap vector range, the "**active trap decoding**" menu will show details such as the **trap class, TIN,** and other trap-specific information.

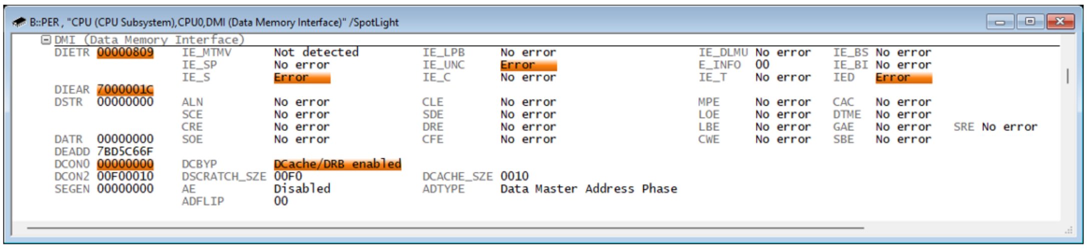


```
Class 4 - System Bus and Peripheral Errors
TIN 6: DIE trap (Data Memory Integrity Error)
uncorrectable error in scratchpad memory at 0x7000001c (DIETR = 0x00000809)
```

Accessing Additional Trap Details

While the *Core Architecture Manual* describes general trap mechanisms, implementation details are documented in the *Family User's Manual*. For example TC3xx User's Manual, states that more detailed information about DIE traps are to be extracted from the **DIEAR** (Data Integrity Error Address Register) and **DIETR** (Data Integrity Error Trap Register).

Using the peripheral view we can get more insights about the error e.g. the address of the memory access causing the trap!



B::PER , "CPU (CPU Subsystem),CPU0,DMI (Data Memory Interface)" /SpotLight

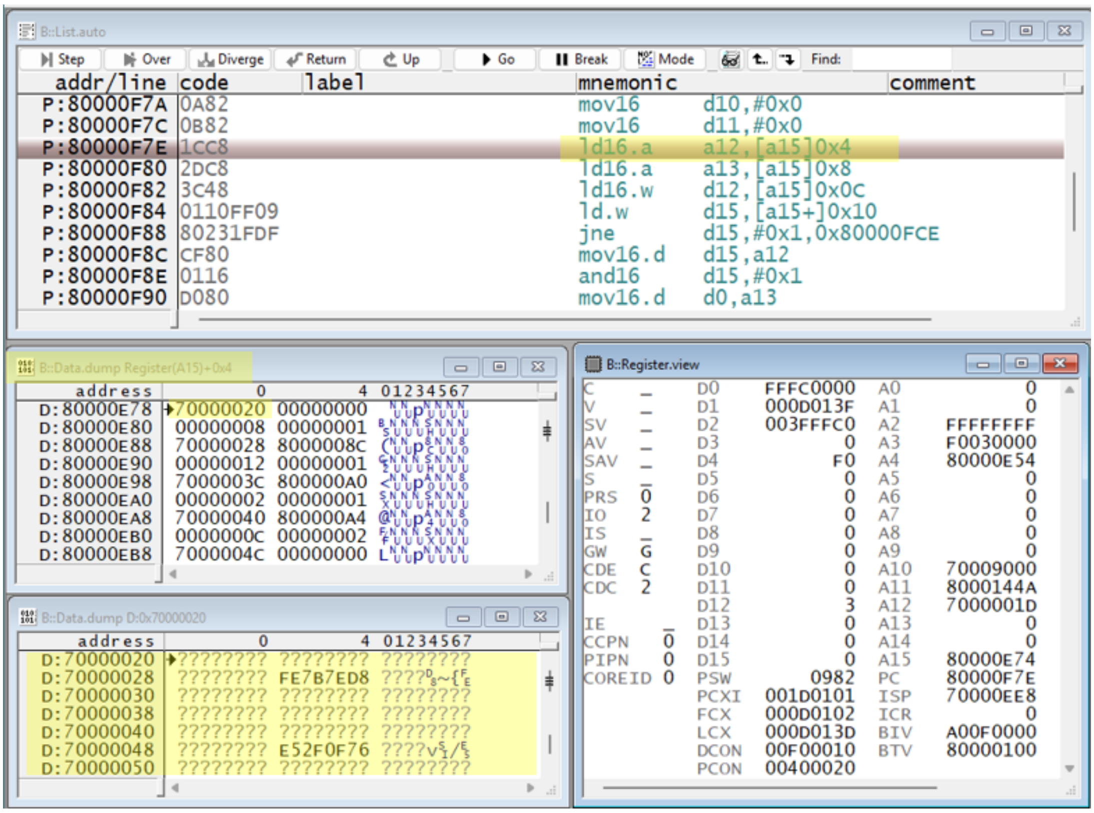
```
DMI (Data Memory Interface)
  DIETR 00000809   IE_MTMV      Not detected        IE_LPB      No error           IE_DLMU No error   IE_BS No error
                   IE_SP        No error            IE_UNC      Error              E_INFO  00         IE_BI No error
                   IE_S         Error               IE_C        No error           IE_T    No error   IED   Error
  DIEAR 7000001C
  DSTR  00000000   ALN          No error            CLE         No error           MPE     No error   CAC   No error
                   SCE          No error            SDE         No error           LOE     No error   DTME  No error
                   CRE          No error            DRE         No error           LBE     No error   GAE   No error   SRE No error
  DATR  00000000   SOE          No error            CFE         No error           CWE     No error   SBE   No error
  DEADD 7BD5C66F
  DCON0 00000000   DCBYP        DCache/DRB enabled
  DCON2 00F00010   DSCRATCH_SZE 00F0                DCACHE_SZE  0010
  SEGEN 00000000   AE           Disabled            ADTYPE      Data Master Address Phase
                   ADFLIP       00
```

Using the Peripheral view in TRACE32, we can observe:

- DIETR : CPUx Data Integrity Error Trap Register
  - IED: Data integrity error condition detected
  - IE_S: Integrity Error - Scratchpad Memory
  - Dual Bit Error Detected
- DIEAR: Data Integrity Error Address Register
  - The access triggering the trap is actually **0x7000001C**

For **synchronous** traps, the **Stack Frame window** can also provide valuable insights such as the exact instruction that triggered the trap.
However, since a DIE trap is **asynchronous**, the direct link to the instruction that triggered it is lost. The trap may occur several instructions after the offending instruction executed. Using `Frame.Up` in such cases may lead to misleading conclusions.

In the following screenshot, the stack unwinding shows that the DIE trap occurred while the CPU was trying to read from the address **0x70000020**. The Data Integrity Error Address Register indicates that the error was triggered by a prior read access to a different address (**0x7000001C).**



```
addr/line  code      label   mnemonic
P:80000F7A 0A82              mov16      d10,#0x0
P:80000F7C 0B82              mov16      d11,#0x0
P:80000F7E 1CC8              ld16.a     a12,[a15]0x4
P:80000F80 2DC8              ld16.a     a13,[a15]0x8
P:80000F82 3C48              ld16.w     d12,[a15]0x0C
P:80000F84 0110FF09          ld.w       d15,[a15+]0x10
P:80000F88 80231FDF          jne        d15,#0x1,0x80000FCE
P:80000F8C CF80              mov16.d    d15,a12
P:80000F8E 0116              and16      d15,#0x1
P:80000F90 D080              mov16.d    d0,a13
```

B::Data.dump Register(A15)+0x4

```
     address         0        4 01234567
  D:80000E78 >70000020 00000000
  D:80000E80  00000008 00000001
  D:80000E88  70000028 8000008C
  D:80000E90  00000012 00000001
  D:80000E98  7000003C 800000A0
  D:80000EA0  00000002 00000001
  D:80000EA8  70000040 800000A4
  D:80000EB0  0000000C 00000002
  D:80000EB8  7000004C 00000000
```

B::Data.dump D:0x70000020

```
     address         0        4 01234567
  D:70000020 >???????? ???????? ????????
  D:70000028  ???????? FE7B7ED8 ????~{
  D:70000030  ???????? ???????? ????????
  D:70000038  ???????? ???????? ????????
  D:70000040  ???????? ???????? ????????
  D:70000048  ???????? E52F0F76 ????v/
  D:70000050  ???????? ???????? ????????
```

B::Register.view

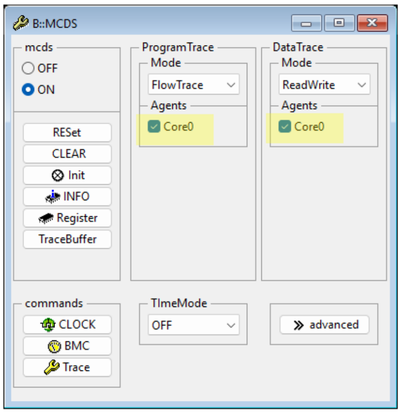
```
C       _    D0     FFFC0000   A0          0
V       _    D1     000D013F   A1          0
SV      _    D2     003FFFC0   A2   FFFFFFFF
AV      _    D3            0   A3   F0030000
SAV     _    D4           F0   A4   80000E54
S       _    D5            0   A5          0
PRS     0    D6            0   A6          0
IO      2    D7            0   A7          0
IS      _    D8            0   A8          0
GW      G    D9            0   A9          0
CDE     C    D10           0   A10  70009000
CDC     2    D11           0   A11  8000144A
             D12           3   A12  7000001D
IE        _  D13           0   A13         0
CCPN      0  D14           0   A14         0
PIPN      0  D15           0   A15  80000E74
COREID    0  PSW        0982   PC   80000F7E
             PCXI   001D0101   ISP  70000EE8
             FCX    000D0102   ICR         0
             LCX    000D013D   BIV  A00F0000
             DCON   00F00010   BTV  80000100
             PCON   00400020
```

Using Trace

Another powerful debugging technique is to use **MCDS trace**.

Since this trap was caused by a memory access, tracing both program flow and data accesses gives a clearer picture.

In the **Trace.List** window, a **trap marker** appears shortly after a byte write access to the DSPR (Data Scratch-Pad RAM) of TriCore0.

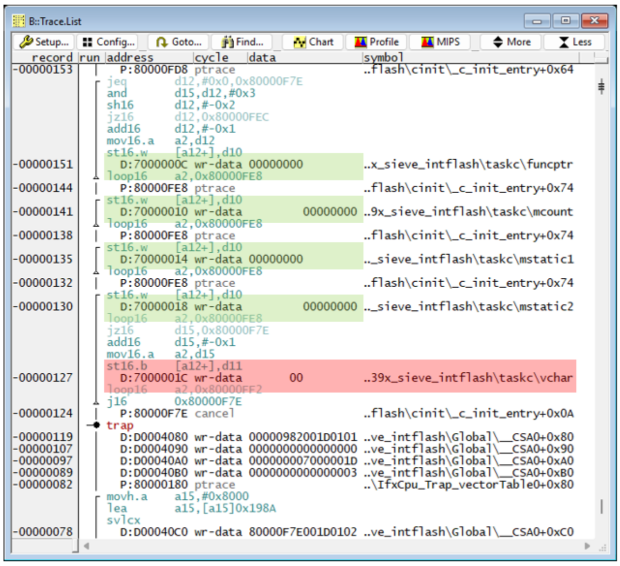


mcds
○ OFF
● ON

ProgramTrace
Mode: FlowTrace
Agents: ☑ Core0

DataTrace
Mode: ReadWrite
Agents: ☑ Core0

TImeMode: OFF



```
    record run address    cycle   data             symbol
 -00000153       P:80000FD8 ptrace                  ..flash\cinit\_c_init_entry+0x64
             jeq        d12,#0x0,0x80000F7E
             and        d15,d12,#0x3
             sh16       d12,#-0x2
             jz16       d12,0x80000FEC
             add16      d12,#-0x1
             mov16.a    a2,d12
             st16.w     [a12+],d10
 -00000151       D:7000000C wr-data 00000000        ..x_sieve_intflash\taskc\funcptr
             loop16     a2,0x80000FE8
 -00000144       P:80000FE8 ptrace                  ..flash\cinit\_c_init_entry+0x74
             st16.w     [a12+],d10
 -00000141       D:70000010 wr-data         00000000 ..9x_sieve_intflash\taskc\mcount
             loop16     a2,0x80000FE8
 -00000138       P:80000FE8 ptrace                  ..flash\cinit\_c_init_entry+0x74
             st16.w     [a12+],d10
 -00000135       D:70000014 wr-data 00000000        ..._sieve_intflash\taskc\mstatic1
             loop16     a2,0x80000FE8
 -00000132       P:80000FE8 ptrace                  ..flash\cinit\_c_init_entry+0x74
             st16.w     [a12+],d10
 -00000130       D:70000018 wr-data         00000000 ..._sieve_intflash\taskc\mstatic2
             loop16     a2,0x80000FE8
             jz16       d15,0x80000F7E
             add16      d15,#-0x1
             mov16.a    a2,d15
             st16.b     [a12+],d11
 -00000127       D:7000001C wr-data       00        ..39x_sieve_intflash\taskc\vchar
             loop16     a2,0x80000FF2
             j16        0x80000F7E
 -00000124       P:80000F7E cancel                  ..flash\cinit\_c_init_entry+0x0A
             trap
 -00000119       D:D0004080 wr-data 00000982001D0101 ..ve_intflash\Global\__CSA0+0x80
 -00000107       D:D0004090 wr-data 0000000000000000 ..ve_intflash\Global\__CSA0+0x90
 -00000097       D:D00040A0 wr-data 000000007000001D ..ve_intflash\Global\__CSA0+0xA0
 -00000089       D:D00040B0 wr-data 0000000000000003 ..ve_intflash\Global\__CSA0+0xB0
 -00000082       P:80000180 ptrace                  ..\IfxCpu_Trap_vectorTable0+0x80
             movh.a     a15,#0x8000
             lea        a15,[a15]0x198A
             svlcx
 -00000078       D:D00040C0 wr-data 80000F7E001D0102 ..ve_intflash\Global\__CSA0+0xC0
```

Root Cause & Solution

The **DSPR (Data Scratchpad RAM)** is **ECC protected** and must be initialized before any read operation. Initialization can be performed either by software, or automatically by hardware (via `UCB_DFLASH.PROCONRAM`).

For **half-word or larger write operations**, ECC bits are pre-calculated and written alongside the data. However, for **byte write operations**, the transaction is internally transformed into a half-word read–modify–write sequence in the DMI module. This caused the detection of **uncorrectable memory integrity errors**.

Solution:
The issue is resolved by enabling **RAM initialization** through the Startup Software (SSW – the boot ROM) in `UCB_DFLASH`.

Note
If you are interested in the demo files used in this case study, please contact Lauterbach Support.