SPARC Radiation Tolerant Processor Chip Set (CBA) Design Considerations List

This document will often be released. Please refer to it regularly.

1. Introduction

1.1 Scope

This document describes the current identified specification deviations (as off June 15, 98) for the TSC691E (rev C), TSC692E (rev B) and TSC693E (rev A), with a work around proposal, when available.

1.2 Applicable documents

TSC691E specification, rev I, MHS September 1998

TSC692E specification, rev H, MHS December 1996

TSC693E specification, rev D, MHS April 1997

2. Integer Unit TSC691E

No deviation identified.

3. Floating-point unit TSC692E

3.1 Store Double floating-point instruction

3.1.1 Problem description

It occurs in the following sequence:

LD [...], %fd1 or LDD [...], %fd1 or FPop [...], %fd1

0 (for LD and LDD) or up to 80 instructions (for FPop)

STD %fd2, [...]

If there is no dependency in the whole sequence and the first instruction writes its result into the register file upon the STDF Write stage, then the STDF instruction may store the wrong data.

3.1.2 Work Around

Do not use STDF instructions, use two STF instead.

3.1.3 Example of failing code

```
!------------------------------------------------------------------------------------------------------------
! initialization
!------------------------------------------------------------------------------------------------------------
init:

set     data1,%l7
set     0x000000100,%r2

!------------------------------------------------------------------------------------------------------------
! example 1
! Work around:
!       replace:    std         %f0, [%l7]
!       by:         st          %f0, [%l7]
!                   st          %f1 [%l7+4]
!------------------------------------------------------------------------------------------------------------
example_1:

ldd     [%l7+16], %f0           ! %f0,f1=0x4040000040800000
nop
nop
nop
ldd     [%l7+24], %f2           ! %f2,f3=0x7fd5555555554000
std     %f0, [%l7]              ! %f0,f1=0x4040000040800000
nop
nop
nop

!------------------------------------------------------------------------------------------------------------
! example 2
! Work around:
!       replace:    std         %f4, [%l7+8]
!       by:         st          %f4, [%l7+8]
!                   st          %f5, [%l7+12]
!------------------------------------------------------------------------------------------------------------
example_2:

ldd     [%l7+32], %f4           ! %f4,f5=0xb58637bd11000007
nop
nop
nop
fadds   %f0, %f0, %f3
nop
nop
nop
nop
std     %f4, [%l7+8]            ! %f4,f5=0xb58637bd11000007
add     %r2, %r2, %r2           ! required
nop
nop

end:
```

```
!------------------------------------------------------------------------------------------------------------
! data segment
!------------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0xfffffff                       ! address=[%l7], store destination example 1
.word   0xffffffff
.word   0xffffffff                      ! address=[%l7+8], store destination example 2
.word   0xffffffff
.word   0x40400000                      ! address=[%l7+16], %f0 example 1 & 2
.word   0x40800000
.word   0x7fd55555                      ! address=[%l7+24], %f2 example 1
.word   0x55554000
.word   0xb58637bd                      ! address=[%l7+32], %f4 example 2
.word   0x11000007
```

3.2 Store Floating-point Status Register with waitstates

3.2.1 Problem description

It occurs in the following sequence with waitstates:

LD %fsr

ST %fsr

The Store floating-point Status Register instruction will store the previous value of FSR and not the updated one.

3.2.2 Work Around

Insert a NOP between these two instructions:

LD %fsr

NOP

ST %fsr

3.2.3 Example of failing code

```
!------------------------------------------------------------------------------------------------------------
! initialization
!------------------------------------------------------------------------------------------------------------
init:

set     data1,%l0
ld      [%l0],%fsr                      ! %fsr=0x0f080000
```

```
!---------------------------------------------------------------------------------------------------------------
! example (with wait states on instruction fetch)
! Work around:
!       between the following two instructions:
!                       ld              [%l0],%fsr
!                       st              %fsr,[%l0+4]
!       insert a nop:
!                       ld              [%l0],%fsr
!                       nop
!                       st              %fsr,[%l0+4]
!---------------------------------------------------------------------------------------------------------------
example:

nop
nop
nop


ld      [%l0+4],%fsr                    ! %fsr=0xcf080000
st      %fsr,[%l0+8]                    ! Check writing

nop
nop
nop

end:

!---------------------------------------------------------------------------------------------------------------
! segment "data"
!---------------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x0f080000                      ! address=[%l0], %fsr value_1
.word   0xcf080000                      ! address=[%l0+4], %fsr value_2
.word   0xffffffff                      ! address=[%l0+8], %fsr checking value
```

3.3 Load and Load Double Floating-point instructions with waitstates

3.3.1 Problem description

It occurs in the following sequence with waitstates on DATA loaded:

LD %fd1 or LDD %fd1

FPop %fd1 or ST %fd1 or STD %fd1

If a Floating-point Load instruction is immediately followed by a Floating-point operation or Floating-point Store instruction which has an operand conflict with the Load instruction, a hardware interlock (FHOLD cycle) is generated with the MHOLD cycle of the DATA loaded (waitstate assertion). In this case the Fpop or STF or STDF instruction is executed with the previous data of the %fd1 register.

3.3.2 Work Around

Insert a NOP between these two instructions:

LD %fd1 or LDD %fd1

NOP

FPop %fd1 or ST %fd1 or STD %fd1

3.3.3 Example of failing code

```
!-----------------------------------------------------------------------------------------------------------
! initialization
!-----------------------------------------------------------------------------------------------------------
init:

set     data1,%l0

ld      [%l0],%fsr                  ! %fsr=0x0f080000
ld      [%l0+4],%f10                ! %f10=0x00000000

!-----------------------------------------------------------------------------------------------------------
! example (with wait states on instruction fetch)
! Work around:
!       between the following two instructions:
!               ld              [%l0+12],%f10
!               st              %f10,%f8,%f10
!       insert a nop:
!               ld              [%l0+12],%f10
!               nop
!               st              %f10,%f8,%f10
!-----------------------------------------------------------------------------------------------------------
example:

ld      [%l0+8],%f8                 ! %f8=0x3febab55

nop
nop
nop

ld      [%l0+12],%f10               ! %f10=0x40000000
fsubs   %f10,%f8,%f10               ! %f10=0x3e22a558

nop
nop
nop

st      %f10,[%l0+16]               ! %f10=0x3e22a558

nop
nop
nop

end:
```

```
!------------------------------------------------------------------------------------------------------------
! segment "data"
!------------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x0f080000                      ! address=[%l0], %fsr value
.word   0x00000000                      ! address=[%l0+4], %f10 init value
.word   0x3febab55                      ! address=[%l0+8], %f8 = %rs2
.word   0x40000000                      ! address=[%l0+12], %f10 = %rs1
.word   0xffffffff                      ! address=[%l0+16], %f10 = %rd (fsubs result)
```

3.4 Store and Store Double Floating-point instructions with waitstates

3.4.1 Problem description

It occurs in the following sequence with waitstates:

FPop [...], %fd1

up to 80 instructions

ST %fd2, [...] or STD %fd2, [...]

If a Floating-point operation is followed after some cycles by a Floating-point Store or Store Double instruction without operand conflicts and any kind of waitstate holds the STF or STDF in its E stage as the FPop writes its result back into the register file, then the STF or STDF instruction may store the wrong data.

3.4.2 Work Around

Do not use waitstates (do not assert any hold signal) when using the TSC692E.

3.4.3 Example of failing code

```
!------------------------------------------------------------------------------------------------------------
! initialization
!------------------------------------------------------------------------------------------------------------
init:

set     data1,%l7

!------------------------------------------------------------------------------------------------------------
! example (with wait states)
! Work around for this example:
!       between the following two intructions:
!                       fadds           %f0,%f0,%f2
!                       nop
!       insert a nop:
!                       fadds           %f0,%f0,%f2
!                       nop
!                       nop
!------------------------------------------------------------------------------------------------------------
```

example:

```
ldd    [%l7], %f0              ! %f0,f1=0x4040000040800000
ldd    [%l7+8], %f2            ! %f2,f3=0x7fd5555555554000
ldd    [%l7+16], %f4           ! %f4,f5=0xb58637bd11000007
nop
nop
nop
fadds  %f0, %f0, %f2
nop
nop
ld     [%g0], %g0              ! required
st     %f4, [%l7+24]           ! %f4=0xb58637bd
nop
nop
nop

end:

!----------------------------------------------------------------------------------------------------------
! segment "data"
!----------------------------------------------------------------------------------------------------------
.align 8

data1:
.word  0x40400000              ! address=[%l7], %f0,f1 init value
.word  0x40800000
.word  0x7fd55555              ! address=[%l7+8], %f2,f3 init value
.word  0x55554000
.word  0xb58637bd              ! address=[%l7+16], %f4,f5 init value
.word  0x11000007
.word  0xffffffff              ! address=[%l7+24], %f4 checking value
.word  0xffffffff
```

3.5 Store Double Floating-point Queue instruction with waitstates

3.5.1 Problem description

It occurs in the following instruction with waitstates on instruction fetch:

STD %fq, [...]

Always store the wrong instruction.

3.5.2 Work Around

Never use STDFQ instruction with waitstates on instruction fetch.

3.5.3 Example of failing code

```
!----------------------------------------------------------------------------------------------------------
! initialization
!----------------------------------------------------------------------------------------------------------
```

```
init:

set     data1,%l7

!-----------------------------------------------------------------------------------------------------------------
! example (with wait states on instruction fetch)
!-----------------------------------------------------------------------------------------------------------------
example:

ldd     [%l7], %f0                       ! %f0,f1=0x4040000040800000
nop
nop
nop
fadds   %f0, %f0, %f2
nop
nop
nop
std     %fq, [%l7+8]                     ! %fq=(example+16) / 85a00820
nop
nop
nop

end:

!-----------------------------------------------------------------------------------------------------------------
! segment "data"
!-----------------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x40400000                       ! address=[%l7], %f0,f1 init value
.word   0x40800000
.word   0xffffffff                       ! address=[%l7+8], %fq checking value
.word   0xffffffff
```

3.6 Store Double Floating-point Queue instruction with disabled parity checking

3.6.1 Problem description

It occurs in the following instruction with $\overline{\text{602MODE}}$ = 0:

STD %fq, [...]

Generation of internal parity is not implemented for the Floating-point Queue instruction data path. Then, in case of wrong parity stored in the Floating-point Queue, a STDFQ leads to a Non-Restartable Hardware Error Exception, instead of bringing back to Execute mode!

3.6.2 Work Around

Use only RT mode ($\overline{\text{602MODE}}$ = 1).

3.6.3 Example of failing code

```
!----------------------------------------------------------------------------------------------------------
! initialization
!----------------------------------------------------------------------------------------------------------
init:

set      data1,%l7

!----------------------------------------------------------------------------------------------------------
! example
!----------------------------------------------------------------------------------------------------------
example:

ldd      [%l7], %f0                   ! %f0,f1=0x4040000040800000
nop
nop
nop
fadds    %f0, %f0, %f2
nop
nop
nop
std      %fq, [%l7+8]                 ! %fq=(example+16) / 0x85a00820
nop
nop
nop

end:

!----------------------------------------------------------------------------------------------------------
! segment "data"
!----------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x40400000                   ! address=[%l7], %f0,f1 init value
.word   0x40800000
.word   0xffffffff                   ! address=[%l7+8], %fq checking value
.word   0xffffffff
```

3.7 Floating-point instruction sequence with one waitstate

3.7.1 Problem description

It may occur in the following sequence of any TSC692E instructions with waitstates:

FPinst0

FPinst1

FPinst2

If a Floating-point instruction FPinst1 generates (required by Fpinst0) a hardware interlock (one FHOLD cycle) and is followed by any other Floating-point instruction FPinst2 fetched with one waitstate, Fpinst2 fetch will fail.

3.7.2 Work Around

Insert a NOP between two Floating-point instructions FPinst1 and FPinst2:

FPinst0

FPinst1

NOP

FPinst2

3.7.3 Example of failing code

```
!------------------------------------------------------------------------------------------------------------
! initialization
!------------------------------------------------------------------------------------------------------------
init:

set     data1,%l7

!------------------------------------------------------------------------------------------------------------
! example (with wait states on instruction fetch)
! Work around:
!       between the following two intructions:
!               fmovs   %f0,%f0
!               ldd     [%l7+8],%f4
!       insert a nop:
!               fmovs   %f0,%f0
!               nop
!               ldd     [%l7+8],%f4
!------------------------------------------------------------------------------------------------------------
example:

ldd     [%l7], %f0                      ! %f0,f1=0x4000000040400000
ldd     [%l7], %f4                      ! %f4,f5=0x4000000040400000
ldd     [%l7], %f6                      ! %f6,f7=0x4000000040400000
nop
nop
nop

fdivd   %f0,%f6,%f0                     ! %f0= 0x3ff0000000000000 (=1)
fmovs   %f0,%f0                         ! => hardware interlock generated
ldd     [%l7+8],%f4                     ! FAIL: writting %f0 instead of %f4

nop
nop
nop

std     %f4,[%l7+16]                    ! %f4,f5=0x7fd5555555554000 expected
std     %f0,[%l7+24]                    ! %f0,f1=0x3ff0000000000000 expected
```

```
nop
nop
nop

end:

!-----------------------------------------------------------------------------------------------------------
! segment "data"
!-----------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x40400000              ! address=[%l7], %f0,f1, %f4,f5 and %f6,f7 init value
.word   0x40800000
.word   0x7fd55555              ! address=[%l7+8], %f4,f5 wanted loading value
.word   0x55554000
.word   0xffffffff              ! address=[%l7+16], %f4,f5 checking value
.word   0xffffffff
.word   0xffffffff              ! address=[%l7+24], %f0,f1 checking value
.word   0xffffffff
```

3.8 Floating-point Compare - Floating-point instruction sequence with waitstates

3.8.1 Problem description

It occurs in the following sequence of instructions with waitstates:

FCMP

FPinst

If a Floating-point compare instruction FCMP is immediately followed by any other Floating-point instruction FPinst with hardware interlock between both, and if a waitstate holds the FCMP in its E stage, and if the FCMP generates an exception, then the IU will trap on the FPinst and return from trap will be done at the wrong address. For instance, if the FPinst should have been re-executed after return, it is the FCMP that will be! One of the risks is then an unwanted infinite loop if the condition triggering the exception is still true.

3.8.2 Work Around

Insert a NOP after a FCMP instruction:

FCMP

NOP

3.8.3 Example of failing code

```
!-----------------------------------------------------------------------------------------------------------
! trap
!-----------------------------------------------------------------------------------------------------------
FP_exception:                   ! FPinst re-executed after return
```

```
nop
jmpl    %l1, %g0
rett    %l2
nop

!------------------------------------------------------------------------------------------------------------
! initialization
!------------------------------------------------------------------------------------------------------------
init:

set     data1,%l7
ld      [%l1],%fsr                      ! %fsr=0x0f880000


!------------------------------------------------------------------------------------------------------------
! example (with wait states on instruction fetch)
! Work around:
!       between the following two intructions:
!                       fcmpes          %f0,%f2
!                       fadds           %f0,%f0,%f20
!       insert a nop:
!                       fcmpes          %f0,%f2
!                       nop
!                       fadds           %f0,%f0,%f20
!------------------------------------------------------------------------------------------------------------
example:

ldd     [%l7+8], %f0                    ! %f0,f1=0x0fffffffffffffff
ldd     [%l7+16], %f2                   ! %f2,f3=0x7fd5555555554000
nop
nop
nop
nop

fcmpes %f0, %f2                         ! fexc pending  (return address if wait states)
fadds   %f0,%f0,%f20                    ! trap taken (return address if ok)

nop
nop
nop

end:

!------------------------------------------------------------------------------------------------------------
! segment "data"
!------------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x0f880000                      ! address=[%l7+8], %fsr init value
.word   0xffffffff
.word   0xffffffff                      ! address=[%l7+8], %f0,f1 init value
.word   0xffffffff
.word 0x7fd55555                        ! address=[%l7+16], %f2,f3 init value
.word 0x55554000
```

3.9 FNULL signals assertion with waitstates

3.9.1 Problem description

FNULL may be wrongly asserted after or during a $\overline{\text{MHOLD}}$, $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$ or CCCV cycle. The IU frozen too early by $\overline{\text{FHOLD}}$ may not output the next address and the Memory Controller may indefinitely restart the same bus cycle. This may prevent usage of the TSC692E with other Memory Controller than the MEC.

FNULL is not a MEC input.

3.9.2 Work Around

Do not use the TSC692E without the MEC.

3.10 $\overline{\text{FHOLD}}$ signal dead-lock with coprocessor

3.10.1 Problem description

$\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$ and CCCV signals wrongly prevent $\overline{\text{FHOLD}}$ deassertion.

This may lead to dead-lock when a coprocessor is used. For instance, a sequence as

CCMP

FPop

may show CCCV and $\overline{\text{FHOLD}}$ asserted in the same cycle. $\overline{\text{FHOLD}}$ is wrongly locked by CCCV which in turn is locked by the CCMP being frozen by $\overline{\text{FHOLD}}$ in its Write stage.

3.10.2 Work Around

Do not use coprocessor with the TSC692E.

3.11 FCCV signal dead-lock with waitstates and coprocessor

3.11.1 Problem description

FCCV may be wrongly deasserted during a $\overline{\text{MHOLD}}$, $\overline{\text{BHOLD}}$, $\overline{\text{CHOLD}}$ or CCCV cycle.

This may lead to dead-lock when a coprocessor is used. For instance, a sequence as

CCMP

FCMP

with waitstates during the Write stage of the CCMP shows the CCMP locked in this Write stage after deassertion of the hold signal because FCCV has been wrongly deasserted. FCMP is then locked in its E stage by CCCV.

3.11.2 Work Around

Do not use coprocessor with waitstates and the TSC692E.

3.12 Data output tristate upon FLUSH rising edge

3.12.1 Problem description

Unlike the TSC691E, the TSC692E Data output are disabled asynchronously on FLUSH rising edge during any Floating Point Store instruction if a TSC691E trap has occured (external interrupts, internal error ...). This may lead to the MEC detecting a parity error on the early tristated Data Bus and asserting $\overline{\text{MEXC}}$, then setting the TSC691E into Error Mode!

The right behaviour for Data Bus tristate with FLUSH is described in TSC692E User's Manual in section "3.2.2. Instruction Pipeline Flush".

3.12.2 Work Around

Do not use MEC parity checking with the TSC692E.

WARNING : The TSC692E never detects parity errors on data loaded!

3.13 Clock edge Data output enabling and disabling

3.13.1 Problem description

Data output are enabled and disabled on CLK rising edges instead of falling edges.

3.13.2 Work Around

No work around known.

3.14 Floating Point Operations FLUSH abortion

3.14.1 Problem description

Floating Point Operations may not be aborted by FLUSH signal during Execute or Write Stage. They may keep on running inside the TSC692E core, potentially leading to malfunction.

If an FPop1 should have been aborted in its Execute or Write Stage and is not, then when an FPop2 is issued, a wrong Unimplemented FPop Exception may occur.

A typical case is encountered when trying to restart a FPop (that should have been aborted and is not) after a trap routine.

3.14.2 Work Around

Trap routines should not use any FPop.

Analyse the FPop flagged as Unimplemented using "std %fq" which returns opcode and address values of the faulty instruction. If the opcode belongs to the Fpop instruction set, try to restart it.

3.14.3 Example of handling code

```
! This routine handles the unimplemented FPop IT relevant of bug 3.14
! In case of an umiplemented FPop trap, we get the opcode of the faulting FPop
! and compare it to the list of implemented FPop.
! If matched, we have encountered bug 3.14 : just execute again the FPop ;
! Otherwise, the FPop is really umimplemented.

! fpuit must be installed in the trap table, tt 0x08

! list of implemented FPop opcodes
! Extended Fpop should not appear in this list because they are
! NOT implemented in the TSC692
! Note: FBcc is an IU intruction

.text
_fpop_mask:
        .word   0xc1f83fe0
_fpop_opcode:
        .word   0x81a00120      ! FABSs
        .word   0x81a00840      ! FADDd
        .word   0x81a00820      ! FADDs
        .word   0x81a80a40      ! FCMPd
        .word   0x81a80ca0      ! FCMPEd
        .word   0x81a80aa0      ! FCMPEs
        .word   0x81a80a20      ! FCMPs
        .word   0x81a009c0      ! FDIVd
        .word   0x81a009a0      ! FDIVs
        .word   0x81a01a40      ! FdTOi
        .word   0x81a018c0      ! FdTOs
        .word   0x81a01900      ! FiTOd
        .word   0x81a01880      ! FiTOs
        .word   0x81a00020      ! FMOVs
        .word   0x81a00940      ! FMULd
        .word   0x81a00920      ! FMULs
        .word   0x81a000a0      ! FNEGs
        .word   0x81a00540      ! FSQRTd
        .word   0x81a00520      ! FSQRTs
        .word   0x81a01920      ! FsTOd
        .word   0x81a01a20      ! FsTOi
        .word   0x81a008c0      ! FSUBd
        .word   0x81a008a0      ! FSUBs
        .word   0x00000000      ! End of list

_fpqueue:
        .word   0x0
        .word   0x0
```

```
.global _fpuit

_fpuit:

        ! Get the FTT field in the %fsr
        set     _fpqueue,%l4    ! we use fpqueue as temp location
        st      %fsr,[%l4]
        ld      [%l4],%l4
        srl     %l4,14,%l4
        and     %l4,0x7,%l4     ! %l4 contains FTT

        subcc   %l4,0x3,%g0     ! Unimp FPop ?
        bne     not_unimp
        nop

        ! Get the FPop opcode from the fpqueue

        set     _fpqueue,%l5
        std     %fq,[%l5]
        nop
        ld      [%l5+4],%l5     ! %l5 contains the FPop opcode
        set     _fpop_opcode,%l3
        set     _fpop_mask,%l4
        ld      [%l4],%l4
        and     %l4,%l5,%l5
        ld      [%l3],%l6
loop:
        subcc   %l6,%l5,%g0
        be      found
        nop
        add     %l3,4,%l3
        ld      [%l3],%l6
        subcc   %l6,%g0,%g0     ! last item in the list?
        bne     loop
        nop

        ba      unimp

found:  ! The faulting Opcode is implemented :
        ! we try and execute it again

        jmpl    %l1,%g0
        rett    %l2

unimp:
        ! Real Unimplemented Opcode
        ! Unimplemented FPop routine goes here ...

        jmpl    %l2,%g0
        rett    %l2+4

not_unimp:
        ! Here we enter the normal FPU trap handling
        ! ... code

! return from trap routine
        jmpl    %l1,%g0
        rett    %l2
```

3.15 FPU register addressing

3.15.1 Problem description

The problem occurs in the following sequence:

Fpop1 %rs1, %rs2, %rd
up to 80 IU instructions (depending on Fpop1 and data)
lddf [], %rd
Fpop2 %rs1, %rs2, %rd

with the following conditions:
condition 1: rs2 (Fpop2) = rd (Fpop1)
condition 2: rd(Fpop1) and rd(lddf) with bit[2] = bit[4] (example f0 and f2, f8 and f10, ...)

In this case, the Fpop1 instruction will store the wrong data in the register File due to the lddf Fp instruction.

3.15.2 Work Around

case 1:

Source:

```
Fpop1 %rs1, %rs2, %rd
IU instructions
lddf [], %rd
Fpop2 %rs1, %rs2, %rd
```

If rd(lddf) and rs2(Fpop2) with bit[2] = bit[4] (rd[4:0], rs1[4:0], rs2[4:0])

Patch:

```
Fpop1 %rs1, %rs2, %rd
IU instructions
ldf [], %rd
ldf [], %rd+1
Fpop2 %rs1, %rs2, %rd
```

case 2: Fpop1 = fmovs or fabss or fnegs

Source:

```
movs %rs2, %rd (or fabss %rs2, %rd or fnegs %rs2, %rd)
lddf [], %rd
Fpop2 %rs1, %rs2, %rd
```

If conditions 1 and 2 are fulfilled:

Patch_1:

```
fmovs %rs2, %rd (or fabss %rs2, %rd or fnegs %rs2, %rd)
ldf [], %rd
```

ldf [], %rd+1
Fpop2 %rs1, %rs2, %rd

or Patch2 (same number of cycles):
fmovs %rs2, %rd (or fabss %rs2, %rd or fnegs %rs2, %rd)
nop
lddf [], %rd
Fpop2 %rs1, %rs2, %rd

case 3: Fpop1 is NOT equal to fmovs or fabss or fnegs or fsubs
Source:
Fpop1 %rs1, %rs2, %rd (with Fpop1 is NOT equal to fabss or fnegs or fmovs or fsubs)
lddf [], %rd
Fpop2 %rs1, %rs2, %rd

Nothing to be patched

Note: the replacement of lddf by 2 ldf works for all cases.

3.15.3 Example of failing code

```
!-----------------------------------------------------------------------------------------------------------
! initialization
!-----------------------------------------------------------------------------------------------------------
init:

set     data1,%l7

ld      [%l7],%fsr                      ! %fsr=0x0f080000
ldd     [%l7+8],%f8                     ! %f8=0xffffffffffffffff
ldd     [%l7+8],%f10                    ! %f10=0xffffffffffffffff
nop

!-----------------------------------------------------------------------------------------------------------
! example
! General work around:
!       replace:        ldd             [%l7+24],%f10
!       by:             ld              [%l7+24],%f10
!                       ld              [%l7+28],%f11
! Work around for this example:
!       between the following two intructions:
!                       fnegs           %f8,%f8
!                       ldd             [%l7+24],%f10
!       insert a nop:
!                       fnegs           %f8,%f8
!                       nop
!                       ldd             [%l7+24],%f10
!-----------------------------------------------------------------------------------------------------------
example:
```

```
ldd     [%l7+16],%f8                 ! %f8=0x3febab5557101f8d
nop
nop
nop

fnegs   %f8,%f8                      ! %f8=0xbfebab5557101f8d
ldd     [%l7+24],%f10                ! %f10=0x4000000000000000 expected
fsubd   %f10,%f8,%f10                ! %f10=0x4006ead555c407e3 expected

nop
nop
nop

std     %f8,[%l7+32]                 ! %f8 checking value
nop
nop
nop
std     %f10,[%l7+40]                ! %f10 checking value
nop
nop
nop

end:

!-----------------------------------------------------------------------------------------------------------
! segment "data"
!-----------------------------------------------------------------------------------------------------------
.align 8

data1:
.word   0x0f080000                   ! address=[%l7], %fsr init value
.word   0xffffffff
.word   0xffffffff                   ! address=[%l7+8], %f8 and %f10 init value
.word   0xffffffff
.word   0x3febab55                   ! address=[%l7+16], %f8 loading value
.word   0x57101f8d
.word   0x40000000                   ! address=[%l7+24], %f10 loading value
.word   0x00000000
.word   0xffffffff                   ! address=[%l7+32], %f8 checking value
.word   0xffffffff
.word   0xffffffff                   ! address=[%l7+40], %f10 checking value (=0x0 if error)
.word   0xffffffff
```

4. Memory Control Unit TSC693E

4.1 TSC693E ERSR CPU halt indication in ERSR clearance at soft reset

4.1.1 Problem description

The 13:th bit (HLT) in the Error Reset and Status Register (ERSR) indicates if the IU/FPU are or have been halted. If this bit is set and a soft reset is triggered, a TSC693E internal parity error will be detected.

In case of any of the following resets:

1. Watch Dog reset

2. Software reset

3. Error reset

the reset cause is written to the ERSR and the parity is re-calculated. The TSC693E detects a parity error in this register and asserts TSC693E hardware Error. This parity error is only performed when the 13:th bit of the error and reset status register is set.

This means that if the IU/FPU were halted (by asserting the external halt signal and then resume axecution by deasserting the same signal), then the SW, WD and Error reset can’t be performed (in the future) due to parity error.

4.1.2 Workaround

No Workaround known.

4.2 UART status after UART clear

4.2.1 Problem description

Clearing the UARTs by setting the associated bits in the UART status register, will assign some default (reset) values to the Parity Enable, Even/Odd Parity and Stop Bits. These values are not the same values in the TSC693E Control Register and the irrespective of that register.

The UARTs enable the following when cleared:

1. Parity Enable

2. Odd parity

3. One Stop bit

It’s not possible to continue programme execution after this action, unless the incoming data has the same configuration.

4.2.2 Workaround

To work around the problem, re-programme the UART configuration bits in the TSC693E Control Register (bits 20:22) after each UART clear operation.

4.3 System Status Register update during Non-Correctable Error

4.3.1 Problem description

The System Fault Status Register doesn’t update the data fault type when an uncorrectable error is detected in the memory.

The memory exception handling is correct. This problem has only been observed during read operations:

1. Read byte
2. Read halfword
3. Read word
4. Read double word

The expected data fault type in the system fault register is 0x103C, but the register always shows 0x78 the reset value.

4.3.2 Workaround

No Workaround known.

4.4 Byte/halfword operations during Waitstates

4.4.1 Problem description

When programming the TSC693E Waitstate Configuration Register to RAM write = 0 WS and RAM read = 1,2,3 WS, Byte write operations will fail and Half word operations will fail.

This problem has been observed during IU operations:

1. stb (store byte)
2. sth (store half word)

This problem has been observed during byte/half word write in RAM on the DEM32 board and on the TSC693E VHDL model:

1. Write byte, 0 WS(write) and 1 WS(read)
2. Write byte, 0 WS(write) and 2 WS(read)
3. Write byte, 0 WS(write) and 3 WS(read)
4. Write hword, 0 WS(write) and 1 WS(read)
5. Write hword, 0 WS(write) and 2 WS(read)
6. Write hword, 0 WS(write) and 3 WS(read)

Where the TSC693E executes a read-modify-write operation.

Read 32-bit memory data, modify byte/half word, write 32-bit memory data.

The expected write strobe, MEMWR1*, is not generated by the TSC693E.

The expected write strobe, MEMWR2*, is generated correctly by the TSC693E.

Due to the missing write strobe data is not written.

EDAC check bits is written if MEMWR2* is used for check bits writing.

4.4.2 Workaround

No Workaround known.

4.5 Wrong DMA access error

4.5.1 Problem description

When a DMA access aborts an illegal store byte to a TSC693E register, the TSC693E register access violation leads to a DMA access error: IRL is set to 8.

stb %g0,[%g1 + 0xe0]

with [%g1 + 0xe0] addressing TSC693E UARTA register is aborted as shown on the following timing diagram.

The TSC693E did not record the context in which the register access violation occurred, that is an IU access, and propagated the error through the DMA access context, triggering a DMA access error.

4.5.2 Work Around

No Workaround known.