Errata Sheet

Rel. 1.2, 25.01.2011

Device	**TC1767ED**
Marking/Step	**EES-AD, ES-AD**
Package	**PG-LQFP-176-4**

01960AERRA

This Errata Sheet describes the deviations from the current user documentation.

Table 1 Current Documentation

TC1767/97ED Target Specification	V1.5	December 2008
TC1767 User's Manual	V1.1	May 2009
TC1767 Data Sheet	V1.3	September 2009
TriCore 1 Architecture	V1.3.8	January 2008

Make sure you always use the corresponding documentation for this device (User's Manual, Data Sheet, Documentation Addendum (if applicable), TriCore Architecture Manual, Errata Sheet) available in category 'Documents' at **www.infineon.com/TC1767**.

Each erratum identifier follows the pattern **Module_Arch.TypeNumber**:

- **Module**: subsystem, peripheral, or function affected by the erratum
- **Arch**: microcontroller architecture where the erratum was firstly detected
  - **AI**: Architecture Independent
  - **CIC**: Companion ICs
  - **TC**: TriCore
  - **X**: XC166 / XE166 / XC2000 Family
  - **XC8**: XC800 Family
  - **[none]**: C166 Family
- **Type**: category of deviation

  - **[none]**: Functional Deviation
  - **P**: Parametric Deviation
  - **H**: Application Hint
  - **D**: Documentation Update
- **Number**: ascending sequential number within the three previous fields. As this sequence is used over several derivatives, including already solved deviations, gaps inside this enumeration can occur.

Note: Devices marked with EES or ES are engineering samples which may not be completely tested in all functional and electrical characteristics, therefore they should be used for evaluation only.

Note: This device is equipped with a TriCore "TC1.3.1" Core. Some of the errata have workarounds which are possibly supported by the tool vendors. Some corresponding compiler switches need possibly to be set. Please see the respective documentation of your compiler.
For effects of issues related to the on-chip debug system, see also the documentation of the debug tool vendor.

The specific test conditions for EES and ES are documented in a separate Status Sheet.

1 History List / Change Summary

Table 2 History List

Version	Date	Remark
1.0	04.07.2008	
1.1	15.03.2010	- Updated Documentation Reference: TC1767ED Target Specification v1.5 2008-12 TC1767 User's Manual V1.1 2009-15 TC1767 Data Sheet V1.3 2009-09 - Removed BROM_TC.H001 (Frequency Ratio fSYS = fOSC/2 for Bootstrap Loaders), see p.7-3 in TC1767 User's Manual V1.1. - Removed FLASH_TC.036 (DFLASH Margin Control Register MARD) not included, updated description see p.5-64 in TC1767 User's Manual V1.1.
1.2	25.01.2011	

Note: Changes to the previous errata sheet version are particularly marked in column "Change" in the following tables.

Table 3 Errata fixed in this step

Errata	Short Description	Chg
BROM_TC.004	No Watchdog Timer reset when in error state during bootcode	Fixed

Table 4 Functional Deviations

Functional Deviation	Short Description	Chg	Pg
BCU_TC.006	Polarity of Bit SVM in Register ECON		11
BROM_TC.005	Power-on reset (PORST) while no external clock is available	New	11
CPU_TC.105	User / Supervisor mode not staged correctly for Store Instructions		11
CPU_TC.106	Incorrect PSW update for certain IP instructions dual-issued with MTCR PSW		12
CPU_TC.107	SYSCON.FCDSF may not be set after FCD Trap		13
CPU_TC.108	Incorrect Data Size for Circular Addressing mode instructions with wrap-around		14
CPU_TC.109	Circular Addressing Load can overtake conflicting Store in Store Buffer		17
CPU_TC.110	Register Banks may be out of sync after FCU Trap		20
CPU_TC.111	Imprecise Return Address for FCU Trap		22
CPU_TC.113	Interrupt may be taken during Trap entry sequence		23
CPU_TC.114	CAE Trap may be generated by UPDFL instruction		26
CPU_TC.115	Interrupt may be taken on exit from Halt mode with Interrupts disabled		27
CPU_TC.117	Cached Store Data Lost on Data Cache Invalidate via Overlay	New	29
DMA_TC.013	DMA-LMB-Master Access to Reserved Address Location		31
DMI_TC.014	Problems with Parity Handling in TriCore Data Memories		32

Table 4 Functional Deviations (cont'd)

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
OCDS_TC.020	ICTTA not used by Triggered Transfer to External Address		49
OCDS_TC.021	TriCore breaks on de-assertion instead of assertion of break bus		49
OCDS_TC.024	Loss of Connection in DAP three-pin Mode		50
OCDS_TC.025	PC corruption when entering Halt mode after a MTCR to DBGSR		51
OCDS_TC.026	PSW.PRS updated too late after a RFM instruction.		52
OCDS_TC.027	BAM breakpoints with associated halt action can potentially corrupt the PC.		53
OCDS_TC.028	Accesses to CSFR and GPR registers of running program can corrupt loop exits.		54
PCP_TC.023	JUMP sometimes takes an extra cycle		55
PCP_TC.027	Longer delay when clearing R7.IEN before atomic PRAM instructions		56
PCP_TC.032	Incorrect PCP behaviour following FPI timeouts (as a slave)		56
PCP_TC.034	Usage of R7 requires delays between operations		56
PCP_TC.035	Atomic PRAM operation right after COPY/BCOPY		57
PCP_TC.036	Unexpected behaviour after failed posted FPI write		57
PCP_TC.038	PCP atomic PRAM operations may operate incorrectly		58
PCP_TC.039	PCP posted error interrupt to CPU may be lost when the queue is full in 2:1 mode		60

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
RESET_TC.001	SCU_RSTSTAT.PORST not set by a combined Debug / System / Application Reset		60
SCU_TC.016	Reset Value of Registers ESRCFG0/1		61
SSC_AI.022	Phase error detection switched off too early at the end of a transmission		61
SSC_AI.023	Clock phase control causes failing data transmission in slave mode		62
SSC_AI.024	SLSO output gets stuck if a reconfig from slave to master mode happens		62
SSC_AI.025	First shift clock period will be one PLL clock too short because not syncronized to baudrate		63
SSC_AI.026	Master with highest baud rate set generates erroneous phase error		63

Table 5 Deviations from Electrical- and Timing Specification

AC/DC/ADC Deviation	Short Description	Chg	Pg
DTS_TC.P001	Test Conditions for Sensor Accuracy T_{TSA}		65
ESD_TC.P002	ESD violation		65
FADC_TC.P003	Incorrect test condition specified in datasheet for FADC parameter "Input leakage current at V_{FAGND}".		65
MSC_TC.P001	Incorrect V_{OS} limits for LVDS pads specified in Data Sheet	New	66

Table 5 Deviations from Electrical- and Timing Specification (cont'd)

AC/DC/ADC Deviation	Short Description	Chg	Pg
PLL_TC.P005	PLL Parameters for f_{VCO} > 780 MHz		66

Table 6 Application Hints

Hint	Short Description	Chg	Pg
ADC_AI.H002	Minimizing Power Consumption of an ADC Module		67
CPU_TC.H004	PCXI Handling Differences in TriCore1.3.1		67
CPU_TC.H005	Wake-up from Idle/Sleep Mode	New	69
FIRM_TC.H000	Reading the Flash Microcode Version		70
FPI_TC.H001	FPI bus may be monopolized despite starvation protection		70
HYS_TC.H001	Effective Hysteresis in Application Environment		71
MCDS_TC.H001	TriCore trigger sources core_cu* and core_clr* toggle with some taken branches		71
MCDS_TC.H002	Missing EOT message at end of trace buffer		72
MCDS_TC.H003	Trace buffer content after abort		73
MCDS_TC.H004	Decoding the data size of DTU messages		73
MSC_TC.H007	Start Condition for Upstream Channel		75
MSC_TC.H008	The LVDS pads require a settling time when coming up from pad power-down state.		75
MultiCAN_AI.H005	TxD Pulse upon short disable request		76

Table 6 Application Hints (cont'd)

Hint	Short Description	Chg	Pg
MultiCAN_AI.H006	Time stamp influenced by resynchronization		76
MultiCAN_TC.H002	Double Synchronization of receive input		77
MultiCAN_TC.H003	Message may be discarded before transmission in STT mode		77
MultiCAN_TC.H004	Double remote request		77
OCDS_TC.H001	IOADDR may increment after aborted IO_READ_BLOCK		78
OCDS_TC.H002	Setting IOSR.CRSYNC during Application Reset		79
OCDS_TC.H003	Application Reset during host communication		79
OCDS_TC.H004	Device Identification by Application Software		80
PCP_TC.H004	Invalid parity error generated by FPI write to PRAM		81
PCP_TC.H005	Unexpected parity errors when address 0 of CMEM is faulty		81
PCP_TC.H006	BCOPY address alignment error may affect next channel FPI operation		81
PCP_TC.H007	Do not use priority 0 to post interrupt to CPU		82
PORTS_TC.H004	Using LVDS Ports in CMOS Mode		82
PORTS_TC.H005	Pad Input Registers do not capture Boundary-Scan data when BSD-mode signal is set to high		82
PWR_TC.H005	Current Peak on V_{DDP} during Power-up		83
SSC_AI.H001	Transmit Buffer Update in Slave Mode after Transmission		83
SSC_AI.H002	Transmit Buffer Update in Master Mode during Trailing or Inactive Delay Phase		84

Table 6 Application Hints (cont'd)

Hint	Short Description	Chg	Pg
SSC_AI.H003	Transmit Buffer Update in Slave Mode during Transmission		84
SSC_TC.H003	Handling of Flag STAT.BSY in Master Mode		85

2 Functional Deviations

BCU_TC.006 Polarity of Bit SVM in Register ECON

The polarity of bit SVM (State of FPI Bus Supervisor Mode Signal) in the SBCU Error Control Capture register SBCU_ECON is inverted compared to its description in the User's Manual.

Actually, it is implemented as follows:

- SVM = 0_B: Transfer was initiated in user modes
- SVM = 1_B: Transfer was initiated in supervisor mode

BROM_TC.005 Power-on reset (PORST) while no external clock is available

In case no stable clock is present at the oscillator input pin (XTAL1) after PORST, the device will wait indefinitely, i.e. it hangs and is not able to execute application code or enter one of the bootstrap loader modes.

Workaround

Proper device start-up after PORST is only possible if a stable clock signal from an external crystal, ceramic resonator or an external clock source is to available at the XTAL1 pin of the device

CPU_TC.105 User / Supervisor mode not staged correctly for Store Instructions

Bus transactions initiated by TriCore load or store instructions have a number of associated attributes such as address, data size etc. derived from the load or store instruction itself. In addition, bus transactions also have an IO privilege level status flag (User/Supervisor mode) derived from the PSW.IO bit field. Unlike attributes derived from the instruction, the User/Supervisor mode status of TriCore initiated bus transactions is not staged correctly in the TriCore pipeline and is derived directly from the PSW.IO bit field.

This issue can only cause a problem in certain circumstances, specifically when a store transaction is outstanding (e.g. held in the CPU store buffer) and the PSW is modified to switch from Supervisor to User-0 or User-1 mode. In this case, the outstanding store transaction, executed in Supervisor mode, may be transferred to the bus in User mode (the bus systems do not discriminate between User-0 and User-1 modes). Due to the blocking nature of load transactions and the fact that User mode code cannot modify the PSW, neither of these other situations can cause a problem.

Example

```
...
st.w [aX], dX   ; Store to Supervisor mode protected SFR
mtcr #PSW, dY   ; Modify PSW.IO to switch to User mode
...
```

Workaround

Any MTCR instruction targeting the PSW, which may change the PSW.IO bit field, must be preceded by a DSYNC instruction, unless it can be guaranteed that no store transaction is outstanding.

```
...
st.w [aX], dX   ; Store to Supervisor mode protected SFR
dsync
mtcr #PSW, dY   ; Modify PSW.IO to switch to User mode
...
```

CPU_TC.106 Incorrect PSW update for certain IP instructions dual-issued with MTCR PSW

In certain situations where an Integer Pipeline (IP) instruction which updates the PSW user status bits (e.g. PSW.V - Overflow) is followed immediately by an MTCR instruction targetting the PSW, with the instructions being dual-issued, the update priority is incorrect. In this case, the PSW user status bits are updated with the value from the IP instruction rather than the later MTCR instruction. This situation only occurs in 2 cases:

- MUL/MADD/MSUB instruction followed by MTCR `PSW`
- RSTV instruction followed by MTCR `PSW`

Example

```
...
rstv
mtcr #PSW, dY   ; Modify PSW
...
```

Workaround

Insert one NOP instruction between the MUL/MADD/MSUB/RSTV instruction and the MTCR instruction updating the `PSW`.

```
...
rstv
nop
mtcr #PSW, dY   ; Modify PSW
...
```

CPU_TC.107 SYSCON.FCDSF may not be set after FCD Trap

Under certain conditions the `SYSCON.FCDSF` flag may not be set after an FCD trap is entered. This situation may occur when the CSA (Context Save Area) list is located in cacheable memory, or, dependent upon the state of the upper context shadow registers, when the CSA list is located in LDRAM.

The `SYSCON.FCDSF` flag may be used by other trap handlers, typically those for asynchronous traps, to determine if an FCD trap handler was in progress when the another trap was taken.

Workaround

In the case where the CSA list is statically located in memory, asynchronous trap handlers may detect that an FCD trap was in progress by comparing the current values of `FCX` and `LCX`, thus achieving similar functionality to the `SYSCON.FCDSF` flag.

In the case where the CSA list is dynamically managed, no reliable workaround is possible.

CPU_TC.108 Incorrect Data Size for Circular Addressing mode instructions with wrap-around

In certain situations where a Load or Store instruction using circular addressing mode encounters the circular buffer wrap-around condition, the first access to the circular buffer may be performed using an incorrect data size, causing too many or too few data bytes to be transferred. The circular buffer wrap-around condition occurs when a load or store instruction using circular addressing mode addresses a data item which spans the boundary of a circular buffer, such that part of the data item is located at the top of the buffer, with the remainder at the base. The problem may occur in one of two cases:

Case 1

Where a **store** instruction using circular addressing mode encounters the circular buffer wrap-around condition, and is preceded in the LS pipeline by a multi-access load instruction, the first access of the store instruction using circular addressing mode may incorrectly use the transfer data size from the second part of the multi-access load instruction. A multi-access load instruction occurs in one of the following circumstances:

- Unaligned access to LDRAM or cacheable address which spans a 128-bit boundary.
- Unaligned access to a non-cacheable, non-LDRAM address.
- Circular addressing mode access which encounters the circular buffer wrap-around condition.

Since half-word store instructions must be half-word aligned, and st.a instructions must be word aligned, they cannot trigger the circular buffer wrap-around condition. As such, this case only affects the following instructions using circular addressing mode: st.w, st.d, st.da.

Example

```
...
LDA   a8,   0xD000000E ; Address of un-aligned load
LDA   a12,  0xD0000820 ; Circular Buffer Base
LDA   a13,  0x00180014 ; Circular Buffer Limit and Index
...
ld.w d6,  [a8]         ; Un-aligned load, split 16+16
add  d4,  d3,  d2      ; Optional IP instruction
st.d [a12/a13+c], d0/d1 ; Circular Buffer wrap, 32+32
...
```

In this example, the word load from address 0xD000000E is split into 2 half-word accesses, since it spans a 128-bit boundary in LDRAM. The double-word store encounters the circular buffer wrap condition and should be split into 2 word accesses, to the top and bottom of the circular buffer. However, due to the bug, the first access takes the transfer data size from the second part of the un-aligned load and only 16-bits of data are written. Note that the presence of an optional IP instruction between the load and store transactions does not prevent the problem, since the load and store transactions are back-to-back in the LS pipeline.

Case 2

Case 2 is similar to case 1, and occurs where a **load** instruction using circular addressing mode encounters the circular buffer wrap-around condition, and is preceded in the LS pipeline by a multi-access load instruction. However, for case 2 to be a problem it is necessary that the first access of the load instruction encountering the circular buffer wrap-around condition (the access to the top of the circular buffer) also encounters a conflict condition with the contents of the CPU store buffer. Again, in this case the first access of the load instruction using circular addressing mode may incorrectly use the transfer data size from the second part of the multi-access load instruction. Since half-word load instructions must be half-word aligned, and ld.a instructions must be word aligned, they cannot trigger the circular buffer wrap-around condition. As such, this case only affects the following instructions using circular addressing mode: ld.w, ld.d, ld.da.

Note: In the current TriCore1 CPU implementation, load accesses are initiated from the DEC pipeline stage whilst store accesses are initiated from the following EXE pipeline stage. To avoid memory port contention problems when a load follows a store instruction, the CPU contains a single store buffer. In the case where a store instruction (in EXE) is immediately followed by a load instruction (in DEC), the store is directed to the CPU store buffer and the load operation overtakes the store. The store is then committed to memory from the store buffer on the next store instruction or non-memory access cycle. The store buffer is only used for store accesses to 'local' memories - LDRAM or DCache. Store instructions to bus-based memories are always executed immediately (in-order). A store buffer conflict is detected when a load instruction is encountered which targets an address for which at least part of the requested data is currently held in the CPU store buffer. In this store buffer conflict scenario, the load instruction is cancelled, the store committed to memory from the store buffer and then the load re-started. In systems with an enabled MMU and where either the store buffer or load instruction targets an address undergoing PTE-based translation, the conflict detection is just performed on address bits (9:0), since higher order bits may be modified by translation and a conflict cannot be ruled out. In other systems (no MMU, MMU disabled), conflict detection is performed on the complete address.

Example

```
...
LDA  a8,  0xD000000E ; Address of un-aligned load
LDA  a12, 0xD0000820 ; Circular Buffer Base
LDA  a13, 0x00180014 ; Circular Buffer Limit and Index
...
st.h [a12]0x14, d7   ; Store causing conflict
ld.w d6, [a8]        ; Un-aligned load, split 16+16
add  d4, d3, d2      ; Optional IP instruction
ld.d [a12/a13+c], d0/d1 ; Circular Buffer wrap, 32+32
                        ; conflict with st.h
...
```

In this example, the half-word store is to address 0xD0000834 and is immediately followed by a load instruction, so is directed to the store buffer. The word load from address 0xD000000E is split into 2 half-word accesses, since it spans a 128-bit boundary in LDRAM. The double-word load encounters the circular buffer wrap condition and should be split into 2 word accesses, to the top and bottom of the circular buffer. In addition, the first circular buffer access conflicts with the store to address 0xD0000834. Due to the bug, after the store buffer is flushed, the first access takes the transfer data size from the second part of the un-aligned load and only 16-bits of data are read. Note that the presence of an optional IP instruction between the two load transactions does not prevent the problem, since the load transactions are back-to-back in the LS pipeline.

Workaround

Where it cannot be guaranteed that a word or double-word load or store instruction using circular addressing mode will not encounter one of the corner cases detailed above which may lead to incorrect behaviour, one NOP instruction should be inserted prior to the load or store instruction using circular addressing mode.

```
...
LDA   a8,   0xD000000E ; Address of un-aligned load
LDA   a12,  0xD0000820 ; Circular Buffer Base
LDA   a13,  0x00180014 ; Circular Buffer Limit and Index
...
ld.w  d6,  [a8]        ; Un-aligned load, split 16+16
add   d4, d3, d2       ; Optional IP instruction
nop                    ; Bug workaround
st.d [a12/a13+c], d0/d1 ; Circular Buffer wrap, 32+32
...
```

CPU_TC.109 Circular Addressing Load can overtake conflicting Store in Store Buffer

In a specific set of circumstances, a load instruction using circular addressing mode may overtake a conflicting store held in the TriCore1 CPU store buffer. The problem occurs in the following situation:

- The CPU store buffer contains a **byte** store instruction, st.b, targeting the base address + 0x1 of a circular buffer.
- A **word** load instruction, ld.w, is executed using circular addressing mode, targetting the same circular buffer as the buffered byte store.
- This word load is only half-word aligned and encounters the circular buffer wrap-around condition such that the second, wrapped, access of the load instruction to the bottom of the circular buffer targets the same address as the byte store held in the store buffer.

Additionally, one of the following further conditions must also be present for the problem to occur:

- The circular buffer base address for the word load is double-word but not quad-word (128-bit) aligned - i.e. the base address has bits (3:0) = 0x8 with the conflicting byte store having address bits (3:0) = 0x9, OR,
- The circular buffer base address for the word load is quad-word (128-bit) aligned and the circular buffer size is an odd number of words - i.e. the base address has bits (3:0) = 0x0 with the conflicting byte store having address bits (3:0) = 0x1.

In these very specific circumstances the conflict between the load instruction and store buffer contents is not detected and the load instruction overtakes the store, returning the data value prior to the store operation.

Note: In the current TriCore1 CPU implementation, load accesses are initiated from the DEC pipeline stage whilst store accesses are initiated from the following EXE pipeline stage. To avoid memory port contention problems when a load follows a store instruction, the CPU contains a single store buffer. In the case where a store instruction (in EXE) is immediately followed by a load instruction (in DEC), the store is directed to the CPU store buffer and the load operation overtakes the store. The store is then committed to memory from the store buffer on the next store instruction or non-memory access cycle. The store buffer is only used for store accesses to 'local' memories - LDRAM or DCache. Store instructions to bus-based memories are always executed immediately (in-order). A store buffer conflict is detected when a load instruction is encountered which targets an address for which at least part of the requested data is currently held in the CPU store buffer. In this store buffer conflict scenario, the load instruction is cancelled, the store committed to memory from the store *buffer and then the load re-started. In systems with an enabled MMU and where either the store buffer or load instruction targets an address undergoing PTE-based translation, the conflict detection is just performed on address bits (9:0), since higher order bits may be modified by translation and a conflict cannot be ruled out. In other systems (no MMU, MMU disabled), conflict detection is performed on the complete address.*

Example - Case 1

```
...
LDA  a12, 0xD0001008 ; Circular Buffer Base
LDA  a13, 0x00180016 ; Circular Buffer Limit and Index
...
st.b [a12]0x1, d2    ; Store to byte offset 0x9
ld.w d6, [a12/a13+c] ; Circular Buffer wrap, 16+16
...
```

In this example the circular buffer base address is double-word but not quad-word aligned. The byte store to address 0xD0001009 is immediately followed by a load operation and is placed in the CPU store buffer. The word load instruction encounters the circular buffer wrap condition and is split into 2 half-word accesses, to the top (0xD0001016) and bottom (0xD0001008) of the circular buffer. The first load access completes correctly, but, due to the bug, the second access overtakes the store operation and returns the previous half-word from 0xD0001008.

Example - Case 2

```
...
LDA  a12, 0xD0001000 ; Circular Buffer Base
LDA  a13, 0x00140012 ; Circular Buffer Limit and Index
...
st.b [a12]0x1, d2    ; Store to byte offset 0x1
ld.w d6, [a12/a13+c] ; Circular Buffer wrap, 16+16
...
```

In this example the circular buffer base address is quad-word aligned but the buffer size is an odd number of words (0x14 = 5 words). The byte store to address 0xD0001001 is immediately followed by a load operation and is placed in the CPU store buffer. The word load instruction encounters the circular buffer wrap condition and is split into 2 half-word accesses, to the top (0xD0001012) and bottom (0xD0001000) of the circular buffer. The first load access completes correctly, but, due to the bug, the second access overtakes the store operation and returns the previous half-word from 0xD0001000.

Workaround

For any circular buffer data structure, if byte store operations (st.b) are not used targeting the circular buffer, or if the circular buffer has a quad-word aligned base address and is an even number of words in depth, then this problem cannot occur. If these restrictions and the other conditions required to trigger the problem cannot be ruled out, then any load word instruction (ld.w) targeting the buffer using circular addressing mode, and which may encounter the circular buffer wrap condition, must be preceded by a single NOP instruction.

```
...
LDA  a12, 0xD0001000 ; Circular Buffer Base
LDA  a13, 0x00140012 ; Circular Buffer Limit and Index
...
st.b [a12]0x1, d2    ; Store to byte offset 0x1
nop                  ; Workaround
ld.w d6, [a12/a13+c] ; Circular Buffer wrap, 16+16
...
```

CPU_TC.110 Register Banks may be out of sync after FCU Trap

In order to improve the performance of Upper Context Save and Restore operations (Call, Interrupt etc.) the current TriCore1 CPU implementation contains shadow registers for the upper context General Purpose Registers (GPRs), `D8`-`D15` and `A10`-`A15`, forming a foreground and a background bank. In normal operation read and write accesses to the upper context registers target the same bank, with read and write accesses targetting different banks just during upper context save and restore operations.

However, in a certain corner case where an FCU trap is taken, read and write accesses to the register banks remain out of synchronisation in the FCU trap handler and cannot be easily re-synchronised. Since FCU traps are non-recoverable system errors, with some system state already lost, maintaining correct behaviour is not critical. However, due to the bug, it is no longer straightforward to discriminate FCU traps from other context management (Class 3) traps. Since the read and write pointers to the register banks are incorrect in the bug situation, the update of `D15` with the Trap Identification Number (TIN) will write to one bank whilst the read of `D15` in the trap handler will read the other (incorrect) bank, returning an invalid TIN. For similar reasons, the upper context GPRs are unusable in an FCU trap handler, since register read and write operations may target different banks.

The problem occurs in the following situation:

- `FCX` (Free Context Pointer) points to an invalid location (Null - End of CSA list, Invalid Segment - Virtual or Peripheral segment).
- CALL / CALLA / CALLI instruction is in the decode pipeline stage and would generate an FCU trap due to the invalid `FCX` pointer.
- Instruction in the Load-Store pipeline execute stage encounters a synchronous trap condition (VAF-D, VAP-D, MPR, MPW, MPP, MPN, ALN, MEM, DSE, SOVF, OVF), which would also be converted into an FCU trap.

Workaround

The LCX (Free Context List Limit) pointer should be initialised in order to trap impending context list overflow before the FCU condition is encountered. However, in order to maintain some system function in the case of an FCU trap, the following workaround is required, split into two parts.

Firstly, the Context Management (Class 3) trap handler must be modified to discriminate FCU traps that incorrectly appear to have a TIN pertaining to another Class 3 trap due to the bug. This is done by checking for the correct behaviour of the upper context registers and jumping to the FCU trap handler if the register file behaviour is found to be in error:

```
_class3_handler:
  mov  d12, #7
  nop
  nop
  jne  d12, #7, _fcu_handler
  mov  d12, #-8
  nop
```

```
nop
jne  d12, #-8, _fcu_handler
; Now read valid D15 to obtain TIN
...
```

Since the initial contents of the upper context registers are unknown, it is necessary to check one of the upper context registers twice, with different values, in case the initial contents match the first value to be checked.

Note: The NOP instructions in the above code are mandatory to ensure that reads from the GPRs target the register file directly, rather than the forwarding paths which always function correctly.

Secondly, within the FCU trap handler itself, only the global and lower context registers may be used, `D0-D7` and `A0-A9`. Since the upper context information is already lost in an FCU trap condition, usage of the global and lower context registers without previously saving this information is acceptable.

CPU_TC.111 Imprecise Return Address for FCU Trap

The FCU trap is taken when a context save operation is attempted but the free context list is found to be empty, or when an error is encountered during a context save or restore operation. In failing to complete the context operation, architectural state is lost, so the occurrence of an FCU trap is a non-recoverable system error.

Since FCU traps are non-recoverable system errors, having a precise return address is not important, but can be useful in establishing the cause of the FCU trap. The TriCore1 CPU does not generate a precise return address for an FCU trap if the cause of the FCU trap was one of the following trap types: FCD, DAE, DIE, CAE or NMI.

In each of these circumstances the return address may be invalid.

Workaround

None

CPU_TC.113 Interrupt may be taken during Trap entry sequence

A problem exists whereby interrupts are not correctly disabled at the very beginning of a trap entry sequence, and under certain circumstances an interrupt may be taken at the start of a trap handler. The problem occurs when an interrupt request is received by the TriCore CPU within a window spanning a single clock cycle either side of a trap condition being detected, and where interrupts are enabled and the interrupt priority number is higher than the current CPU priority number (CCPN). In this case the trap entry sequence begins and the upper context registers are stored to the appropriate CSA. However, before the first instruction of the trap handler is executed the interrupt condition is detected and the interrupt handler entered at a time when interrupts should be disabled. This problem affects all trap types but does not affect interrupts - an interrupt cannot be taken during the entry sequence of another interrupt.

It should be noted that no state information is lost when this issue occurs. When the interrupt handler completes and the RFE instruction is executed, program execution restarts with the first instruction of the interrupted trap handler and the trap handler then continues as normal.

The main issue associated with this problem is that the handling of the interrupt will delay the start of the trap handler. For the majority of trap types associated with the program flow this is not a problem. However, where the interrupted trap type denotes a serious system problem, such as an NMI trap, the delay in servicing the trap may be of concern. In addition, if interrupts are re-enabled within the interrupt handler then the delay in returning to the trap handler will be further extended by the handling of any additional higher priority interrupt requests which may occur. However, once processing of the initial interrupt handler is complete and the RFE instruction executed to return to the trap handler, interrupts are correctly disabled immediately and the trap handler will continue, even if further interrupts are pending when the RFE instruction is executed.

Another point to note is that this issue can cause some assumptions made in system software to be invalid. For example, if a system does not allow interrupts or traps to re-enable interrupts - then it would have been safe to assume that whenever a trap or interrupt is entered, that the code that has been suspended (and hence the state information saved in the CSA) is for a user task and typically non-privileged. Unfortunately, with this issue that premise no longer holds - the code interrupted and state saved in a CSA can be that of a privileged trap handler. Dealing with this changed circumstance is easy, provided it is considered whenever CSA's are examined or manipulated.

Workaround

As described previously, the main problem associated with this erratum is the delay that may be incurred before the servicing of certain critical trap types, such as NMI, if no additional action is taken. If this is an issue for a system, then in order to minimize the impact of this erratum it is necessary to adapt the interrupt handlers to check for the occurrence of this issue and react accordingly.

The occurrence of this issue may be checked for by one of two methods, dependent upon whether all trap classes or just a limited set are considered timing critical.

If it is required to check for the occurrence of this issue for all trap classes, then this may be performed by checking the value of the `PCXI.PIE` register bit within the interrupt handlers, before any further context operations (such as BISR) are performed. If `PCXI.PIE` is clear, such that no interrupt should have been taken, then this indicates the occurrence of this issue. Although when this method is used then it is preferred to check the value of `PCXI.PIE` before any further context operations are performed, it is possible to use this method after additional context operations have been performed. In this case it is necessary to traverse the CSA list to check the required `PCXI.PIE` value from the appropriate saved context.

If it is necessary within a system to check for the occurrence of this issue just for specific, timing critical, trap classes, then this may be performed by examination of the return address, held in the `A11` register, within the interrupt handler and comparing this return address against the trap vector address(es). For example, if only the NMI trap is a system issue requiring immediate action, the following code may be added to the interrupt handler to determine if the interrupt was taken at the start of the NMI handler:

```
...
< Timing critical section of Interrupt Handler >
movh.a a12, #@his(NMITrapAddress)       ; BTV OR 0xE0
```

```
  lea    a12, [a12]@los(NMITrapAddress)  ; BTV OR 0xE0
  eq.a   d13, a12, a11    ; Compare with A11, result in d13
  < Call / Branch to NMI handler based on d13 result >
```

Note that this code segment assumes that the `BTV` CSFR is static during runtime. If this is not the case then it would be necessary to determine the trap offset address during runtime by reading the `BTV` CSFR and ORing with the TCN offset of the trap of interest. If more than one trap class is considered timing critical within a system, it is possible to adapt the previous code to check the return address of the interrupt handler against a number (or range) of trap class entry addresses.

If the interrupted traps are considered recoverable, and are not time sensitive, the interrupt handler can simply complete and it's terminating RFE will correctly return execution to the first instruction of the trap handler - where it will now execute to completion without undesired interruption. If the interrupted traps are considered recoverable but are time sensitive and need to be executed immediately, then some method of deferring the interrupt processing is required. If the test of the situation (e.g. checking the `PCXI.PIE` bit is clear) is at the start of the Interrupt handler, then there are two simple methods to consider:

The first method would be to re-request the interrupt, by writing the appropriate Service Request Control Register with the SETR bit set to one, and then executing an RFE which will be taken back to the trap handler. The interrupt will now be pending again, but will not be taken until the trap handler executes its RFE to re-enable interrupts. This method is simple if the device (and hence it's SRC register address) generating the interrupt is known. If this is not easy to determine (statically or dynamically), the second method might be preferred.

The second method would be to jump to the trap handler, after setting the trap identification number (TIN) and the return address (which the trap handler will use) to be the next instruction in the interrupt handler. This relies on the fact that the CSA saved away by the preemption of the trap handler is equally valid as an execution context for the interrupt handler. The code for this method is as follows:

```
interruptN:
  mfcr   d15, PCXI
  jnz.t  d15, 23, interruptReal
  ; force CSA into memory
  dsync
  sh.h   d14, d15, #12
  insert d15, d14, d15, #6, #16
  mov.a  a15, d15
  ; load trap value of d15 from CSA
  ld.w   d15, [a15]0x3C
  mov.a  a15, a11
  movh.a a11, #@his(interuptReal)
  lea    a11, [a11]@los(interruptReal)
  ; jump to trap handler, will return to interruptReal
  ji      a15
  ; the remaining part of the interrupt handler
interruptReal:
  ...
```

Again it is preferred that this method be used immediately at the beginning of the interrupt handler, since this approach works straightforwardly provided there is no state in the Upper Context registers or on the interrupt stack that is required by the interrupt handler when it returned to. Although it is possible to adapt this approach to operate later during interrupt handling, additional steps need to be taken to ensure the correct state is maintained when returning to the interrupt handler.

CPU_TC.114 CAE Trap may be generated by UPDFL instruction

UPDFL is a User mode instruction implemented as part of the TriCore1 Floating-Point Unit (FPU), which allows individual bits of the `PSW` user status bits, `PSW[31:24]`, to be set or cleared. Contrary to early revisions of the TriCore1.3.1 architecture manual, and in contrast to most other FPU instructions, the UPDFL instruction should not generate Co-Processor Asynchronous Error (CAE) traps. However, in certain circumstances the TriCore1.3.1 FPU will generate CAE traps for UPDFL instructions.

The TriCore1.3.1 FPU will generate a CAE trap upon execution of the UPDFL instruction in the following situation:

- After execution of the UPDFL instruction, one or more of the `PSW[31:26]` bits are set - either the `PSW` bit(s) are set by UPDFL or were set prior to execution and not cleared by the UPDFL instruction.
- FPU traps are enabled for one of the asserted PSW[31:26] bits, via the corresponding `FPU_TRAP_CON.FxE` bit being set.
- The `FPU_TRAP_CON.TST` CSFR bit is clear - no previous FPU trap has been generated without the subsequent clearing of `FPU_TRAP_CON.TST`.

Workaround

The UPDFL instruction is normally used in one of two situations:

- Clearing the FPU sticky flags held in `PSW[30:26]`.
- Setting the FPU rounding mode bits in `PSW[25:24]`.

In the first case, if all the `PSW[31:26]` bits are cleared by UPDFL, no CAE trap will be generated.

In the second case, UPDFL may still be used to set the FPU rounding mode, but in this case the remaining PSW bits, [31:26], must be cleared by UPDFL in order to avoid generation of an unexpected CAE trap.

In all other cases, where FPU traps are enabled, some other method of manipulating the PSW user status bits must be used in order to avoid extraneous CAE trap generation. For instance, if in Supervisor mode the PSW may be read using the MFCR instruction, the high order PSW bits modified and written back using the MTCR instruction.

CPU_TC.115 Interrupt may be taken on exit from Halt mode with Interrupts disabled

A problem exists whereby an interrupt may be taken by the TriCore CPU upon exiting Halt mode, even if interrupts are disabled at that point.

The problem occurs when an interrupt request is received by the TriCore CPU, with the pending interrupt priority number (PIPN) higher than the current CPU priority number (CCPN), and interrupts are enabled. In this case, where only the CPU pipeline status is preventing the interrupt from being taken immediately, the interrupt is latched and taken as soon as the pipeline can accept an interrupt. This may cause unexpected behavior whilst debugging, where interrupts are enabled before entry to Halt mode, or where interrupts are temporarily enabled during Halt mode. In this case an interrupt may be latched whilst the CPU is in Halt mode, and subsequently disabling interrupts during Halt mode, by setting `ICR.IE` = 0_B, will not prevent the interrupt from being serviced immediately upon exit from Halt mode.

It should be noted that no corruption of the program flow is associated with this issue and that it affects debugging only, primarily the debugger single-stepping functionality. The problem may or may not be visible whilst debugging, dependent upon the implementation of single-stepping by the debugger. If single-stepping is implemented by the debugger setting Break-Before-Make (BBM) breakpoints on all instructions except the next to be executed, then if this problem occurs the next instruction when single-stepping will be the first instruction of the interrupt handler. However, if single-stepping is implemented by setting a Break-After-Make (BAM) breakpoint on the next instruction to be executed, or a BBM breakpoint on the next but one instruction, the problem will not be visible. In this case, when single-stepping, the interrupt handler will be executed in its entirety before returning to the interrupted program flow and the breakpoint being taken after the next instruction to be single-stepped.

Workaround

As described previously, this problem affects debug only and in this case the taking of the interrupt immediately upon exit from Halt mode cannot be avoided if the conditions to trigger the problem occur. However, the debugger single-stepping functionality may be implemented in such a way that this problem does not directly affect the user, as follows:

Upon first hitting a breakpoint, the debugger should read and hold the current interrupt enable status from `ICR.IE`. Interrupts should then be disabled by setting `ICR.IE` = 0_B.

If the next debugger action is to single-step, a BAM breakpoint should be placed on the next instruction to be executed and the CPU re-started. In this case a previously latched interrupt may be serviced, but will not result in a further breakpoint being flagged until the interrupt handler returns and the next instruction intended to be single-stepped is executed.

Subsequent single-step operations may be implemented using any appropriate method, since interrupts will be disabled before Halt mode is entered.

If the debugger action is to re-start normal execution, the interrupt enable status should be restored from the value read upon hitting the initial breakpoint and the CPU re-started.

CPU_TC.117 Cached Store Data Lost on Data Cache Invalidate via Overlay

Cached store data can be lost if the overlay system requests a data cache invalidate in the same cycle as a cache line is being written. The overlay control provides a mechanism to do a single cycle invalidate of all valid/clean lines in the data cache by writing the OCON.DCINVAL bit. Please note that there is no problem if the data cache is used exclusively for read data (e.g. flash constants).

```
Cache line state transition on DCINVAL.
   valid/clean -> (DCINVAL) -> invalid/clean
```

A normal store operation transitions the cache line to a valid/dirty state.

```
Cache line state transition on normal store operation.
    valid/clean -> (write) -> valid/dirty
    invalid/clean -> (write) -> valid/dirty
```

In the case where the write and invalidate are received in the same cycle, the dirty bit is correctly updated but the valid bit is incorrectly cleared.

```
Cache line state transition on store operation with DCINVAL
     valid/clean -> (write+DCINVAL)- > invalid/dirty
     invalid/clean -> (write+DCINVAL) -> invalid/dirty
```

This leads to a loss of data as the store data ends up being held in an invalid cache line and hence never re-read.

Workaround-1

Ensure that the data cache is never used to cache write data. This can be ensured by software design but may limit performance in some systems.

Workaround-2

Ensure that the core is never storing data when OCON.DCINVAL is asserted.

This requires the CPUs store buffers to be empty when the invalidate is asserted. This can only be done by getting the CPU to firstly flush all write data with a DSYNC command, then to write the OCON.DCINVAL to trigger an invalidate.

The following example code sequence performs the required operations:-

- Read the OCON register to get the current SHOVEN field
- Create a new OCON value with DCINVAL, OVSTRT and OVCONF bits set
- Perform a DSYNC operation to flush all write data to memory
- Write OCON with the new value.
- Read back OCON to ensure write is complete

```
;; Set up A14 with address of OCON Register
movh.a  a14,#(((0xF87FFBE0)+0x8000>>16) & 0xffff)lea
a14,[a14]((((0xF87FFBE0)+0x8000)&0xffff)-0x8000)
;; Load a15 with contents of OCON
ld.w   d15, [a14]
;; Set OCONF, DCINVAL, OVSTRT start values
movh   d14  , #0x0305
;; Combine existing SHOVEN
insert d15, d14,d15,#0,#16
; Flush all store data
dsync
;; Store New value back to OCON
st.w   [a14], d15
;; Re-read to ensure store is complete
ld.w   d15, [a14]
```

Attention: This routine must be run with interrupts disabled, either as part of an interrupt service routine or guarded by enable/disable instructions.

This routine may be run periodically or run as part of a dedicated interrupt service routine. If the latter approach is used it is suggested that an unused

SRN either in the CPU or Cerberus is utilised to trigger the invalidate. In all cases the routine must be run with interrupts disabled to ensure that no writes are in progress when the invalidate occurs.

The OCON.OVCONF bit may be used to indicate the state of the invalidate operation. If it is cleared in advance, the routine above will set it when the cache invalidate operation is performed.

DMA_TC.013 DMA-LMB-Master Access to Reserved Address Location

DMA-LMB-Master goes into an unintentional lock-up state when a Read or Write access is made to a reserved memory location with an unrecognised slave.

Subsequent Read/Write accesses from a DMA Channel, MLI, Cerberus or the DMA-FPI-Slave to all memory locations mapped to the DMA-LMB-Master (80000000_H to $DFFFFFFF_H$) will be halted until control of the DMA-LMB-Master is regained.

In the case of a lock-up DMA-LMB-Master Read access, the next LMB access and associated response will have the following effect:

- ERROR Response: The DMA-LMB-Master will treat this error response as its own. It will clear the lock-up state and return an error to the DMA access requester. Normal operation will then continue. Halted DMA access requests will resume. There is no corruption of the data flow.
- NSC (No Special Condition) Acknowledge: The DMA-LMB-Master will treat this response as its own and again clear the lock-up state. The correct response to an unrecognised slave is an ERROR. Therefore the DMA-LMB-Master has signalled an invalid response back to the DMA access requester resulting in a corruption of the data flow.
- RETRY Response: The DMA-LMB-Master will treat the retry response as its own and again clear the lock-up state. The access will be repeated to the same reserved address location again resulting in a lock-up condition. The sequence is broken by the first ERROR response or NSC acknowledge.

The effect of a DMA-LMB-Master Write accesses to an unrecognised slave is the same as above with one exception:

- If the next access is a Read access from the EBU-LMB-Slave then the DMA-LMB-Master will clear the lock-up state and respond as above. The EBU read completes but the data read by the Originator (e.g. TriCore) will be the write data of the DMA-LMB-Master Write access.

The following should be noted:

- At all times the DMA-FPI-Master and DMA-FPI-Slave remain accessible.
- If the LMB-DMA-Master is in the lock-up state then accesses can still be made to the LMB bus by all other LMB-Masters (e.g. LFI-LMB-Master).

Hint

Do not perform a DMA channel, MLI, Cerberus or DMA-FPI-Slave access to a reserved address: all areas specified as reserved in the Memory Map Chapter, LMB Address Map Table must not be accessed by the DMA (ME, MLI, Cerberus).

Workaround

The LMB-Bus-Control-Unit can recognise a DMA-LMB-Master access to an unrecognised slave. It can be programmed to raise an interrupt and then generate a Class 3 Application Reset to clear the lock-up state.

DMI_TC.014 Problems with Parity Handling in TriCore Data Memories

A small number of cases exist in which the handling of parity errors in the TriCore data memories (LDRAM, DCache and Data Cache Tag) does not function correctly, potentially leading to data corruption for accesses to these memories. This data corruption may occur whether the access to one of these memories is from the TriCore CPU, or, in the case of LDRAM, from another bus master access via the LMB.

Workaround

In systems where the Data Memory parity handling must be enabled, the following is required to guarantee correct behaviour:

- Compatibility mode must be selected for the TriCore Data side memories by setting `COMPAT.DIE` = 1_B. In this case parity errors are signalled to the SCU and returned to the CPU as an NMI trap, rather than as a DIE trap directly to the CPU.

AND

- If the system has a data cache, the data cache must be used to cache read-only data only (such as Flash contents). Writes to cacheable locations must not be used with the Data Cache enabled.

Note that this does not concern the program side which works as expected.

DMI_TC.015 LDRAM Access Limitations for 2KByte Data Cache Configurations

TriCore1.3.1 based devices are physically built with a certain size of Data Memory (DMEM) and a data tag memory to support a certain maximum size of Data Cache (DCache). Within these physical limitations, software may select the exact split between LDRAM and DCache where DMEM size = LDRAM size + DCache size. The software selection is performed according to the configuration of the DCache size in `DMI_CON.DC_SZ_CFG`, with any DMEM not configured as DCache ordinarily available as LDRAM.

However, a problem exists where the DCache is configured to be 2KByte, DMI_CON.DC_SZ_CFG = 0001_B. In this case the expected amount of LDRAM is available for accesses from the CPU (DMEM size - 2KByte), but the address range checking is incorrect for accesses to LDRAM from the LMB and the available LDRAM size for LMB accesses is limited to (DMEM size - 4KByte).

Example

A TC1767 device is physically built to support a maximum of 72KByte DMEM and 4KByte DCache. Where the DCache size is configured as 4KByte, available LDRAM is 68KByte, where the DCache size is configured as 0KByte, available LDRAM is 72KByte. However, when the DCache size is configured as 2KByte, 70KByte LDRAM is addressable by the CPU, but only the bottom 68KByte is addressable by LMB bus masters.

Workaround

In systems where a 2KByte DCache is configured, the top 2KByte of LDRAM is only available for usage by the CPU, and cannot contain data structures that may be required by other bus masters. For instance, this space could be used as part of the CSA list. However, note that since this memory is not addressable by LMB masters in the 2KByte DCache configuration, this would affect debuggers. Hence it would only be possible to view this memory space in a debugger if it takes appropriate steps to make the memory region accessable (e.g. by temporarily setting the DCache size to 0KByte) to examine that address range.

DMI_TC.016 CPU Deadlock possible when Cacheable access encounters Flash Double-Bit Error

A problem exists whereby the TriCore CPU may become deadlocked when attempting a mis-aligned load access to a cacheable address. The problem will be triggered in the following situation:

- The TriCore CPU executes a load instruction whose target address is not naturally aligned - a data word access which targets an address which is not word aligned, or a data / address double-word access which is not double-word aligned.
- The mis-aligned load access targets a cacheable address, whether the device is configured with a data cache or not.
- The mis-aligned load access spans two halves of the same 128-bit cache line. For instance, a data word access with address offset 6_H.
- The mis-aligned load access results in a cache miss, which will refill the 128-bit cache line / Data Line Buffer (DLB) via a Block Transfer 2 (BTR2) read transaction on the LMB, and this LMB read encounters a bus error condition in **the second beat of the block transfer**.

It should be noted that under normal operation, LMB block transfers will not result in a bus error condition being flagged on the second beat of a block transfer. However, such a condition may be encountered when accessing the on-chip Flash, if the second double-word of data accessed from the Flash (for the second half of the cache line) contains an uncorrectable double-bit error.

When this condition is triggered, the first part of the requested data is obtained from the valid first beat of the BTR2 transfer, and the second part is required from the errored second beat. In this case, no error is flagged to the TriCore CPU and the transaction is incorrectly re-started on the LMB. In the case of a Flash double-bit error, this transaction will be re-tried continuously on the LMB by the DMI LMB master and the CPU become deadlocked. This situation would then only be recoverable by a Watchdog reset.

The problem exists within the DMI DLB, which is used as a single cache line when no data cache is configured, and as a streaming buffer when data cache is present. As such the problem affects all load accesses to cacheable locations, whether data cache is configured or not, since the DLB is used in both cases.

Note: This problem affects load accesses to the on-chip Flash only. Instruction fetches which encounter a similar condition (bus error on later beat of block transfer) behave as expected and will return a PSE trap upon any attempt to execute an instruction from a Flash location containing a double-bit error.

Workaround

As described previously, this problem should not be encountered during normal operation and will only be triggered in the case of a double-bit error being detected in an access to the on-chip Flash.

However, in order to remove the possibility of encountering this issue, all load accesses to cacheable addresses within the on-chip Flash should be made using natural alignment - word transfers should be word aligned, double-word transfers double-word aligned.

It is also possible to check for the occurrence of this problem by having some other master, such as the PCP, periodically poll the LBCU `LEATT` register to check for the occurrence of LMB error conditions, specifically if one is detected during a BTR2 read transfer from the DMI, as reported by `LEATT.OPC` and `LEATT.TAG`.

DMI_TC.017 DMI line buffer is not invalidated by a write to OVC_OCON.DCINVAL if cache off.

A problem exists whereby the DMI line buffer is not invalidated by a write to OVC_OCON.DCINVAL when operating with the D-cache turned off. This means that the user cannot rely on a write to OVC_OCON.DCINVAL to make sure that any stale data in the DMI line buffer is invalidated. This can be a problem for users who want to use the OVC_OCON.DCINVAL bit to ensure coherency between the DMI and background memory.

It should be noted that this problem is not encountered when the D-cache is turned on. When the D-cache is turned on, writing a one to OVC_OCON.DCINVAL will correctly invalidate all clean cache entries and invalidate the DMI line buffer. The problem only concerns systems with no cache or systems where the cache is turned off.

Detailed description

D-Cache turned on:

When D-cache is turned on, the DMI line buffer is only used as a performance enhancement mechanism with no logical existence to the user. It is therefore not operating as a micro-cache and the current issue does not apply. When the dcache is turned on, writing to OVC_OCON.DCINVAL will always invalidate all clean lines in the dcache. No stale data will subsist in the DMI line buffer.

D-Cache turned off:

The problem occurs when the dcache is turned off. When the dcache is turned off (or non-existent) the DMI line buffer operates as a 16-byte cache. Writing a one to the OVC_OCON.DCINVAL register should invalidate the data inside the DMI line buffer as long as the data is not dirty. This invalidation mechanism does not work on AUDO-Future devices. Writing to OVC_OCON.DCINVAL will have no effect at all. Any cache line which was previously loaded into the DMI line buffer will not be invalidated (whether it was dirty or not).

Workaround

The workaround consists in executing a cachei.wi instruction with an operand register containing a random non-protected cacheable address. The DMI line buffer will respond to cachei.wi instructions regardless of the content of its operand, provided that the operand contains a cacheable address which is not protected. On execution of cachei.wi, the DMI line buffer will flush and invalidate itself. For example, executing the following two instructions should flush and invalidate the DMI line buffer in any circumstance. Note that the current workaround always invalidates the entry regardless of whether it was dirty or not.

```
movh.a a0, #0x8000 ;; Cachei operand is random non-
protected cacheable address.
cachei.wi [a0]     ;; The DLB gets invalidated regardless
of the value in a0.
```

If the user is not concerned in invalidating the DMI line buffer but simply guaranteeing its coherency with external memory then there is another simple workaround. This consists in issueing a read to a dummy cacheable address pointing outside the 16-byte block containing the next required data. Access to the next required data will then necessarily result in a refill and the resulting data will be coherent. This is what the following code does (a0 contains a dummy address and a1 contains the address for the user's required data).

```
movh.a a0, #0x8000 ;; Dummy address is 0x80000000.
ld.w d0, [a0]      ;; a0 has to point to different 16-byte
block than a1.
ld.w d0, [a1]      ;; This load will be executed fresh from
memory with a refill.
                   ;; Read data will be coherent with rest
of memory.
```

FADC_TC.005 Equidistant multiple channel-timers

The description is an example for timer_1 and timer_2, but can also affect all other combinations of timers.

Timer_1 and Timer_2 are running with different reload-values. Both timers should start conversions with the requirement of equidistant timing.

Problem description:

Timer_1 becomes zero and starts a conversion. Timer_2 becomes zero during this conversion is running and sets the conversion-request-bit of channel_2. At the end of the conversion for channel_1 this request initiates a start for channel_2. But the Timer_2 is reloaded only when setting the request-bit for channel_2 and is decremented during the conversion of channel_1.

The correct behavior would be a reload when the requested conversion (of channel_2) is started.

Therefore the start of conversion for channel_2 is delayed by maximum one conversion-time. After this delay it will be continued with equidistant conversion-starts. Please refer to the following figure.

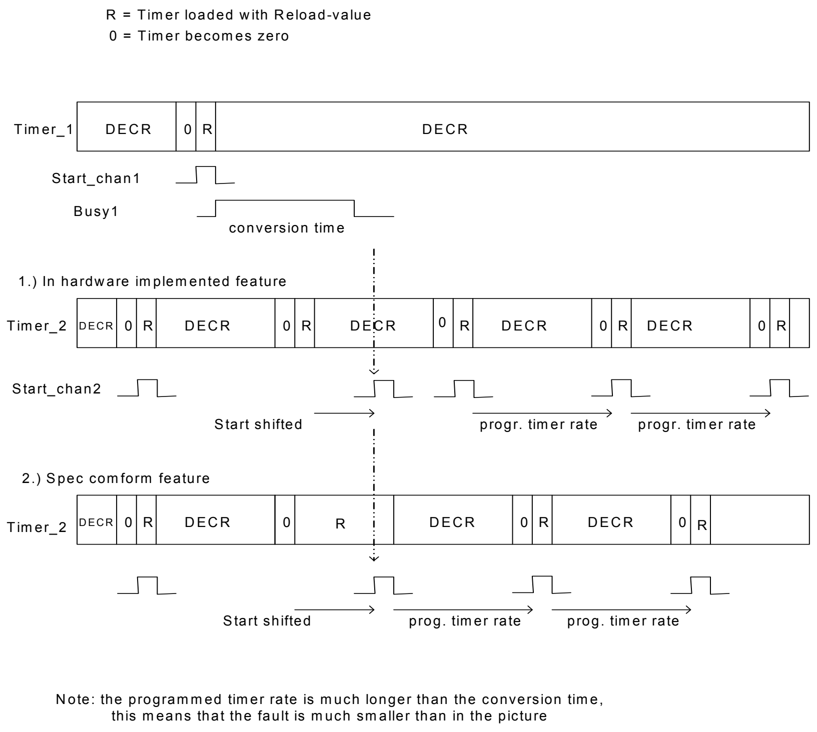

Figure 1 Timing concerning equidistant multiple timers

Workaround

Use one timer base in combination with neighboring trigger and selection by software which result has to be taken into account.

FIRM_TC.010 Data Flash Erase Suspend Function

This problem affects devices with microcode version V11 (see FIRM_TC.H000 for identification of the microcode version):

A sector DFx in the Data Flash may not be correctly erased when two successive erase operations are executed **without** any programming between the erase operations, as described in the following sequence (x, y = 0 or 1, x ? y):

1. A Program Flash sector or a Data Flash sector (DFy or DFx) has been erased.
2. An erase command on Data Flash sector DFx is issued.
3. While the erase operation on sector DFx is in progress, during a certain critical time window a programming command on sector DFy is issued, i.e. the erase operation on sector DFx is suspended.

In other words, potentially critical sequences are:

- Erase PFLASH --> erase DFx --> program DFy (suspend erase DFx), or
- Erase DFy --> erase DFx --> program DFy (suspend erase DFx), or
- Erase DFx --> erase DFx --> program DFy (suspend erase DFx).

As a consequence, sector DFx may not be correctly erased after the suspended erase has been completed (i.e. DFx may be weakly programmed). The effect is non-permanent, i.e. erasing DFx again will solve the issue.

Note: Sector DFy is always correctly programmed.

Therefore, both Data Flash sectors or a Program and a Data Flash sector must not be erased one after the other if the second erase operation might be suspended.

Workaround 1

Additionally program (a page of) Data Flash sector DFx or DFy, before starting the erase of DFx, e.g.:

1. Erase PFLASH or DFx or DFy
2. **Additional step**: Program DFx or DFy (specified rules for Data Flash page programming must remain valid), check for completion of programming (busy)
3. ... (any operation except erase of DFx, DFy, or PFLASH)...
4. Erase DFx
5. Concurrent programming of DFy may be triggered.

Workaround 2

After starting the erase of Data Flash sector DFx, delay the start of the programing of sector DFy by at least 250 ms, e.g.:

1. Erase DFx
2. **Additional step**: Wait > 250 ms
3. Concurrent programming of DFy may be triggered.

Workaround 3

Issue a reset in case of two consecutive erase operations without intermediate programming, e.g.:

1. Erase DFx
2. Erase DFy
3. **Additional step**: Reset
4. Erase DFx (if needed)
5. Concurrent programming of DFy may be triggered.

Workaround 4

After a concurrent erase on Data Flash sector DFx has been triggered, verify the state of DFx.

In case of weak programming, re-erase DFx (**Additional step**).

Workaround 5

Do not use concurrent erase/program operations on Data Flash.

FLASH_TC.027 Flash erase time out of specification

As per specification in the Data Sheet following are the flash erase timings:

Table 7 Flash erase timings as per spec.

Flash	Micro code version	Erase Time
P-Flash, 2 MByte	V11	40s [at cold and room temperature]
D-Flash, 64 Kbyte [Both Data Flash]	V11	2.5s [at cold temperature]

Actual Flash erase timings may reach the following maximum values. A minimum erase time budget per erase operation of 0.5 s must however be tolerated regardless of size-proportional erase times derived from the table.

Table 8 Actual Flash erase timings.

Flash	Micro code version	Erase Time
P-Flash, 2 MByte	V11	52s [at cold temperature]
		45s [at room and above temperature]
D-Flash, 64 Kbyte [Both Data Flash]	V11	3.2s [at all temperature]

Maximum erase time at various CPU operating frequencies can be calculated according to the following table. Frequency dependency for Data Flash is lower than for Program Flash.

Table 9 Relative erase time increments.

Frequency [MHz]	Increment for P-Flash	Increment for D-Flash
66	10%	6%
80	6%	4%
100	3%	1.5%
133	0%	0%

<u>FLASH_TC.035</u> Flash programing time out of specification

As per specification flash programing time specified is per page 5msec

Where as actual programing time measured on the device is per page 5.5msec

<u>MCDS_TC.021</u> Incorrect wiring between TQU_MCX and TQU_TC

The MCX Action registers are wired not as specified, but as given in **Table 10** below: MCXACT0-8 have two functions concurrently, MCXACT31-39 are not connected.

Table 10 Assignment

Register	Purpose	Function
MCXACT0	tsu_rel_en, tc_trig[0]	TSU emulation clock time stamp enable, Trigger 0 from TQU_MCX to TQU_TC
MCXACT1	tsu_rel_sync, tc_trig[1]	TSU emulation clock sync message request, Trigger 1 from TQU_MCX to TQU_TC
MCXACT2	tsu_abs_en, tc_trig[2]	TSU reference clock time stamp enable, Trigger 2 from TQU_MCX to TQU_TC
MCXACT3	tsu_abs_sync, tc_trig[3]	TSU reference clock sync message request, Trigger 3 from TQU_MCX to TQU_TC
MCXACT4	wtu_enable[0], tc_evt[0]	WTU_MCX watch-point message request ID 0, Enable 0 from TQU_MCX to TQU_TC

Table 10 Assignment (cont'd)

Register	Purpose	Function
MCXACT5	wtu_enable[1], tc_evt[1]	WTU_MCX watch-point message request ID 1, Enable 1 from TQU_MCX to TQU_TC
MCXACT6	wtu_enable[2], tc_evt[2]	WTU_MCX watch-point message request ID 2, Enable 2 from TQU_MCX to TQU_TC
MCXACT7	wtu_enable[3], tc_evt[3]	WTU_MCX watch-point message request ID 3, Enable 3 from TQU_MCX to TQU_TC
MCXACT8	wtu_enable[4], tc_evt[4]	WTU_MCX watch-point message request ID 4, Enable 4 from TQU_MCX to TQU_TC
…	…	…
MCXACT31	open	no function
MCXACT32	open	no function
MCXACT33	open	no function
MCXACT34	open	no function
MCXACT35	open	no function
MCXACT36	open	no function
MCXACT37	open	no function
MCXACT38	open	no function
MCXACT39	open	no function
…	…	…

Workaround

No functionality is lost, just use MCXACT0-8 wherever MCXACT31-39 would be used and ignore the additional messages caused by the original action assigned of MCXACT0-8.

MCDS_TC.022 PTU error count not always correct

The counter responsible for the parameter of the PTU.ERR message increments by no more than one each MCDS clock cycle. If the PTU FIFO is full and the TriCore executes two branches within one MCDS clock cycle only one branch is accounted for.

Note: This was only seen with TriCore executing empty loops yet.

Workaround

Run MCDS at the same clock rate as the fastest core.

<u>MCDS_TC.023</u> DTU Trigger Comparators don't keep last value

Inside the DTU trace units the address and data of each bus transaction are offered to comparators. These comparators' outputs are then forwarded to the TQUs for trace qualification purposes.

Between bus transactions and while an ongoing transaction is waiting for completion these comparators should not change state to allow construction of address range conditions in the TQU.

Currently the comparators revert to "false" on each clock cycle of MCDS which is not also a clock cycle of the observed bus.

Workaround

During the completion clock cycle of each transaction the comparator outputs are always valid.

It is therefore proposed to use the write indication in the access type comparator (e.g. SPBACRNG.RW) as enable (AND-function already implemented in EVT register) for any action based on the address and/or data comparators.

Ranges should be built as state machine states.

<u>MCDS_TC.024</u> MCDS_CT.RC does not reflect current setting

The source for the reference clock used by MCDS to generate absolute time stamps can be selected from two different sources. The selection is done by writing to the MCDS_CT.RC control bit.

This selection does work as intended, but the value read back from the control bit is stuck at zero. Tools therefore cannot rely on this bit to check and/or save the current setup of MCDS.

Workaround

As the functionality (selection of a reference clock source) is not affected, only the data management inside the tools needs to be adapted. It is proposed to store the current setting of MCDS_CT.RC also in an internal variable and use this instead of the register bit for reading.

Note: As rewriting MCDS_CT.RC with the same value does not cause any side effects, the internal variable can be copied to the register bit anytime "to make sure" the correct setting is active.

MCDS_TC.025 Tick counter sometimes inaccurate

The MCDS tick messages are intended to provide cycle accurate timing information with minimum memory footprint. According to the message format every 255 emulation clock cycles latest such a tick message is generated.

Due to a design bug a multick message with maximum value (i.e. no other message for 255 emulation clock cycles) which is generated at the precise point of time the TSU internal time stamp generator also reaches the count value 255 will cause the loss of one count for the next multick message.

Furthermore the next multick message can show an incremented count value if the current multick is generated at a point of time where the TSU internal time stamp generator approaches the count value 255.

Workaround

As the probability of this non-deterministic behavior is rather low (small sub-range of an eight bit counter plus intervals with no trace activity) and only the length (and not the existence) of a gap is affected, the consequences of this bug can be neglected in most cases.

If a missing or extra clock cycle does matter, explicit TSU messages of type TSR can be used to adjust the tick accumulator in the tool software.

<u>OCDS_AI.001</u> DAP restart lost when DAP0 inactive

To speed up the resynchronization after a loss of connection, the DAP module can be forced back to the “Enabled” (not yet “Active”) state by a control signal (“restart”) driven by the on-chip debug logic (e.g. CBS_OSTATE) and actuated by higher-level on-chip tool firmware.

In the current design this feature is implemented as synchronous reset, i.e. it requires clock edges inside the DAP module to work properly. As DAP0 is the only clock source used by DAP, the reset is not sensed if DAP0 is not toggled by the host at least once while restart is asserted.

Workaround

There is no workaround available.

Attention: Do not assert restart for longer periods of time unless the interface shall be functionally “locked”. The bug-fixed version of future devices will also reset DAP as long as restart is asserted, but will additionally store the rising edge until at least two DAP0 clock edges have been seen.

<u>OCDS_AI.002</u> JTAG Instruction must be 8 bit long

The JTAG TAP controller implemented in all Infineon devices strictly adheres to the standard IEEE 1149-1-2001. One side effect of this standard requires special awareness, as it can cause severe errors.

Upon entry to the **Capture-IR** state the internal shift register is preloaded with a constant, namely 01_H.

In the **Shift-IR** state the bits from the host are prepended, i.e. for each incoming bit the old LSB is dropped, the remaining 7 bits are shifted right one bit position and the incoming bit becomes the new MSB.

Upon entry to the **Update-IR** state the content of the internal shift register is copied into the INSTRUCTION register **unconditionally**.

If the final state of the shift register happens to be a valid, but unintended instruction, the device may enter a state very detrimental to the application. An extreme example is the INTEST instruction, which turns off all outputs of the device and is activated by instruction 01_H, i.e. if no bit at all is shifted in by the host!

Recommendations

- Always shift in at least as many bits as the INSTRUCTION register holds. This means 8 bit for Infineon devices.
- Check the bits returned via TDO: Must be 01_H followed by any data shifted in excluding the last eight bits. This allows to “check the pipe” by shifting in more than the required 8 bits.
- Use the protection offered by IOPATH: Keeping IOPATH different from 00_B whenever possible will block all Boundary Scan functions.
- Do not use the DAP telegrams jtag_setIR and jtag_swapIR with **n** less than eight.
- Use the CRC protected DAP interface if the application environment may cause transmission errors on the JTAG signals.

OCDS_TC.014 Triggered Transfer does not support half word bus transactions

The register bit CBS_IOCONF.EX_BUS_HW does not have any influence on the transaction width; only word wide transfer (32 bit) is implemented.

Workaround

No workaround possible. Choose source (IOADDR) and destination (ICTTA) addresses in word wide areas only.

OCDS_TC.015 IOCONF register bits affected by Application Reset

The IOCONF register is erroneously cleared by each Application Reset. Therefore Communication Mode is entered whenever the TriCore is reset.

As the interaction with the tool is suspended anyway due to Error State of the IOClient, no immediate damage is done.

To resume interaction after leaving the Error State (IO_SUPERVISOR instruction) however the required mode must be restored by rewriting the IOCONF register (IO_CONFIG instruction).

Workaround

After detecting an Application Reset (IOINFO.BUS_RST set) the IOCONF register should be rewritten by the tool after the Error State is left.

OCDS_TC.016 Triggered Transfer dirty bit repeated by IO_READ_TRIG

The dirty bit appended to the data of an IO_READ_WORD instruction during Triggered Transfer mode indicates that there was at least one extra trigger event missed prior to capturing the transmitted data. The dirty bit is therefore cleared after each IO_READ_WORD. A consecutive IO_READ_TRIG instruction however will erroneously undo the clear. The next IO_READ_WORD will then again see a set dirty bit even if no trigger was missed.

Workaround

Do not issue an IO_READ_TRIG instruction after an IO_READ_WORD returned a set dirty bit.

OCDS_TC.018 Startup to Bypass Mode requires more than five clocks with TMS=1

If the DAP state machine is brought to **Enabled** state by Power On Reset, the TAP controller is fed 0_B bits on its TMS input. When the special sequence for Startup in Bypass Mode is detected by DAP the TAP controller already has left the **Test-Logic-Reset** state.

Workaround

Shifting in more than five 1_B bits (recommendation: 10) will securely bring the TAP state machine back to **Test-Logic-Reset**.

OCDS_TC.020 ICTTA not used by Triggered Transfer to External Address

In “Triggered Transfer to External Address” Mode bits 24…0 of the target address are fixed to the reset value of ICTTA. Only the most significant byte can by changed by IO_SET_TRADDR (or by writing to ICTTA).

Note: This is the behavior of the Cerberus implemented prior to AudoNG.

It is therefore not possible to use Cerberus as “DMA” work-alike to move trace data to the outside world via an interface like ASC.

Workaround

No workaround in “Triggered Transfer to External Address” mode possible, only the fixed address xx10F068$_H$ can be used.

In “Internal Mode” however ICTTA is working as specified, so for certain use cases the intended DMA functionality can be activated by a code snippet executed by the TriCore or PCP as long as the Debug Interface is not needed concurrently.

OCDS_TC.021 TriCore breaks on de-assertion instead of assertion of break bus

The central OCDS building block “Cerberus” provides two Break Buses. All break sources can be individually enabled to assert one of these buses, while all break targets can be programmed to react on the assertion of one of the break buses.

The break target TriCore however does not react on the assertion (transition 0 to 1) of the break bus (as seen in the associated status bit MCDBBSS.BBSx) like all the other break targets, but on the deassertion (transition 1 to 0 of the status bit).

It is therefore not possible to:

- Stop the TriCore together with another core (e.g. PCP) at the same instant.
- Use a single transition of an external signal connected to any $\overline{\text{BRKIN}}$ pin to cleanly break the whole SoC.

Workaround

- All internal break sources can be programmed in a manner so that the break event is encoded as a pulse. For simple sources (e.g. SBCU) this is trivial, for complex sources (e.g. MCDS) a different trigger logic may be required. The temporal distance of the break requests to different targets can thus be kept low relative to the intrinsic core specific delays (e.g. caused by pipeline flushing).
- External break sources may be redesigned to deliver an active low pulse also.

OCDS_TC.024 Loss of Connection in DAP three-pin Mode

Devices of the Audo Future family allow tool access via dedicated pins in two basic protocol modes: DAP and JTAG. To avoid changes to the application environment, the tool selects the protocol to be used by signalling over the same pins later used for communication.

Default startup mode is two-pin DAP. Other modes (JTAG or three-pin DAP) are selected by specific telegrams which need to be sent to the device.

One of the major advantages of DAP versus JTAG within a harsh automotive environment is its robustness regarding bit errors in the telegram transmission. This is achieved by adding checksums (CRC) to all telegrams. The only exception is the telegram to switch from DAP to JTAG (see jtag_mode telegram), as this telegram is potentially sent by legacy JTAG-only tools not able to generate properly formatted DAP telegrams.

A method has been implemented to prevent the following safety hole: If bit errors (e.g. caused by EMC) change another DAP telegram to the telegram meaning “switch to JTAG mode”, a tool using DAP pins only would loose connection in an unrecoverable manner. Therefore the DAP module can be configured by the tool via SFR CBS_OSTATE.DJMODE to ignore the “switch to JTAG mode” (also called BYPASS) telegram.

Due to an imperfection within the design the intended protection via CBS_OSTATE.DJMODE does not become effective in three-pin DAP mode (CBS_OSTATE.DJMODE = 11_B).

Note: Two-pin DAP mode is not affected.

Workaround

None for three-pin DAP mode.

It is recommended to select the protected two-pin DAP mode (CBS_OSTATE.DJMODE = 01_B) instead if unintentional and non recoverable loss of the tool connection (e.g. due to EMC) shall be prevented safely.

OCDS_TC.025 PC corruption when entering Halt mode after a MTCR to DBGSR

In cases where the CPU is forced into HALT mode by a MTCR instruction to the DBGSR register, there is a possibility of PC corruption just before HALT mode is entered. This can happen for MTCR instructions injected via the CPS as well as for user program MTCR instructions being fetched by the CPU. In both cases the PC is potentially corrupted before entering HALT mode. Any subsequent read of the PC during HALT will yield an erroneous value. Moreover, on exiting HALT mode the CPU will resume execution from an erroneous location. .

The corruption occurs when the MTCR instruction is immediately followed by a mis-predicted LS branch or loop instruction. The forcing of the CPU into HALT takes priority over the branch resolution and the PC will erroneously be assigned the mispredicted target address before going into HALT.

- Problem sequence 1:
- 1) CPS-injected MTCR instruction to DBGSR sets HALT Mode
- 2) LS-based branch/loop instruction
- 3) LS-based branch/loop is mispredicted but resolution is overridden by HALT.
- Problem sequence 2:
- 1) User code MTCR instruction to DBGSR sets HALT Mode
- 2) LS-based branch/loop instruction
- 3) LS-based branch/loop is mispredicted but resolution is overridden by HALT.

Workaround

External agents should halt the CPU using the BRKIN pin instead of using CPS injected writes to the CSFR register. Alternatively, the CPU can always be halted by using the debug breakpoints. Any user software write to the DBGSR CSFR should be followed by a dsync.

<u>OCDS_TC.026</u> PSW.PRS updated too late after a RFM instruction.

When a breakpoint with an associated TRAP action occurs, the Tricore will enter a special trap called a ’debug monitor’. The RFM instruction (return from monitor) is used to return from the debug monitor trap. After the RFM, the CPU should resume execution at the point where it left it when the breakpoint happened.On execution of the RFM instruction, a light-weight debug context is restored and the PSW CSFR is loaded with its new value. The updated value of the PSW.PRS field should then be used to select the appropriate protection register set for all subsequently fetched instructions. Because PSW.PRS can be updated too late after an RFM instruction, the instruction following an RFM potentially sees the old value of the PSW.PRS field as opposed to the new one.This can be problematic since the PSW.PRS field is crucial in terms of code protection and debug. Indeed there is a possibility that the instruction immediately following the RFM be submitted to inadequate protection rules (as defined by the old PSW.PRS field).

- Problem sequence:
- instr (monitor)
- instr (monitor)
- instr (monitor)
- RFM (monitor)
- Instruction1   // Uses debug monitor’s PSW.PRS field as opposed to newly restored one.
- instruction2

Workaround

To fix this the user needs to do the following before exiting the monitor using RFM:

.

- > Retrieve the old value of PSW from location DCX+4

- > Do a MFCR and a MTCR to copy the old value of PSW.PRS into PSW without changing other PSW fields.
- > DSYNC
- > RFM

This sequence will guarantee that all instructions fetched subsequently to the RFM will be submitted to the new PSW.PRS field.

OCDS_TC.027 BAM breakpoints with associated halt action can potentially corrupt the PC.

BAM breakpoints can be programmed to trigger a halt action. When such a breakpoint is taken the CPU will go into HALT mode immediately after the instruction is executed. This mechanism is broken in the case of conditional jumps. When a BAM breakpoint with halt action is triggered on a conditional jump, the PC for the next instruction will potentially be corrupted before the CPU goes into HALT mode. On exiting HALT mode the CPU will see the corrupted value of the PC and hence resume code execution from an erroneous location. Reading the PC CSFR whilst in HALT mode will also yield a faulty value.

Workaround

In order to avoid PC corruption the user should avoid placing BAM breakpoints with HALT action on random code which could contain conditional jumps.The simplest thing to do is to avoid BAM breakpoints with HALT action altogether. A combination of BBM breakpoints and other types of breakpoint actions can be used to achieve the desired functionality.:

Workaround for single-stepping:

An ’intuitive’ way of implementing single-stepping mode is to place a halt-action BAM breakpoint on the address range from 0x00000000 to 0xFFFFFFFF. Every time the CPU is woken up via the CERBERUS it will execute the next instruction and go back to HALT mode. Unfortunately this will trigger the bug described by the current ERRATA.

The solution is to implement single-stepping using BBM breakpoints:

- 1) Create two debug trigger ranges:
- First range: 0x00000000 to current_instruction_pc (not included)

- Second range: current_instuction_pc (not included) to 0xFFFFFFFF
- 2) Associate the two debug ranges with BBM breakpoints.
- 3) Associate the BBM breakpoints with a HALT action.
- 4) Wake up the CPU via CERBERUS
- 5) CPU will execute the next instruction, update the PC and go to HALT mode.
- 6) Start again (go back to 1)

<u>OCDS_TC.028</u> Accesses to CSFR and GPR registers of running program can corrupt loop exits.

Overview:

A hardware problem has been identified whereby FPI accesses to the [0xF7E10000 : 0xF7E1FFFF] region will potentially corrupt the functionality of the Tricore LOOP instruction. This is particularly relevant because the Tricore CPU CSFR and GPR registers are mapped to that region. So any access to those registers by an external agent will potentially cause the LOOP instruction not to work. Note that this problem will not happen if the CPU was halted at the time of the FPI access.

Typical bug behaviour:

The loop instruction should exit (fall through) when its loop count operand is zero. The identified problem will typically cause the loop instruction to underflow: instead of exiting when its loop count operand is zero, the loop instruction will erroneously jump back to its target with a -1 (0xFFFFFFFF) loop counter value, and then continue to iterate possibly ad infinitum. Note that the offending FPI access will not cause the bug to happen immediately but only when the loop instruction finally tries to exit.

Influencing factors:

The following factors influence the likelihood of the bug happening:

1) The bug will not happen if the LOOP instruction and its predecessor are both entirely contained in the same aligned 8-byte word.

2) The bug is much less likely to happen if the CPU is running from program cache or program scratchpad.

3) The problem will be more visible on later compiler versions which make a more intensive use of the loop instruction.

Workaround:

The workaround consists in preventing all FPI agents from accessing the [0xF7E10000 : 0xF7E1FFFF] region when the CPU is not halted.

This means that the CPU CSFR and GPR registers can't be accessed on-the-fly whilst the CPU is running. This is particularly relevant for debug tool providers who may be polling those registers as the application is running. Note that accessing FPI addresses outside of the [0xF7E10000 : 0xF7E1FFFF] region will not cause the problem to happen.

An Application Note for tool partners, describing an alternative, more complex workaround for register access within the critical region by an external tool, is available from Infineon.

PCP_TC.023 JUMP sometimes takes an extra cycle

Following a taken JUMP, the main state machine may misleadingly take an additional cycle of pause. This occurs if the already prefetched next or second next instruction after the JUMP is one of the following instructions:

- LD.P
- ST.P
- DEBUG
- Any instruction with extension .PI

This does not cause any different program flow or incorrect result, it just adds an extra dead cycle.

Workaround

None.

PCP_TC.027 Longer delay when clearing `R7.IEN` before atomic PRAM instructions

User Manual states that, when clearing `R7.IEN`, a delay of one instruction before the mask becomes effective is needed. However, two instructions (for example, two NOPs) are required between the clearing instruction and an atomic PRAM instruction (MSET.PI/MCLR.PI/XCH.PI).

PCP_TC.032 Incorrect PCP behaviour following FPI timeouts (as a slave)

When PRAM is being accessed from the FPI bus and an FPI time-out occurs then this can lead to corruption or loss of the current and subsequent FPI accesses. In general an FPI time-out during an access to the PCP is unlikely since FPI time-out is usually programmed for a large number of FPI clock cycles and the only time that the FPI access cannot be immediately responded to by the PCP is during the execution of atomic PRAM instructions. FPI accesses are locked out for the entire duration of any sequence of back to back atomic PRAM instructions. The combination of a low FPI time-out setting and long sequences of atomic PRAM instructions could therefore result in FPI time-out.

Workaround

Keep the FPI time-out setting as high as possible and do not include long sequences of back to back atomic PRAM instructions. If N is the highest amount of back to back atomic PRAM instructions in any PCP channel program, FPI time-out should at least be 10 times N.

PCP_TC.034 Usage of R7 requires delays between operations

If the following instruction sequence is used:

```
    <INSTR>  writing to R7
```

directly followed by

```
    <INSTR>  reading from R7
```

then the second instruction will fail, providing wrong data. The write will be anyway successful.

Workaround

Add a NOP between a write to R7 followed by a read from R7.

PCP_TC.035 Atomic PRAM operation right after COPY/BCOPY

The COPY/BCOPY instructions have an outer loop defined by R6.CNT1 (Transfer Count), enabled if CNC = 10_B.

If the instruction which follows the COPY/BCOPY is an atomic PRAM operation and the PCP channel is interrupted while in the COPY/BCOPY outer loop, the COPY/BCOPY operation will not be completed successfully.

Workaround

Do not allow atomic PRAM operations directly following COPY/BCOPY. In case both operations are necessarily consecutive, insert a NOP in between.

PCP_TC.036 Unexpected behaviour after failed posted FPI write

If PCP posts an FPI write (including those in atomic FPI operations) which produces an FPI error, following malfunctions may be observed:

- The PCP may lock the FPI bus
- Improper PCP instruction execution may occur
- PRAM content corruption
- A subsequent COPY operation could write incorrect data
- A subsequent invalid BCOPY (i.e. a BCOPY instruction which will also generate an FPI error) could cause the next channel to perform an unexpected error exit
- A byte or half-word COPY in the next channel may write incorrect data

Workaround

In a product intent system, FPI errors are extremely unlikely and no workaround is required. For debugging purposes, it may be useful to prevent these issues by not allowing any pending FPI write `(PCP_FTD.FPWC` = 10_B), although this action may impact PCP performance. Register `PCP_FTD`[1] address is F004 $3F30_H$, field `FPWC` is bits [6:5].

Note: FPI errors are extremely unlikely and, in any case, are an indication of system malfunction. In such situation, the recommended procedure is to restart the system.

PCP_TC.038 PCP atomic PRAM operations may operate incorrectly

PCP atomic PRAM instructions (XCH.PI, MSET.PI, MCLR.PI) may operate incorrectly when the read part coincides with the write part of an FPI read-modify-write operation to PRAM.

Workaround 1

If atomicity is required for the application, replace all atomic PRAM instructions with FPI RMW instructions (XCH.F, SET.F, CLR.F).

If atomicity is not required, either:

- ensure that no FPI master (including PCP itself) issues an FPI RMW operation on PRAM, or
- replace all MSET.PI, MCLR.PI and XCH.PI with their non-atomic equivalents.

Equivalent non-atomic instructions:

- `MSET.PI`

```
    OR.PI    Rx, offset1
    ; Rx now contains result that MSET.PI
    ; would have generated but PRAM is unchanged
    ST.PI    Rx, offset1
    ; Rx written to PRAM so MSET.PI
    ; functionality (non-atomic) is achieved
```

- `MCLR.PI`

1) Register PCP_FTD is not documented in the Target Specification/User's Manual. Its symbolic name may therefore not be supported by all versions of tools (compiler, debugger, etc.).

```
        AND.PI    Ry, offset2
        ; Ry now contains result that MCLR.PI
        ; would have generated but PRAM is unchanged
        ST.PI     Ry, offset2
        ; Ry written to PRAM so MCLR.PI
        ; functionality (non-atomic) is achieved
```

- XCH.PI

```
        MOV       Ry, Rx, cc_UC
        ; Ry used a temporary holding register
        LD.PI     Rx, offset3
        ; Rx now contains PRAM value but PRAM is unchanged
        ST.PI     Ry, offset3
        ; Ry written to PRAM so XCH.PI
        ; functionality (non-atomic) is achieved
```

Workaround 2

Place a dummy FPI read in front of every PCP atomic PRAM instruction, i.e.

- Replace MSET.PI with:

```
        CLR R7 0x5 (prevent nested interrupt)
        NOP
        LD.F R4, [R0], size=32
                (dummy load, addr setup required)
        MSET.PI
```

- Replace MCLR.PI with:

```
        CLR R7 0x5 (prevent nested interrupt)
        NOP
        LD.F R4, [R0], size=32
                (dummy load, addr setup required)
        MCLR.PI
```

- Replace XCH.PI with:

```
        CLR R7 0x5 (prevent nested interrupt)
        NOP
        LD.F R4, [R0], size=32
                (dummy load, addr setup required)
        XCH.PI
```

PCP_TC.039 PCP posted error interrupt to CPU may be lost when the queue is full in 2:1 mode

In the unlikely case where ..

- PCP 2:1 mode is enabled,
- PCP is configured to post error interrupts to CPU,
- a channel is running,
- this channel's R7.CEN is cleared,
- PCP exits this channel with posting an interrupt to the CPU,
- as a result of the posted interrupt, CPU queue becomes full,
- and the same channel is invoked again immediately with context restore optimization,

the current channel should exit with posting an error interrupt to CPU, but actually the error interrupt to CPU is lost.

Workaround

Application software should not clear R7.CEN if there is a chance that the channel is going to be executed again.

RESET_TC.001 SCU_RSTSTAT.PORST not set by a combined Debug / System / Application Reset

Causing simultaneously a System, Application, and Debug Reset via CBS_OSTATE.RSTCL0…3 in most cases does not leave the SCU_RSTSTAT.PORST bit set as specified. Bit SCU_RSTSTAT.PORST stays set only if reset source ESR0 was configured not to generate any reset (SCU_RSTCON.ESR0 = 00_B). If the ESR0 reset source is configured to generate a reset, bit SCU_RSTSTAT.PORST is cleared and bit SCU_RSTSTAT.ESR0 is set instead.

Bits CBS_OSTATE.RSTCL0…3 are either set by setting bits CBS_OCNTRL.OJC4…7 or CBS_OJCONF.OJC4…7.

Debugging of "PORST-only" application software under debugger control is therefore not working if the ESR0 reset source generates a reset.

Workaround

Before triggering a simultaneous System, Application, and Debug Reset by the OCDS system bit field SCU_RSTSTAT.ESR0 should be cleared. In addition it should be checked that bit field SCU_RSTSTAT.ESR1 is also cleared.

If bit field SCU_RSTSTAT.ESR0 needs to contain a value different from 00_B instead of checking bit SCU_RSTSTAT.PORST the three bits SCU_RSTSTATCB0, SCU_RSTSTATCB1, and SCU_RSTSTATCB3 should be checked to be set.

SCU_TC.016 Reset Value of Registers ESRCFG0/1

The reset value of register `SCU_ESRCFG0` is 0x00000100 (instead of 0x00000110).

The reset value of register `SCU_ESRCFG1` is 0x00000080 (instead of 0x00000090).

This means that bit `DFEN` = 0_B, i.e. the digital 3-stage median filter is disabled after a System Reset.

Note: The 3-stage median filter operates on the FPI-Bus frequency. All input spikes lasting less than one FPI-Bus cycle are reliably suppressed. Any request lasting at least 2 FPI-Bus cycles is reliably recognized.

Workaround

In case the digital 3-stage median filter shall be enabled, bit `DFEN` must be set to 1_B by software.

SSC_AI.022 Phase error detection switched off too early at the end of a transmission

The phase error detection will be switched off too early at the end of a transmission. If the phase error occurs at the last bit to be transmitted, the phase error is lost.

Workaround

Don't use the phase error detection.

SSC_AI.023 Clock phase control causes failing data transmission in slave mode

If `SSC_CON.PH` = 1 and no leading delay is issued by the master, the data output of the slave will be corrupted. The reason is that the chip select of the master enables the data output of the slave. As long as the chip is inactive the slave data output is also inactive.

Workaround

A leading delay should be used by the master.

A second possibility would be to initialize the first bit to be sent to the same value as the content of `PISEL.STIP`.

SSC_AI.024 SLSO output gets stuck if a reconfig from slave to master mode happens

The slave select output SLSO gets stuck if the SSC will be re-configured from slave to master mode. The SLSO will not be deactivated and therefore not correct for the 1st transmission in master mode. After this 1st transmission the chip select will be deactivated and working correctly for the following transmissions.

Workaround

Ignore the 1st data transmission of the SSC when changed from slave to master mode.

SSC_AI.025 First shift clock period will be one PLL clock too short because not syncronized to baudrate

The first shift clock signal duration of the master is one PLL clock cycle shorter than it should be after a new transmit request happens at the end of the previous transmission. In this case the previous transmission had a trailing delay and an inactive delay.

Workaround

Use at least one leading delay in order to avoid this problem.

SSC_AI.026 Master with highest baud rate set generates erroneous phase error

If the SSC is in master mode, the highest baud rate is initialized and `CON.PO` = 1 and `CON.PH` = 0 there will be a phase error on the MRST line already on the shift edge and not on the latching edge of the shift clock.

- Phase error already at shift edge
  The master runs with baud rate zero. The internal clock is derived from the rising and the falling edge. If the baud rate is different from zero there is a gap between these pulses of these internal generated clocks.
  However, if the baud rate is zero there is no gap which causes that the edge detection is to slow for the "fast" changing input signal. This means that the input data is already in the first delay stage of the phase detection when the delayed shift clock reaches the condition for a phase error check. Therefore the phase error signal appears.
- Phase error pulse at the end of transmission
  The reason for this is the combination of point 1 and the fact that the end of the transmission is reached. Thus the bit counter `SSCBC` reaches zero and the phase error detection will be switched off.

Workaround

Don’t use a phase error in master mode if the baud rate register is programmed to zero (`SSCBR` = 0) which means that only the fractional divider is used.

Or program the baud rate register to a value different from zero (`SSCBR` > 0) when the phase error should be used in master mode.

3 Deviations from Electrical- and Timing Specification

DTS_TC.P001 Test Conditions for Sensor Accuracy T_{TSA}

Parameter “Sensor Accuracy” (symbol T_{TSA}) is not subject to production test, it is verified by design / characterization.

The corresponding note will be added in the next revisions of the Data Sheet.

ESD_TC.P002 ESD violation

In the data sheet the ESD strength based on human body model is specified as:

Secure Voltage Range V_{HBM} = 0 V - 2000 V.

The real secure ESD Voltage ranges of the part have been characterized to be:

Secure Voltage Range V_{HBM} = 0 - 1750 V

Care has to be taken that these voltage limits are not exceeded during handling of the parts.

FADC_TC.P003 Incorrect test condition specified in datasheet for FADC parameter “Input leakage current at V_{FAGND}”.

In datasheet the test condition for FADC parameter “Input leakage current at V_{FAGND}” is specified as: $0V < V_{IN} < V_{DDMF}$.

The actual test condition is: $V_{IN} = 0V$.

It is not allowed to raise V_{FAGND} above 1.5V when the FADC is in power down mode, in order not to damage the device.

MSC_TC.P001 Incorrect V_{OS} limits for LVDS pads specified in Data Sheet

Table 11 Parameters as per Data Sheet

Parameter	Symbol	Min Value	Max Value	Unit	Note
Output offset voltage	V_{OS}	1075	1325	mV	

Table 12 Actual Parameters

Parameter	Symbol	Min Value	Max Value	Unit	Note
Output offset voltage	V_{OS}	1060	1340	mV	

New limits (starting with date codes of week 39/2010) are valid for whole temperature and VDD range.

Change in V_{OS} limits will not cause any impact to the LVDS communication, because the remaining 3 specified parameters (V_{OH}, V_{OL} and V_{OD}) for the LVDS communication are not affected.

PLL_TC.P005 PLL Parameters for f_{VCO} > 780 MHz

When the PLL is configured for VCO frequencies f_{VCO} > 780 MHz, the specified PLL parameters may be exceeded.

Workaround

Select the values for the P-, N- and K2-dividers such that the desired target frequency f_{PLL} is achieved with f_{VCO} ? 780 MHz.

4 Application Hints

ADC_AI.H002 Minimizing Power Consumption of an ADC Module

For a given number of A/D conversions during a defined period of time, the total energy (power over time) required by the ADC analog part during these conversions via supply V_{DDM} is approximately proportional to the converter active time.

Recommendation for Minimum Power Consumption:

In order to minimize the contribution of A/D conversions to the total power consumption, it is recommended

1. to select the internal operating frequency of the analog part (f_{ADCI} or f_{ANA}, respectively)[1)] near the **maximum** value specified in the Data Sheet, and
2. to switch the ADC to a power saving state (via `ANON`) while no conversions are performed. Note that a certain wake-up time is required before the next set of conversions when the power saving state is left.

Note: The selected internal operating frequency of the analog part that determines the conversion time will also influence the sample time t_S. The sample time t_S can individually be adapted for the analog input channels via bit field STC.

CPU_TC.H004 PCXI Handling Differences in TriCore1.3.1

The TriCore1.3.1 core implements the improved architecture definition detailed in the TriCore Architecture Manual V1.3.8. This architecture manual version continues the process of removing ambiguities in the description of context save and restore operations, a process started in Architecture Manual V1.3.6 (released October 2005).

1) Symbol used depends on product family: e.g. f_{ANA} is used in the documentation of devices of the AUDO-NextGeneration family.

Several previous inconsistencies regarding the updating of the PCXI and the storing of PCXI fields in the first word of a CSA are now removed.

- CALL has always placed the full PCXI into the CSA
- BISR has always placed the full PCXI into the CSA
- SVLCX has always placed the full PCXI into the CSA
- RET has always restored the full PCXI from the CSA
- RFE has always restored the full PCXI from the CSA

From the TriCore V1.3.8 architecture manuals onwards it is also made explicit that:

- CALL, BISR and SVLCX now explicitly update the PCXI.PCPN, PCXI.PIE, PCXI.UL, PCXI.PCXS and PCXI.PCXO fields after storing the previous PCXI contents to memory.
- RSLCX now restores the full PCXI from the CSA.

However, prior to the TriCore V1.3.6 architecture manual, and as implemented by the TriCore1.3 core, the following behaviour was present:

- BISR and SVLCX previously only updated the PCXI.UL, PCXI.PCXS and PCXI.PCXO fields after storing the previous PCXI contents to memory. PCXI.PCPN and PCXI.PIE were not updated.
- RSLCX previously restored only the PCXI.UL, PCXI.PCXS and PCXI.PCXO fields of the PCXI.

The main implication of this change is that the value held in the PCXI.PCPN and PCXI.PIE fields following a BISR, SVLCX or RSLCX instruction may be different between the TriCore1.3.1 and TriCore1.3 cores. If it is necessary to determine the priority number of an interrupted task after performing a BISR or SVLCX instruction, and before the corresponding RSLCX instruction, then either of the following access methods may be used.

Method #1

For applications where the time prior to execution of the BISR instruction is not critical, the priority number of the interrupted task may be read from the PCXI before execution of the BISR instruction.

```
...
mfcr    d15, #0xFE00
bisr    #<New Priority Number>
```

```
...
```

Method #2

For applications where the time prior to execution of the BISR instruction is critical, the priority number of the interrupted task may be read from the CSA pointed to by the `PCXI` after execution of the BISR instruction.

```
...
bisr    #<New Priority Number>
mfcr    d15, #0xFE00             ; Copy PCXI to d15
sh.h    d14, d15, #12            ; Extract PCX seg to d14
insert  d15, d14, d15, #6, #16 ; Merge PCX offset to d15
mov.a   a15, d15                 ; Copy to address reg
ld.bu   d15, [a15]0x3           ; Load byte containing PCPN
...
```

Note that contrary to the TriCore architecture specification, no DSYNC instruction is stricly necessary after the BISR (or SVLCX) instruction, in either the TriCore1.3 or TriCore1.3.1, to ensure the previous CSA contents are flushed to memory. In both TriCore1.3 and TriCore1.3.1, any lower context save operation (BISR or SVLCX) will automatically flush any cached upper context to memory before the lower context is saved.

CPU_TC.H005 Wake-up from Idle/Sleep Mode

A typical use case for idle or sleep mode is that software puts the CPU into one of these modes each time it has to wait for an interrupt.

Idle or Sleep Mode is requested by writing to the Power Management Control and Status Register (`PMCSR`). However, when the write access to `PMCSR` is delayed e.g. by a higher priority bus access, TriCore may enter idle or sleep mode while the interrupt which should wake up the CPU is already executed. As long as no additional interrupts are triggered, the CPU will endlessly stay in idle/sleep mode.

Therefore, e.g. the following software sequence is recommended (for user mode 1, supervisor mode):

```
_disable();             // disable interrupts
```

```
do {
SCU_PMCSR = 0x1;        // request idle mode
if( SCU_PMCSR );        // ensure PMCSR is written

_enable();              // after wake-up: enable interrupts
_nop();
_nop();                 // ensure interrupts are enabled
_disable();             // after service: disable interrupts
} while( !condition ); // return to idle mode depending on
                        // condition set by interrupt handler
_enable();
```

FIRM_TC.H000 Reading the Flash Microcode Version

The 1-byte Flash microcode version number is stored at the bit locations 103-96 of the LDRAM address D000 $000C_H$ after each reset, and subject to be overwritten by user data at any time.

The version number is defined as “Vsn”, contained in the byte as:

- **s** = highest 4 bit, hex number
- **n** = lowest 4 bit, hex number

Example: V21, V23, V3A, V3F, etc.

FPI_TC.H001 FPI bus may be monopolized despite starvation protection

During a sequence of back to back 64-bit writes performed by the CPU to PCP memories (PRAM/CMEM) the LFI will lock the FPI bus and no other FPI master (PCP, DMA, OCDS) will get a grant, regardless of the priority, until the sequence is completed.

A potential situation would be a routine which writes into the complete PRAM and CMEM to initialize the parity bits (for devices with parity) or ECC bits (for devices with ECC), respectively. If the write accesses are tightly concatenated, the FPI bus may be monopolized during this time. Such situation will not be detected by the starvation protection.

Workaround

Avoid 64-bit CPU to PCP PRAM/CRAM accesses.

HYS_TC.H001 Effective Hysteresis in Application Environment

Pad hysteresis values are specified for a noise-free environment. The methodology of measuring the hysteresis on product level comprises noise due to clock signals, program execution and device activity, etc. This can lead to a measurable hysteresis that is smaller than it really is. Therefore hysteresis should be checked again in the real target application, at system level.

The measured hysteresis in a noise-free environment is within the specified product limits.

MCDS_TC.H001 TriCore trigger sources core_cu* and core_clr* toggle with some taken branches

The instruction pointer comparators of the Memory Protection Unit primarily work on the code fetch address. Instructions “fetched” by the LP pipeline from its internal buffer no longer have their original address, therefore the comparison results seen by the TQU_TC as triggers **core_cu*** and **core_clr*** are invalid during branches taken by the LP pipeline.

If these triggers are used for program trace enabling, unexpected IPE/IPA pairs will result.

If these triggers are used to qualify other traces some messages may go missing.

Workaround

Use the core triggers only for linear code stretches or single instructions.

For code stretches containing branches use the comparators of the PTU_TC.

MCDS_TC.H002 Missing EOT message at end of trace buffer

The MCDS message format is constructed with a “streaming” connection in mind: Without any knowledge of physical memory organization the “nibble stream” (sequence of 4 bit wide chunks of data) can be decomposed into standard messages. Only the interpretation of the payload of such messages requires deeper knowledge of the device under test.

The end of the stream is marked by a special message (EOT, 0xFFF).

Unfortunately there are scenarios where the EOT message is not written into the on-chip buffer memory:

- Tracing was never started, e.g. because MCDS was not enabled or locked.
- The memory block was switched from MCDS to tool access (BBB mode, see TILES register in the EMEM specification) too early.
- Due to a malfunction the memory location containing EOT was overwritten by MCDS. This has been observed in rare cases when the post-trigger area is very short (less than three memory lines[1)]) or tracing stops very near a paragraph boundary.

From a message decoder’s point of view the root cause is irrelevant. The problem is simply to render the last memory line “invalid” if the nibble stream either stops unexpectedly or no longer adheres to the specified format.

Workaround

A “single shot” decoder, which works on the complete trace memory after tracing has stopped, shall use the algorithm given in the MCDS specification to find the start of the trace and then follow the thread until either an EOT message is found or the **NOW** pointer is reached. In the latter case do not use the content of the line pointed to by **NOW**.

An “on the fly” decoder, which processes the trace data fetched from the memory while the trace is ongoing (“continuous trace”) shall keep track of the paragraph boundaries. These are easily found by numbering the received nibbles starting from the first nibble after each SKIP message. If the trace ends

1) A memory line is the amount of data MCDS is able to write to the on chip memory per clock cycle. See EMEM description for the product specific value.

without EOT message the last (counter modulo line-size) nibbles should be discarded.

MCDS_TC.H003 Trace buffer content after abort

Usually a single-shot measurement using MCDS terminates automatically when the pre-configured amount of trace data has been recorded. A dedicated definition (see MCDS_MCXACT29) is used to specify the “trigger”-point in the measurement. In “post-trigger-only” mode only events concurrent with and after this point are recorded.

On the other hand, the MCDS_FIFOCTL.FLSH bit allows to manually terminate an ongoing measurement at any time.

No matter how the measurement is terminated, the logical end is marked by an ’end-of-trace’-message in the memory word pointed to by MCDS_FIFONOW.

If however the manual termination is done in “post-trigger-only” mode before said trigger has happened, the complete trace buffer is logically empty, i.e. the content is invalid. It may happen that the ’end-of-trace’-message is not at the beginning of the memory word.

Workaround

When terminating the measurement manually via MCDS_FIFOCTL.FLSH in “post-trigger-only” mode, consult the MCDS_FIFOCTL.TRG flag afterwards. If it is not set discard the complete trace buffer content.

Note: This usually is done anyway to minimize the amount of uploaded data.

MCDS_TC.H004 Decoding the data size of DTU messages

Compression of trace information is of paramount importance for the MCDS design. A dedicated “data size” field was therefore avoided for DTU messages. On the other hand, the decoder needs to know how big the data object has been prior to compression to properly decode the message stream. The compromise was to consider the size implicitly given by any uncompressed message’s length as constant for all subsequent compressed messages, i.e. until the next uncompressed message “sets” a new size.

This algorithm works properly for uncompressed DTW, DTWD and DTR messages, as the message-specific length information can be used directly. For DTRD messages however there is a problem: The length includes the subtype field, which happens to be two bits wide. As the symbolic length encoding only supports “nibble” values it is not possible to make an uncompressed DTRD message with 8 data bits: The smallest possible symbolic length for 8 data bits + 2 subtype bits turns out to be 12.

The obvious solution is to consider 10 bit wide data fields as BYTE as well. Unfortunately the MCDS module used by AUDO-NextGeneration and AUDO-Future emulation devices will transmit a HALFWORD with a value of ’0’ in the most significant 6 bit positions also as a 10 bit wide data field. And if bit positions 9 and 10 happen to be ’0’ too, the decoder has no way to tell BYTE from HALFWORD.

The same problem applies for 31/32 bit wide WORDs: They must be transmitted in messages using the symbolic length 44, making them indistinguishable from DOUBLEs with bits 33…64 equal to ’0’.

For HALFWORDs the matter is even worse, as the next available symbolic length is 32, making 15/16 bit wide HALFWORDs look the same as WORDs with bits 17…32 equal to ’0’.

Workaround

First of all, the problem is not a severe one: The only place where DTRD messages are mandatory at all is in case of split DTR messages - which happens for DOUBLE sized data mainly.

In all other cases

- either a DTR can be used, which does not have the described problem
- or the size is known by the tool anyway. Remember, a DTRD does not contain any address information, so to “understand” the message the address also must be known by the tool software.

Note: If the address is left away (to save trace bandwidth) because it is constant, then the penalty of using a DTR instead of DTRD will be very small as the compression will shrink the address field to almost nothing.

MSC_TC.H007 Start Condition for Upstream Channel

The reception of the upstream frame is started when a falling edge (1-to-0 transition) is detected on the SDI line.

In addition, reception is also started when a low level is detected on the SDI line while the upstream channel was in idle state, i.e.

- when the upstream channel is switched on (bit field `URR` in register `USR` is set to a value different from 000_B) and the SDI line is already on a low level, or
- after a frame has been received, and the SDI line is on a low level at the end of the last stop bit time slot (e.g. when the SDI line is permanently held low).

Therefore, make sure that the SDI line is pulled high (e.g. with an internal or external pull-up) while no transmission is performed.

MSC_TC.H008 The LVDS pads require a settling time when coming up from pad power-down state.

The LVDS pad power-down state is the default state for the LVDS pad at:

- power-up
- all resets (including software reset)
- as soon as the LVDS is disabled (by PDR register).

The settling time until reaching normal operating electrical levels is defined as the duration between programming of the relevant PDR register to enable LVDS and the time where the LVDS pads reach the specified operating levels in the datasheets.

This settling time:

- increases with decreasing temperature (first order)
- decreases with increasing voltage supply, V_{DDP} (second order)
- is not dependent on the external capacitive load on the LVDS pads
- is not dependent on the system frequency

The settling time is shown in the Table 1 below:

Table 13 **LVDS pads settling time**

Conditions	Typical	Maximum
+25°C, V_{DDP} = 3.3v	15µs	60µs
-40°C, V_{DDP} = 3.3v	470µs	1.3ms
-40°C, V_{DDP} = 3.14v	520µs	1.4ms

Note: LVDS settling time for higher temperatures remains within the limits defined above by +25°C.

Workaround

In case of reset, or ,if switching off then on the LVDS pad by software, the PDR register should be programmed soonest possible for the LVDS pads to reach its stable level.

User has to take care that the communication is not started by the application software within the settling time period after enabling of LVDS in PDR register.

MultiCAN_AI.H005 TxD Pulse upon short disable request

If a CAN disable request is set and then canceled in a very short time (one bit time or less) then a dominant transmit pulse may be generated by MultiCAN module, even if the CAN bus is in the idle state.

Example for setup of the CAN disable request:

`CAN_CLC.DISR` = 1 and then `CAN_CLC.DISR` = 0

Workaround

Set all INIT bits to 1 before requesting module disable.

MultiCAN_AI.H006 Time stamp influenced by resynchronization

The time stamp measurement feature is not based on an absolute time measurement, but on actual CAN bit times which are subject to the CAN resynchronization during CAN bus operation.The time stamp value merely indicates the number of elapsed actual bit times. Those actual bit times can be shorter or longer than nominal bit time length due to the CAN resynchronization events.

Workaround

None.

MultiCAN_TC.H002 Double Synchronization of receive input

The MultiCAN module has a double synchronization stage on the CAN receive inputs. This double synchronization delays the receive data by 2 module clock cycles. If the MultiCAN is operating at a low module clock frequency and high CAN baudrate, this delay may become significant and has to be taken into account when calculating the overall physical delay on the CAN bus (transceiver delay etc.).

MultiCAN_TC.H003 Message may be discarded before transmission in STT mode

If `MOFCRn.STT`=1 (Single Transmit Trial enabled), bit TXRQ is cleared (TXRQ=0) as soon as the message object has been selected for transmission and, in case of error, no retransmission takes places.

Therefore, if the error occurs between the selection for transmission and the real start of frame transmission, the message is actually never sent.

Workaround

In case the transmission shall be guaranteed, it is not suitable to use the STT mode. In this case, `MOFCRn.STT` shall be 0.

MultiCAN_TC.H004 Double remote request

Assume the following scenario: A first remote frame (dedicated to a message object) has been received. It performs a transmit setup (`TXRQ` is set) with clearing NEWDAT. MultiCAN starts to send the receiver message object (data frame), but loses arbitration against a second remote request received by the same message object as the first one (NEWDAT will be set).

When the appropriate message object (data frame) triggered by the first remote frame wins the arbitration, it will be sent out and NEWDAT is not reset. This leads to an additional data frame, that will be sent by this message object (clearing NEWDAT).

There will, however, not be more data frames than there are corresponding remote requests.

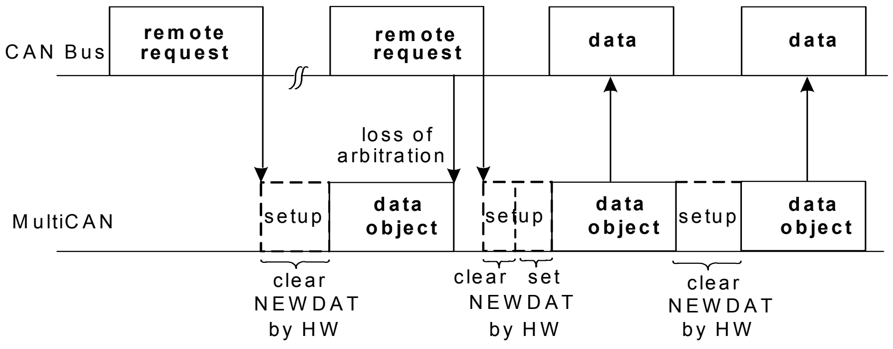

Figure 2 Loss of Arbitration

OCDS_TC.H001 IOADDR may increment after aborted IO_READ_BLOCK

If an IO_READ_BLOCK instruction is aborted by the host (switching the TAP controller to the update-DR state before enough data bits have been shifted out) it may happen under certain clock ratios that the IOADDR register is incremented nevertheless. This will result in an access to the wrong data in the succeeding IO_READ_* or IO_WRITE_* instruction.

Workaround

As the host is actively causing the abort, it should be fully aware of the situation. The workaround now simply is to rewrite the IOADDR register (using the IO_SET_ADDRESS instruction) after each aborted block transfer.

Note: This usually is done anyway at the beginning of the next transaction.

OCDS_TC.H002 Setting IOSR.CRSYNC during Application Reset

If the host is shifting in a Communication Mode IO_READ_WORD instruction in the very moment an Application Reset happens, the read request flag (CBS_IOSR.CRSYNC) may be already set after the execution of the startup software. A monitor program may be confused by this and drop out of the higher level communication protocol, especially if the host posts an instruction (with the IO_WRITE_WORD instruction) after detecting the reset.

Workaround

Two correlated activities should be incorporated in the tool software:

- After each reset the host should explicitly use CBS_IOCONF.COM_RST to reset any erroneously pending requests.
- The higher level protocol should require a specific answer to the very first command sent from the host to the device. Erroneous read requests then can be detected and skipped.

OCDS_TC.H003 Application Reset during host communication

Not only the host is able to cause resets of the device: External pins driven by the application, the internal watchdog and even the application program itself can trigger the reset generation process.

The only way to communicate reset events to the host is for Cerberus to reject the next instruction with “never-ending busy”, which should lead to communication time out on the host side.

The decision to accept or reject an instruction is done very early in the bit stream of the instruction. If an Application Reset happens after this point of time, the instruction will complete in most cases, and only the next one will be rejected.

As the temporal distance from reset event and instruction rejection is not fixed (apart from being sequential), it is highly recommended to check the IOINFO register (using the IO_SUPERVISOR instruction) each time an abnormally long busy period is experienced by the host. Especially a repetition of the rejected instruction should only be attempted if the possibility of Cerberus being in Error State has been excluded.

Workaround

Use IO_SUPERVISOR whenever a (too) long busy bit is observed.

OCDS_TC.H004 Device Identification by Application Software

While each device type can easily be recognized by test equipment using the JTAG ID, over the years each device family has had a proprietary way to provide the same information to application software running on the device. When reusing software for another device family the algorithm had to be adapted.

To worsen things, using the wrong algorithm may cause fatal errors, e.g. traps when accessing illegal addresses.

Starting with the Audo Future family the JTAG ID as available as a standard SFR (CBS_JTAGID) at a fixed address, namely in the address space of the “main” Cerberus. The value found in this register unambiguously defines where additional information (e.g. CHIPID) can be found in the device on hand.

Older devices obviously do not have the CBS_JTAGID, so accessing its address may cause problems.

Workaround

Each Cerberus module ever implemented has a version number mapped into a register (CBS_JDPID) at a fixed address (0xF0000408).

Note: This register is not published in all user manual versions.

- If the version number found in this register (CBS_JDPID[7:0]) is less than 0x50, no CBS_JTAGID register is provided. The original software algorithm shall be employed by the reused software.

- If the version number found in this register (CBS_JDPID[7:0]) is 0x50 or higher, the content of the CBS_JTAGID register shall be used to select the proper algorithm.

PCP_TC.H004 Invalid parity error generated by FPI write to PRAM

If an FPI write is performed to a PRAM location that contains a parity error then the PCP generates a memory error when it should not.

Workaround

Ensure that there are no PRAM locations with an incorrect parity bit by initialising every PRAM location (with parity checking enabled) prior to enabling trapping of PRAM parity errors.

PCP_TC.H005 Unexpected parity errors when address 0 of CMEM is faulty

When CMEM parity error detection is enabled and its content at address 0 is faulty, only an access to the mentioned address should raise the parity error flag. However, in this specific situation a FPI read access to any CMEM location will result in a parity error.

PCP_TC.H006 BCOPY address alignment error may affect next channel FPI operation

When a BCOPY is executed starting on a non-aligned address (e.g. address 0x...4 or 0x...C for double-word burst, BTR2), the channel will perform an error exit. If the first FPI instruction of the next channel is a byte or half-word FPI RMW or FPI write, the data written by this instruction may be corrupted due to the previous error.

PCP_TC.H007 Do not use priority 0 to post interrupt to CPU

Posting interrupts with priority 0 to CPU bus is not permitted, because the CPU will never acknowledge the interrupt and thus the service request node will never be cleared. If this is repeated sufficient times to fill the CPU queue, then the PCP would stall and the system would have to be reset to restore PCP operation. As well, care has to be taken when using a lower amount of arbitration rounds in the ICU, because a high priority value can be seen as priority 0. Example: 11000000_B with 3 arbitration rounds will be seen as 00000000_B.

PORTS_TC.H004 Using LVDS Ports in CMOS Mode

The following constraint applies to an LVDS pair used in CMOS mode:

Only one pin of a pair shall be used as output, the other shall be used as input. Using both pins as outputs or inputs simultaneously is not allowed because of the cross-coupling between them.

PORTS_TC.H005 Pad Input Registers do not capture Boundary-Scan data when BSD-mode signal is set to high

The principle of Boundary-Scan is that the BSD-cells can overrule the input and output data for all functional system components (including port-input registers).

In current implementation the peripheral port input registers(P<n>_IN) are however capturing the direct pad-input data even when the BSD-mode signal is set to high.

This limits the usage of INTEST.

Work around:

In case of INTEST, do not read port input registers.

PWR_TC.H005 Current Peak on V_{DDP} during Power-up

During power-up, a current peak may be observed on V_{DDP}. It is caused by internal cross currents generated by level shifters whose state is undefined until the core voltage reaches at least 0.5V. This effect is statistical and may vary from one device to the other, upon operating conditions, etc. This effect may only occur during power-up. It can not happen during power-down or power-fail.

The following table classifies the V_{DD}/V_{DDP} ranges with respect to peak severity.

Table 14 Worst Case Power-up Cross Current

V_{DD}	V_{DDP}	Comment
> 0.5 V	don't care	normal operation
< 0.5 V	< 0.8 V	I_{DDP} < 2 mA
< 0.5 V	0.8 V < V_{DDP} < 1.0 V	I_{DDP} < 5 mA
< 0.5 V	= 3.6 V	I_{DDP} < 270 mA

Even under worst case conditions, this effect has no impact on lifetime nor reliability

SSC_AI.H001 Transmit Buffer Update in Slave Mode after Transmission

If the Transmit Buffer register `TB` is written in slave mode in a time window of one SCLK cycle after the last SCLK edge (i.e. after the last data bit) of a transmission, the first bit to be transmitted may not appear correctly on line MRST.

Note: This effect only occurs if a configuration with PH = 1_B (shift data on trailing edge) is selected.

It is therefore recommended to update the Transmit Buffer in slave mode after the transmit interrupt (TIR) has been generated (after first SCLK phase of first bit), and before the current transmission is completed (before last SCLK phase of last bit).

As this may be difficult to achieve in systems with high baud rates and long interrupt latencies, alternatively the receive interrupt at the end of a transmission may be used. A delay of 1.5 SCLK cycles (bit times) after the receive interrupt (last SCLK edge of transmission) should be provided before updating the Transmit Buffer of the slave. The master must provide a pause that is sufficient to allow updating of the slave Transmit Buffer before starting the next transmission.

SSC_AI.H002 Transmit Buffer Update in Master Mode during Trailing or Inactive Delay Phase

When the Transmit Buffer register TB is written in master mode after a previous transmission has been completed, the start of the next transmission (generation of SCLK pulses) may be delayed in the worst case by up to 6 SCLK cycles (bit times) under the following conditions:

- a trailing delay (SSOTC.TRAIL) > 0 and/or an inactive delay (SSOTC.INACT) > 0 is configured
- the Transmit Buffer is written in the last module clock cycle (f_{SSC} or f_{CLC}) of the inactive delay phase (if INACT > 0), or of the trailing delay phase (if INACT = 0).

No extended leading delay will occur when both TRAIL = 0 and INACT = 0.

This behaviour has no functional impact on data transmission, neither on master nor slave side, only the data throughput (determined by the master) may be slightly reduced.

To avoid the extended leading delay, it is recommended to update the Transmit Buffer after the transmit interrupt has been generated (i.e. after the first SCLK phase), and before the end of the trailing or inactive delay, respectively.

Alternatively, bit BSY may be polled, and the Transmit Buffer may be written after a waiting time corresponding to 1 SCLK cycle after BSY has returned to 0_B.

After reset, the Transmit Buffer may be written at any time.

SSC_AI.H003 Transmit Buffer Update in Slave Mode during Transmission

After reset, data written to the Transmit Buffer register TB are directly copied into the shift register. Further data written to TB are stored in the Transmit Buffer while the shift register is not yet empty, i.e. transmission has not yet started or is in progress.

If the Transmit Buffer is written in slave mode during the first phase of the shift clock SCLK supplied by the master, the contents of the shift register are overwritten with the data written to `TB`, and the first bit currently transmitted on line MRST may be corrupted. No Transmit Error is detected in this case.

It is therefore recommended to update the Transmit Buffer in slave mode after the transmit interrupt (TIR) has been generated (i.e. after the first SCLK phase).

After reset, the Transmit Buffer may be written at any time.

SSC_TC.H003 Handling of Flag **`STAT.BSY`** in Master Mode

In register `STAT` of the High-Speed Synchronous Serial Interface (SSC), some flags have been made available that reflect module status information (e.g. error, busy) closely coupled to internal state transitions. In particular, flag `STAT.BSY` will change twice during data transmission: from 0_B to 1_B at the start, and from 1_B to 0_B at the end of a transmission. This requires some special considerations e.g. when polling for the end of a transmission:

In master mode, when register `TB` has been written while no transfer was in progress, flag `STAT.BSY` is set to 1_B after a constant delay of 5 FPI bus clock cycles. When software is polling `STAT.BSY` after `TB` was written, and it finds that `STAT.BSY` = 0_B, this may have two different meanings: either the transfer has not yet started, or it is already completed.

Recommendations

In order to poll for the end of an SSC transfer, the following alternative methods may be used:

- either test flag `RSRC.SRR` (receive interrupt request flag) instead of `STAT.BSY`
- or use a software semaphore that is set when `TB` is written, and which is cleared e.g. in the SSC receive interrupt service routine.