CPU_Trap_Recognition_1 for KIT_AURIX_TC397_TFT

TRAP error recognition and reaction

AURIX™ TC3xx Microcontroller Training
V1.0.0

Please read the Important Notice and Warnings at the end of this document

Scope of work

This example shows how to identify the root cause of a trap.

The tutorial describes what types of traps are supported by the AURIX™ microcontroller, their root causes and how to identify them. AURIX™ architecture supports different types of traps. Three different traps can be provoked with this example and the tutorial guides the user through the needed steps to observe the root cause of each trap.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Introduction

- A trap occurs as a result of an event such as a Non-Maskable Interrupt (NMI), an instruction exception, a memory management exception or an illegal access. Traps are always active; they cannot be disabled by software.

- The TriCore™ architecture specifies eight general classes for traps. Each trap class has its own trap handler. Within each class, specific traps are distinguished by a Trap Identification Number (TIN).

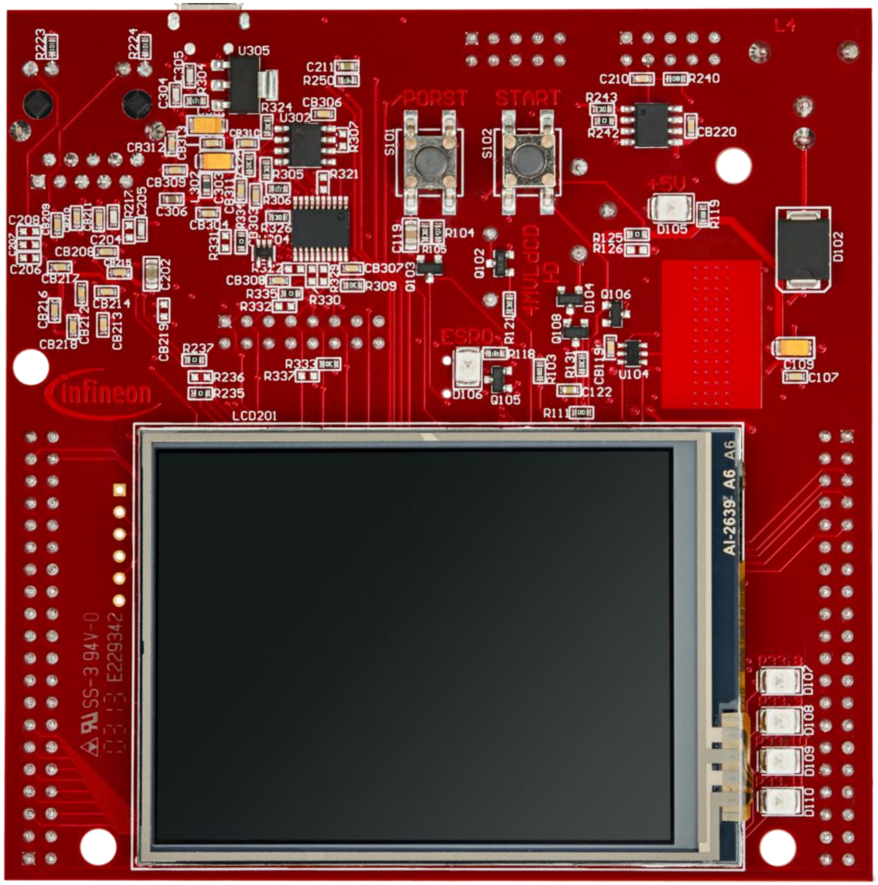
- Traps can be further classified as synchronous or asynchronous, and as hardware or software generated.

- Three different combinations of trap types are supported:
  - Synchronous and hardware generated
  - Asynchronous and hardware generated
  - Synchronous and software generated

Copyright © Infineon Technologies AG 2020. All rights reserved.

Hardware setup

This code example has been developed for the board KIT_A2G_TC397_5V_TFT.



Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Supported traps

The following table provides an overview about all supported traps and their types:

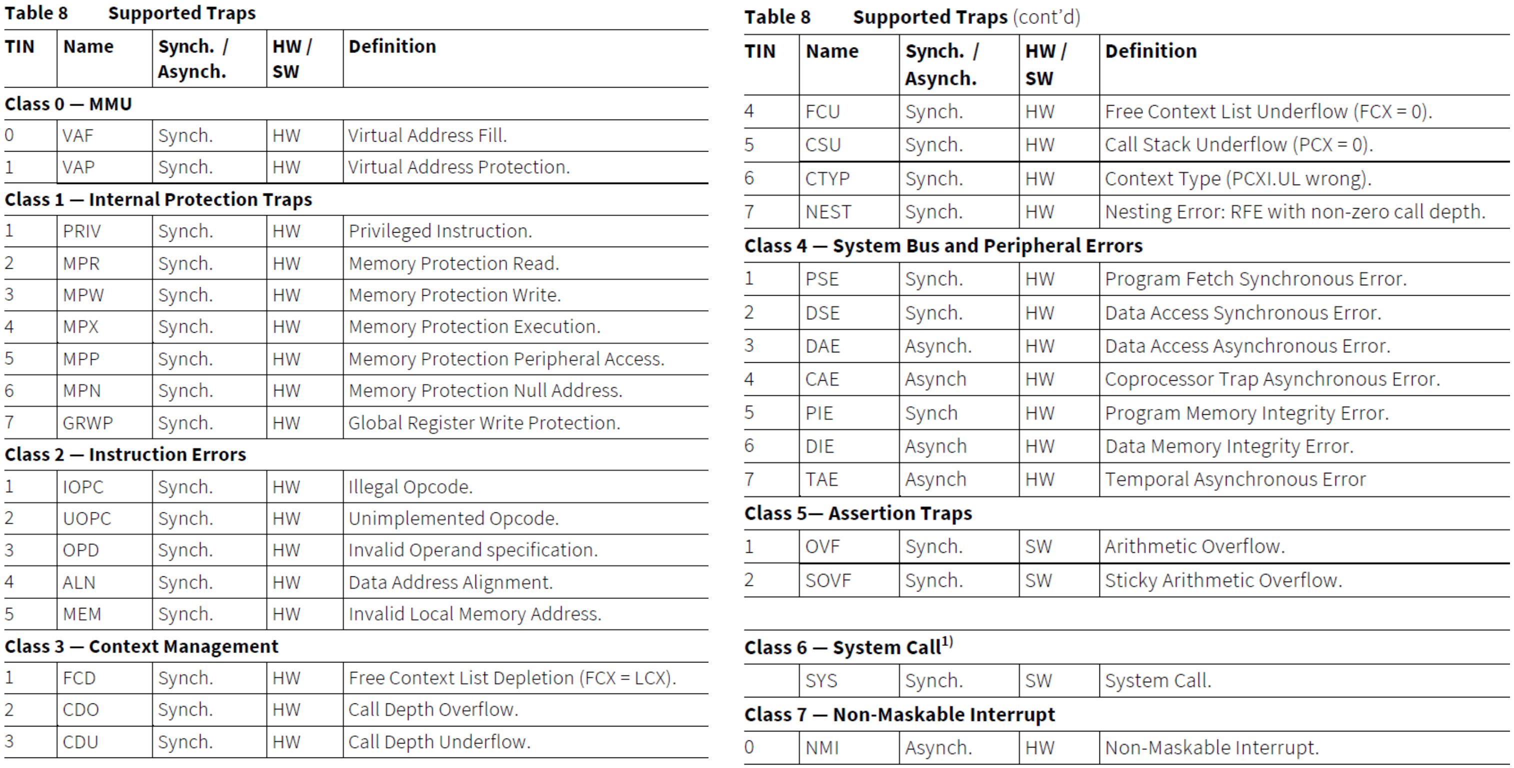


Table 8 Supported Traps

TIN	Name	Synch. / Asynch.	HW / SW	Definition
Class 0 — MMU				
0	VAF	Synch.	HW	Virtual Address Fill.
1	VAP	Synch.	HW	Virtual Address Protection.
Class 1 — Internal Protection Traps				
1	PRIV	Synch.	HW	Privileged Instruction.
2	MPR	Synch.	HW	Memory Protection Read.
3	MPW	Synch.	HW	Memory Protection Write.
4	MPX	Synch.	HW	Memory Protection Execution.
5	MPP	Synch.	HW	Memory Protection Peripheral Access.
6	MPN	Synch.	HW	Memory Protection Null Address.
7	GRWP	Synch.	HW	Global Register Write Protection.
Class 2 — Instruction Errors				
1	IOPC	Synch.	HW	Illegal Opcode.
2	UOPC	Synch.	HW	Unimplemented Opcode.
3	OPD	Synch.	HW	Invalid Operand specification.
4	ALN	Synch.	HW	Data Address Alignment.
5	MEM	Synch.	HW	Invalid Local Memory Address.
Class 3 — Context Management				
1	FCD	Synch.	HW	Free Context List Depletion (FCX = LCX).
2	CDO	Synch.	HW	Call Depth Overflow.
3	CDU	Synch.	HW	Call Depth Underflow.
4	FCU	Synch.	HW	Free Context List Underflow (FCX = 0).
5	CSU	Synch.	HW	Call Stack Underflow (PCX = 0).
6	CTYP	Synch.	HW	Context Type (PCXI.UL wrong).
7	NEST	Synch.	HW	Nesting Error: RFE with non-zero call depth.
Class 4 — System Bus and Peripheral Errors				
1	PSE	Synch.	HW	Program Fetch Synchronous Error.
2	DSE	Synch.	HW	Data Access Synchronous Error.
3	DAE	Asynch.	HW	Data Access Asynchronous Error.
4	CAE	Asynch	HW	Coprocessor Trap Asynchronous Error.
5	PIE	Synch	HW	Program Memory Integrity Error.
6	DIE	Asynch	HW	Data Memory Integrity Error.
7	TAE	Asynch	HW	Temporal Asynchronous Error
Class 5— Assertion Traps				
1	OVF	Synch.	SW	Arithmetic Overflow.
2	SOVF	Synch.	SW	Sticky Arithmetic Overflow.
Class 6 — System Call[1]				
	SYS	Synch.	SW	System Call.
Class 7 — Non-Maskable Interrupt				
0	NMI	Asynch.	HW	Non-Maskable Interrupt.

Please refer to the TriCore™ TC1.6.2 core architecture manual and the AURIX™ TC3xx User's Manual for detailed information about each trap.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Trap types

- **Synchronous traps:**
  - Synchronous traps are associated with the execution or attempted execution of specific instructions or with attempts to access a virtual address that requires the intervention of the memory-management system.
  - The trap is triggered and serviced immediately.
- **Asynchronous traps:**
  - Since asynchronous traps are associated with hardware conditions, they are similar to interrupts.
  - They are routed via the trap vector.
  - Some asynchronous traps are triggered indirectly from instructions, that have been previously executed, but the direct association with the instructions causing the trap is lost.
- **Hardware traps:**
  - Hardware traps are generated in response to exception conditions detected by the hardware.
  - In most, but not all cases, the exception conditions are associated with the attempted execution of a particular instruction.
- **Software traps:**
  - Software traps are generated as an intentional result of executing a system call or an assertion instruction.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Trap handling

- When a trap occurs, a trap identifier is generated by hardware. The trap identifier has two components that can be used to determine more information about the trap and why it was caused (refer to slide Supported traps):
  - The Trap Class Number (TCN)
  - The Trap Identification Number (TIN)

- In most cases, the debugger will stop the code execution within one of the trap handlers (implemented in the iLLD header ***IfxCpu_Trap.c***)

- An instance of the structure ***IfxCpu_Trap*** is declared within each trap handler. When a trap occurs, the instance provides four information fields about the trap:
  - ***tCpu***: Which CPU caused the trap
  - ***tClass***: TCN, Class of the trap (refer to slide Supported traps)
  - ***tId***: TIN, Id of the trap (refer to slide Supported traps)
  - ***tAddr***: Return Address (RA) (refer to the next slide)

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Return Address

- The Return Address (RA) might help to locate the specific line of code which caused the trap.
- The return address, which is stored in the instance of the ***IfxCpu_Trap*** structure, is read from the return address register A[11].
- Depending on the **trap type**, the return address is different:
  - For most of the **synchronous** traps, the return address is the 32-bit Program Counter (PC) of the instruction that caused the trap. (The PC holds the address of the instruction which is currently running when the core is halted.)
  - On a **System Call (SYS)** trap, triggered by the SYSCALL instruction, the return address points to the instruction immediately following SYSCALL.
  - A **Free Context List Depletion (FCD)** trap is generated after a context save operation that causes the free context list becoming "almost empty".
    The responsible for the FCD trap can be a hardware interrupt or a trap handler. The operation responsible for the context save normally is completed before the FCD trap is executed. Because of this, the return address of the FCD trap is the first instruction of the trap/interrupt/called routine or the instruction following a Save Lower Context (SVLCX) or Begin Interrupt Service Routine (BISR) instruction.
  - For an **asynchronous** trap, the return address is the address of the instruction that would have been executed next, if the asynchronous trap had not been triggered.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Additional debug information

- The bit field ERROR_ADDRESS of the **Data Error Address Register (DEADD)** contains the trap address information for the data memory. The content of the DEADD register is valid if the **Data Synchronous Trap Register (DSTR)** or the **Data Asynchronous Trap Register (DATR)** register are non-zero (depending on the trap type). The bit fields in the DSTR and the DATR registers can provide additional information about the trap (refer to the TC3xx User's Manual).
  - These information are valid in case traps such as:
    - **Data Address Alignment (ALN)**
    - **Data Access Synchronous Error (DSE)**
    - **Data Access Asynchronous Error (DAE)**
    - **Invalid Local Memory Address (MEM)**
    - **Memory Protection Write (MPW)**
    - **Memory Protection Read (MPR)**
    - **Memory Protection Peripheral Access (MPP)**
    - **Memory Protection Null Address (MPN)**

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Additional debug information

- The **Program Memory Interface Synchronous Trap Register (PSTR)** contains synchronous trap information for the program memory system. The register is updated with trap information for **Program Fetch Synchronous Error traps (PSE)**.

- The **Program (or Data) Integrity Error Address Register (PIEAR / DIEAR)** and the **Program (or Data) Integrity Error Trap Register (PIETR / DIETR)** can be interrogated to determine the source of the **Program (or Data) Memory Integrity Error (PIE / DIE)** more precisely.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Trap provocation

- Three different combinations of trap types can be provoked in this example:
  - Synchronous Hardware trap
  - Asynchronous Hardware trap
  - Synchronous Software trap

- The trap provocation is implemented in the function ***run_trap_provocation()*** and can be started by setting one of the three ***g_provokeXYTrap*** **(X = Synchronous / Asynchronous; Y = Hardware / Software)** variables.

- The implemented code for the first two traps is based on the MTU_MBIST_1 and SMU_IR_Alarm_1 examples. For further information on the code, please refer to the specific tutorials.

- The third trap is provoked by using two instructions: __***mtcr()*** (Move To Core Register) and ***trapv*** (assembly code). For further information on these instructions, please refer to the TriCore™ TC1.6.2 core architecture manual - Instruction set manual.

Note: __***mtcr()*** is an intrinsic function of the Tasking compiler, which moves contents of a data register to the addressed Core Special Function Register (CSFR). __***mtcr()*** performs a Move to Core Register (***MTCR***) TriCore™ instruction and is followed by an ***ISYNC*** instruction.

- For a better understanding of the trap behavior, the required code instructions used to avoid the cause of each trap, are implemented and can be activated by setting the ***AVOID_PROVOCATION*** macro to true.

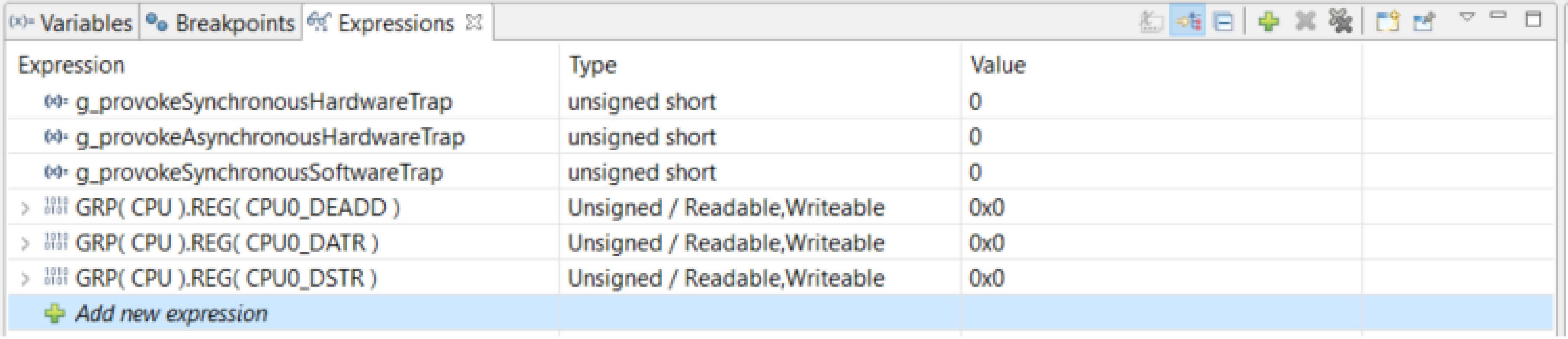
Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

After code compilation and flashing the device, perform the following steps:

- Add the three variables “***g_provokeSynchronousHardwareTrap***”, “***g_provokeAsynchronousHardwareTrap***” and “***g_provokeSynchronousSoftwareTrap***” in the Expressions window of the debugger.

- Add the three registers DEADD, DATR and DSTR in the Expressions window of the debugger.



Expression	Type	Value
g_provokeSynchronousHardwareTrap	unsigned short	0
g_provokeAsynchronousHardwareTrap	unsigned short	0
g_provokeSynchronousSoftwareTrap	unsigned short	0
GRP(CPU).REG(CPU0_DEADD)	Unsigned / Readable,Writeable	0x0
GRP(CPU).REG(CPU0_DATR)	Unsigned / Readable,Writeable	0x0
GRP(CPU).REG(CPU0_DSTR)	Unsigned / Readable,Writeable	0x0
Add new expression		

Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

1.1 Synchronous hardware trap

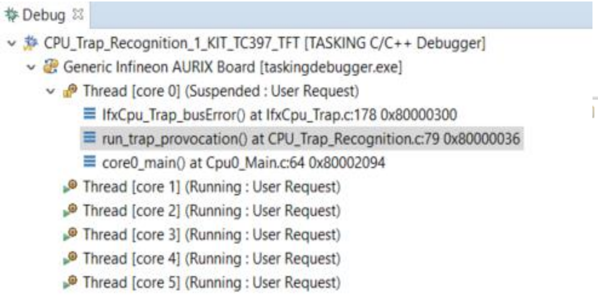
- Provoke the synchronous hardware trap by setting the value of "***g_provokeSynchronousHardwareTrap***" in the "Expressions" window to "1".
- Press the "Resume" button to start the program.
- Observe the following information:
  - The debugger stopped in the ***IfxCpu_Trap_busError()*** function (***IfxCpu_Trap.c***).
  - The "Variables" window of the debugger displays the "***trapWatch***" structure and the value of its parameters.
  - The trap is provoked by CPU0, it is a trap of class 4, the trap id is 2 and the Return Address (RA) is 0x80000036 (2147483702_{10}).
  - It is a Data Access Synchronous Error (Trap table, class 4 and tin 2).

Copyright © Infineon Technologies AG 2020. All rights reserved.

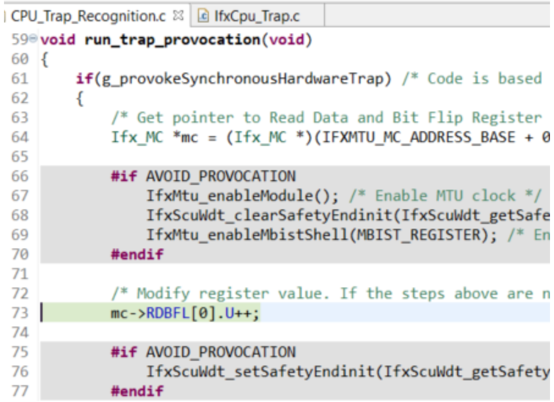
Run and Test

1.2 Synchronous hardware trap

- Observe the following information:
  - The call stack in the “Debug” window displays the function which was called before the trap occurred (in this case the function ***run_trap_provocation()***, the address displayed behind this function equals the Return Address (RA)).
  - By clicking on this function, the debugger jumps to the specific code line in the ***CPU_Trap_Recognition.c*** file and to the corresponding assembly line in the “Disassembly” window. The address of the assembly line equals the Return Address (RA).



```
Debug
CPU_Trap_Recognition_1_KIT_TC397_TFT [TASKING C/C++ Debugger]
  Generic Infineon AURIX Board [taskingdebugger.exe]
    Thread [core 0] (Suspended : User Request)
      IfxCpu_Trap_busError() at IfxCpu_Trap.c:178 0x80000300
      run_trap_provocation() at CPU_Trap_Recognition.c:79 0x80000036
      core0_main() at Cpu0_Main.c:64 0x80002094
    Thread [core 1] (Running : User Request)
    Thread [core 2] (Running : User Request)
    Thread [core 3] (Running : User Request)
    Thread [core 4] (Running : User Request)
    Thread [core 5] (Running : User Request)
```



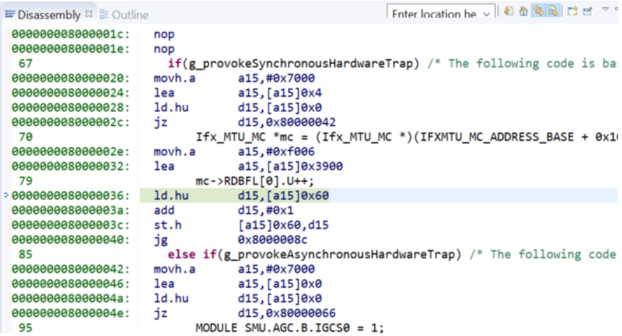
CPU_Trap_Recognition.c | IfxCpu_Trap.c

```
void run_trap_provocation(void)
{
    if(g_provokeSynchronousHardwareTrap) /* Code is based
    {
        /* Get pointer to Read Data and Bit Flip Register
        Ifx_MC *mc = (Ifx_MC *)(IFXMTU_MC_ADDRESS_BASE + 0

        #if AVOID_PROVOCATION
            IfxMtu_enableModule(); /* Enable MTU clock */
            IfxScuWdt_clearSafetyEndinit(IfxScuWdt_getSafe
            IfxMtu_enableMbistShell(MBIST_REGISTER); /* En
        #endif

        /* Modify register value. If the steps above are n
        mc->RDBFL[0].U++;

        #if AVOID_PROVOCATION
            IfxScuWdt_setSafetyEndinit(IfxScuWdt_getSafety
        #endif
```



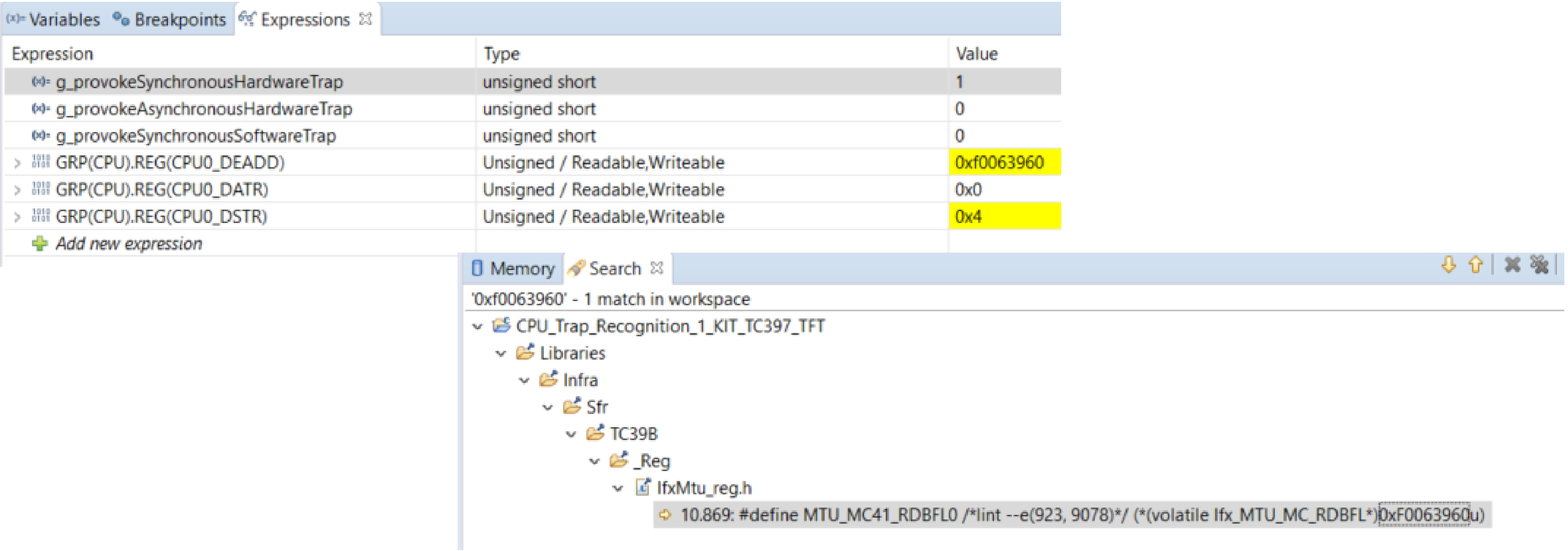
Disassembly | Outline

```
000000008000001c:  nop
000000008000001e:  nop
 67                  if(g_provokeSynchronousHardwareTrap) /* The following code is ba
0000000080000020:  movh.a      a15,#0x7000
0000000080000024:  lea         a15,[a15]0x4
0000000080000028:  ld.hu       d15,[a15]0x0
000000008000002c:  jz          d15,0x80000042
 70                      Ifx_MTU_MC *mc = (Ifx_MTU_MC *)(IFXMTU_MC_ADDRESS_BASE + 0x1
000000008000002e:  movh.a      a15,#0xf006
0000000080000032:  lea         a15,[a15]0x3900
 79                      mc->RDBFL[0].U++;
0000000080000036:  ld.hu       d15,[a15]0x60
000000008000003a:  add         d15,#0x1
000000008000003c:  st.h        [a15]0x60,d15
0000000080000040:  jg          0x8000008c
 85                  else if(g_provokeAsynchronousHardwareTrap) /* The following code
0000000080000042:  movh.a      a15,#0x7000
0000000080000046:  lea         a15,[a15]0x0
000000008000004a:  ld.hu       d15,[a15]0x0
000000008000004e:  jz          d15,0x80000066
 95                      MODULE_SMU.AGC.B.IGCS0 = 1;
```

Run and Test

1.3 Synchronous hardware trap

- Observe the following additional information:
  - The **LBE** bit field in the **DSTR** register is set (Load Bus Error - Data load from bus causing error, refer to AURIX™ TC3xx User’s Manual).
  - The **DEADD** register displays the address 0xf0063960, which is the address of the modified register which caused the trap.
  - By running a file search (Search -> File) for the address, the search finds the specific **RDBFL0** register which equals the modified **MBIST** DMA register.



Expression	Type	Value
g_provokeSynchronousHardwareTrap	unsigned short	1
g_provokeAsynchronousHardwareTrap	unsigned short	0
g_provokeSynchronousSoftwareTrap	unsigned short	0
GRP(CPU).REG(CPU0_DEADD)	Unsigned / Readable,Writeable	0xf0063960
GRP(CPU).REG(CPU0_DATR)	Unsigned / Readable,Writeable	0x0
GRP(CPU).REG(CPU0_DSTR)	Unsigned / Readable,Writeable	0x4

```
'0xf0063960' - 1 match in workspace
CPU_Trap_Recognition_1_KIT_TC397_TFT
  Libraries
    Infra
      Sfr
        TC39B
          _Reg
            IfxMtu_reg.h
              10.869: #define MTU_MC41_RDBFL0 /*lint --e(923, 9078)*/ (*(volatile Ifx_MTU_MC_RDBFL*)0xF0063960u)
```

Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

2.1 Asynchronous hardware trap

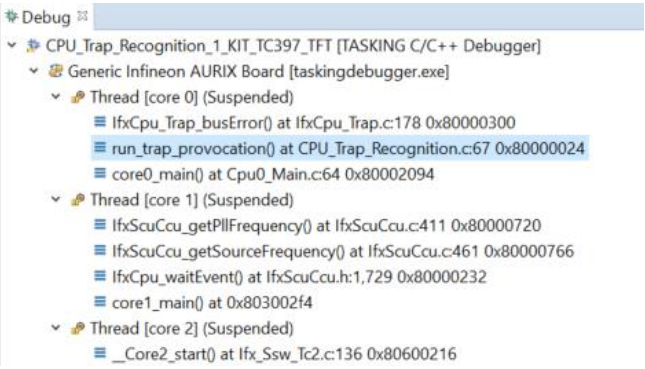
- Restart the program by pressing the “Restart” button in the debugger.
- Provoke the asynchronous hardware trap by setting the value of “***g_provokeAsynchronousHardwareTrap***” in the “Expressions” window to “1”.
- Press the “Resume” button to start the program.
- Observe the following information:
  - The debugger stopped in the ***IfxCpu_Trap_busError()*** function (***IfxCpu_Trap.c***).
  - The “Variables” window of the debugger displays the “***trapWatch***” structure and the values of its parameters.
  - The trap is provoked by CPU0, it is a trap of class 4, the trap id is 3 and the Return Address (RA) is 0x80000024 (2147483684_{10}).
  - It is a Data Access Asynchronous Error (Trap table, class 4 and tin 3).

Copyright © Infineon Technologies AG 2020. All rights reserved.

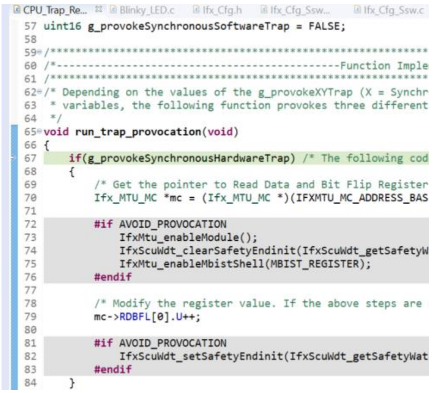
Run and Test

2.2 Aynchronous hardware trap

- Observe the following information:
  - The call stack in the “Debug” window displays the function which was called before the trap occurred (in this case the function ***run_trap_provocation()***, the address displayed behind this function equals the Return Address (RA)).
  - By clicking on this function, the debugger jumps to the specific code line in the ***CPU_Trap_Recognition.c*** file and to the corresponding assembly line in the “Disassembly” window. The address of the assembly line equals the return address.
  - Because it is an asynchronous trap, the specific code line is **not** pointing to the line which is causing the trap. It is the code line of the instruction that would have been executed next, if the asynchronous trap had not been triggered.
  - Since there is no other instruction within the function ***run_trap_provocation()***, it is impossible to find the line of code by using the Return Address (RA) in this example.



```
CPU_Trap_Recognition_1_KIT_TC397_TFT [TASKING C/C++ Debugger]
  Generic Infineon AURIX Board [taskingdebugger.exe]
    Thread [core 0] (Suspended)
      IfxCpu_Trap_busError() at IfxCpu_Trap.c:178 0x80000300
      run_trap_provocation() at CPU_Trap_Recognition.c:67 0x80000024
      core0_main() at Cpu0_Main.c:64 0x80002094
    Thread [core 1] (Suspended)
      IfxScuCcu_getPllFrequency() at IfxScuCcu.c:411 0x80000720
      IfxScuCcu_getSourceFrequency() at IfxScuCcu.c:461 0x80000766
      IfxCpu_waitEvent() at IfxScuCcu.h:1,729 0x80000232
      core1_main() at 0x803002f4
    Thread [core 2] (Suspended)
      _Core2_start() at Ifx_Ssw_Tc2.c:136 0x80600216
```



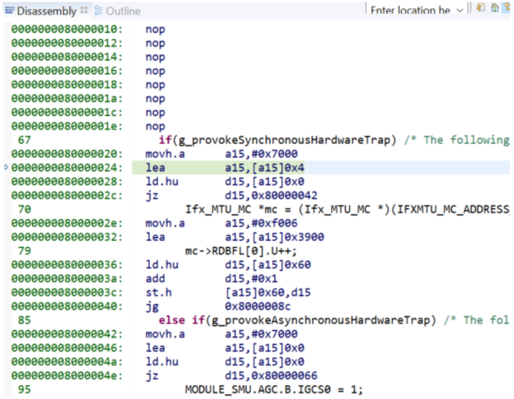
```c
uint16 g_provokeSynchronousSoftwareTrap = FALSE;

/*****************************************************************
/*---------------------------------------------Function Imple
/*****************************************************************
/* Depending on the values of the g_provokeXYTrap (X = Synchr
 * variables, the following function provokes three different
 */
void run_trap_provocation(void)
{
    if(g_provokeSynchronousHardwareTrap) /* The following cod
    {
        /* Get the pointer to Read Data and Bit Flip Register
        Ifx_MTU_MC *mc = (Ifx_MTU_MC *)(IFXMTU_MC_ADDRESS_BAS

        #if AVOID_PROVOCATION
            IfxMtu_enableModule();
            IfxScuWdt_clearSafetyEndinit(IfxScuWdt_getSafetyW
            IfxMtu_enableMbistShell(MBIST_REGISTER);
        #endif

        /* Modify the register value. If the above steps are
        mc->RDBFL[0].U++;

        #if AVOID_PROVOCATION
            IfxScuWdt_setSafetyEndinit(IfxScuWdt_getSafetyWat
        #endif
    }
```



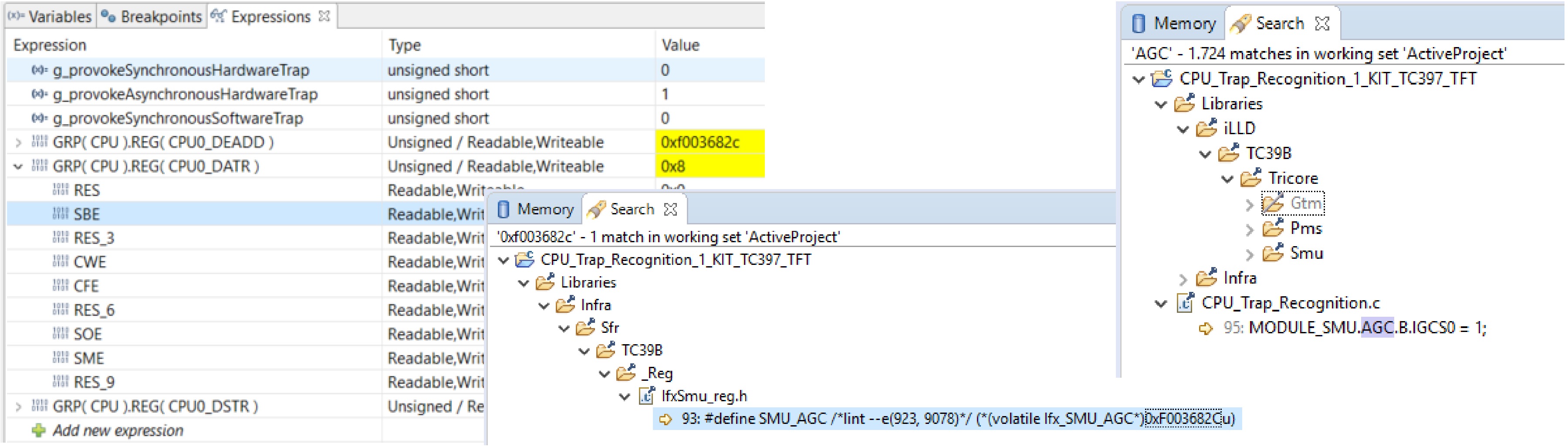
```
0000000080000010:   nop
0000000080000012:   nop
0000000080000014:   nop
0000000080000016:   nop
0000000080000018:   nop
000000008000001a:   nop
000000008000001c:   nop
000000008000001e:   nop
 67                   if(g_provokeSynchronousHardwareTrap) /* The following
0000000080000020:   movh.a       a15,#0x7000
0000000080000024:   lea          a15,[a15]0x4
0000000080000028:   ld.hu        d15,[a15]0x0
000000008000002c:   jz           d15,0x80000042
 70                       Ifx_MTU_MC *mc = (Ifx_MTU_MC *)(IFXMTU_MC_ADDRESS
000000008000002e:   movh.a       a15,#0xf006
0000000080000032:   lea          a15,[a15]0x3900
 79                       mc->RDBFL[0].U++;
0000000080000036:   ld.hu        d15,[a15]0x60
000000008000003a:   add          d15,#0x1
000000008000003c:   st.h         [a15]0x60,d15
0000000080000040:   jg           0x8000008c
 85                   else if(g_provokeAsynchronousHardwareTrap) /* The fol
0000000080000042:   movh.a       a15,#0x7000
0000000080000046:   lea          a15,[a15]0x0
000000008000004a:   ld.hu        d15,[a15]0x0
000000008000004e:   jz           d15,0x80000066
 95                       MODULE_SMU.AGC.B.IGCS0 = 1;
```

Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

2.3 Asynchronous hardware trap

- Due to the fact that the Return Address (RA) cannot be used, the following information might help to locate the cause of the trap:
  - The **SBE** bit field in the **DATR** register is set (Store Bus Error - Data store to bus causing error, refer to AURIX™ TC3xx User’s Manual).
  - The **DEADD** register displays the address 0xf003682c, which is the address of the modified register that caused the trap.
  - By running a file search (Search -> File) for the address, the search finds the specific **SMU_AGC** register which equals the modified register. The name of the modified register helps to find the code line which is causing the trap (By using another search for “AGC”).



Expression	Type	Value
g_provokeSynchronousHardwareTrap	unsigned short	0
g_provokeAsynchronousHardwareTrap	unsigned short	1
g_provokeSynchronousSoftwareTrap	unsigned short	0
GRP(CPU).REG(CPU0_DEADD)	Unsigned / Readable,Writeable	0xf003682c
GRP(CPU).REG(CPU0_DATR)	Unsigned / Readable,Writeable	0x8
RES	Readable,Writ	0x0
SBE	Readable,Writ	
RES_3	Readable,Writ	
CWE	Readable,Writ	
CFE	Readable,Writ	
RES_6	Readable,Writ	
SOE	Readable,Writ	
SME	Readable,Writ	
RES_9	Readable,Writ	
GRP(CPU).REG(CPU0_DSTR)	Unsigned / Re	
Add new expression		

```
'0xf003682c' - 1 match in working set 'ActiveProject'
CPU_Trap_Recognition_1_KIT_TC397_TFT
  Libraries
    Infra
      Sfr
        TC39B
          _Reg
            IfxSmu_reg.h
              93: #define SMU_AGC /*lint --e(923, 9078)*/ (*(volatile Ifx_SMU_AGC*)0xF003682Cu)
```

```
'AGC' - 1.724 matches in working set 'ActiveProject'
CPU_Trap_Recognition_1_KIT_TC397_TFT
  Libraries
    iLLD
      TC39B
        Tricore
          Gtm
          Pms
          Smu
    Infra
  CPU_Trap_Recognition.c
    95: MODULE_SMU.AGC.B.IGCS0 = 1;
```

Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

3.1 Synchronous software trap

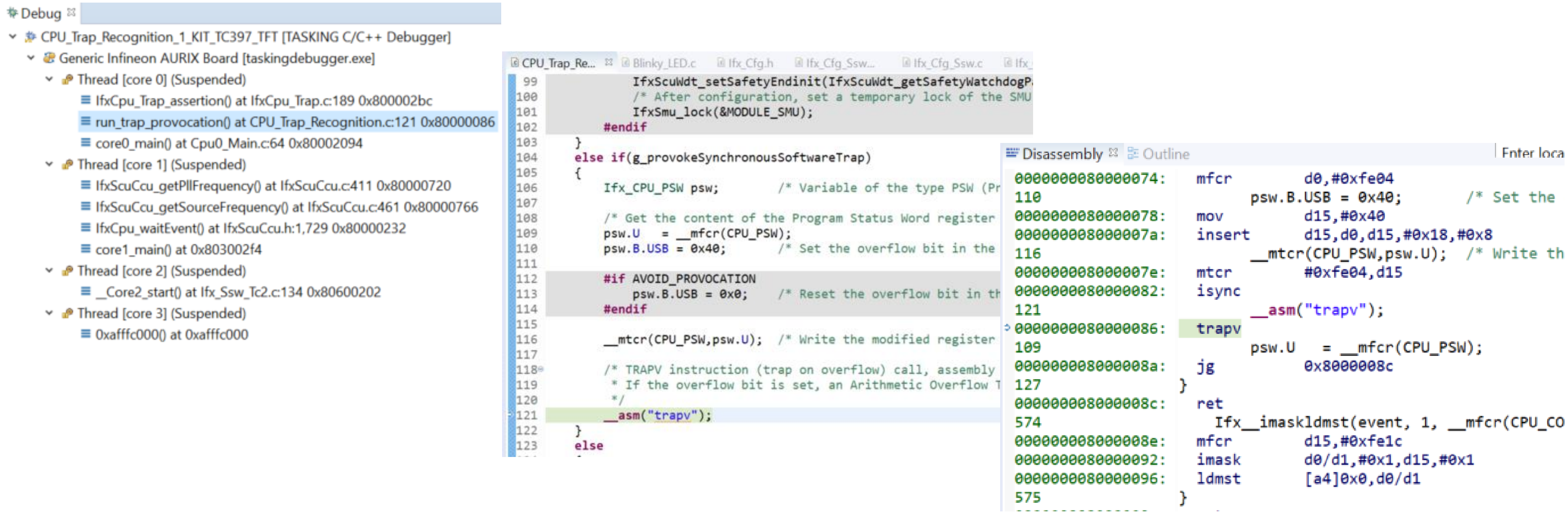
- Restart the program by pressing the “Restart” button in the debugger.
- Provoke the synchronous software trap by setting the value of “***g_provokeSynchronousSoftwareTrap***” in the “Expressions” window to “1”.
- Press the “Resume” button to start the program.
- Observe the following information:
  - The debugger stopped in the ***IfxCpu_Trap_assertion()*** function (***IfxCpu_Trap.c***).
  - The “Variables” window of the debugger displays the “***trapWatch***” structure and the value of its parameters.
  - The trap is provoked by CPU0, it is a trap of class 5, the trap id is 1 and the Return Address (RA) is 0x80000086 (2147483782_{10}).
  - It is an Arithmetic Overflow Error (Trap table, class 5 and tin 1).

Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

3.2 Synchronous software trap

- Observe the following information:
  - The call stack in the “Debug” window displays the function which was called before the trap occurred (in this case the function ***run_trap_provocation()***, the address displayed behind this function equals the Return Address (RA)).
  - By clicking on this function, the debugger jumps to the specific code line in the ***CPU_Trap_Recognition.c*** file and to the corresponding assembly line in the “Disassembly” window. The address of the assembly line equals the Return Address (RA).



```
CPU_Trap_Recognition_1_KIT_TC397_TFT [TASKING C/C++ Debugger]
  Generic Infineon AURIX Board [taskingdebugger.exe]
    Thread [core 0] (Suspended)
      IfxCpu_Trap_assertion() at IfxCpu_Trap.c:189 0x800002bc
      run_trap_provocation() at CPU_Trap_Recognition.c:121 0x80000086
      core0_main() at Cpu0_Main.c:64 0x80002094
    Thread [core 1] (Suspended)
      IfxScuCcu_getPllFrequency() at IfxScuCcu.c:411 0x80000720
      IfxScuCcu_getSourceFrequency() at IfxScuCcu.c:461 0x80000766
      IfxCpu_waitEvent() at IfxScuCcu.h:1,729 0x80000232
      core1_main() at 0x803002f4
    Thread [core 2] (Suspended)
      _Core2_start() at Ifx_Ssw_Tc2.c:134 0x80600202
    Thread [core 3] (Suspended)
      0xafffc000() at 0xafffc000
```

```
            IfxScuWdt_setSafetyEndinit(IfxScuWdt_getSafetyWatchdogP
            /* After configuration, set a temporary lock of the SMU
            IfxSmu_lock(&MODULE_SMU);
        #endif
    }
    else if(g_provokeSynchronousSoftwareTrap)
    {
        Ifx_CPU_PSW psw;        /* Variable of the type PSW (Pr

        /* Get the content of the Program Status Word register
        psw.U   = __mfcr(CPU_PSW);
        psw.B.USB = 0x40;       /* Set the overflow bit in the

        #if AVOID_PROVOCATION
            psw.B.USB = 0x0;    /* Reset the overflow bit in th
        #endif

        __mtcr(CPU_PSW,psw.U);  /* Write the modified register

        /* TRAPV instruction (trap on overflow) call, assembly
         * If the overflow bit is set, an Arithmetic Overflow T
         */
        __asm("trapv");
    }
    else
```

```
0000000080000074:   mfcr        d0,#0xfe04
110                      psw.B.USB = 0x40;       /* Set the
0000000080000078:   mov         d15,#0x40
000000008000007a:   insert      d15,d0,d15,#0x18,#0x8
116                      __mtcr(CPU_PSW,psw.U);  /* Write th
000000008000007e:   mtcr        #0xfe04,d15
0000000080000082:   isync
121                      __asm("trapv");
0000000080000086:   trapv
109                      psw.U   = __mfcr(CPU_PSW);
000000008000008a:   jg          0x8000008c
127                 }
000000008000008c:   ret
574                   Ifx__imaskldmst(event, 1, __mfcr(CPU_CO
000000008000008e:   mfcr        d15,#0xfe1c
0000000080000092:   imask       d0/d1,#0x1,d15,#0x1
0000000080000096:   ldmst       [a4]0x0,d0/d1
575                 }
```

Copyright © Infineon Technologies AG 2020. All rights reserved.

References

- AURIX™ Development Studio is available online:
- https://www.infineon.com/aurixdevelopmentstudio
- Use the „*Import...*“ function to get access to more code examples.

- More code examples can be found on the GIT repository:
- https://github.com/Infineon/AURIX_code_examples

- For additional trainings, visit our webpage:
- https://www.infineon.com/aurix-expert-training

- For questions and support, use the AURIX™ Forum:
- https://www.infineonforums.com/forums/13-Aurix-Forum

Copyright © Infineon Technologies AG 2020. All rights reserved.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2020-12
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2020 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
CPU_Trap_Recognition_1 _KIT_TC397_TFT

IMPORTANT NOTICE
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the thereof can reasonably be expected to result in personal injury.