ANALOG DEVICES

Blackfin Embedded Processor

Silicon Anomaly List — ADSP-BF606/BF607/BF608/BF609

ABOUT ADSP-BF606/BF607/BF608/BF609 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF606/BF607/BF608/BF609 product(s) and the functionality specified in the ADSP-BF606/BF607/BF608/BF609 data sheet(s) and the Hardware Reference book(s).

SILICON REVISIONS

A silicon revision number with the form "-x.x" is branded on all parts. The REVID bits <31:28> of the SDU_IDCODE register can be used to differentiate the revisions as shown below.

Silicon REVISION	SDU_IDCODE.REVID
0.0	0x0
0.1	0x1

ANOMALY LIST REVISION HISTORY

The following revision history lists the anomaly list revisions and major changes for each anomaly list revision.

Date	Anomaly List Revision	Data Sheet Revision	Additions and Changes
11/08/2013	E	0	Added Anomalies: 16000041, 16000042
05/23/2013	D	PrG	Added Silicon Revision 0.1 Added Anomalies: 16000039, 16000040 Revised Anomaly: 16000023
01/15/2013	C	PrF	Added Anomalies: 16000023, 16000024, 16000025, 16000027, 16000028, 16000029, 16000033, 16000034. 16000035, 16000036, 16000037, 16000038 Revised Anomalies: 16000020, 16000021, 16000031
07/12/2012	B	PrD	Added Anomalies: 16000030, 16000031
03/06/2012	A	PrC	Initial Version

NR004212E

One Technology Way, P.O.Box 9106, Norwood, MA 02062-9106 U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.461.3113

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF606/BF607/BF608/BF609 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.0	0.1
1	16000003	TRU_STAT.ADDRERR and TRU_ERRADDR.ADDR May Not Reflect the Correct Status	x	x
2	16000004	The EPPI Data Enable (DEN) Signal is Not Functional	x	.
3	16000005	Using L1 Instruction Cache with Parity Enabled is Unreliable	x	.
4	16000006	SEQSTAT.SYSNMI Clears Upon Entering the NMI ISR	x	.
5	16000007	DDR2 Memory Reads May Fail Intermittently	x	x
6	16000008	Instruction Memory Stalls Can Cause IFLUSH to Fail	x	x
7	16000009	TESTSET Instruction Cannot Be Interrupted	x	x
8	16000010	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x
9	16000011	False Hardware Error when RETI Points to Invalid Memory	x	x
10	16000012	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
11	16000013	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
12	16000014	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
13	16000015	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x
14	16000017	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x
15	16000018	RSI Boot Cleanup Routine Does Not Clear Registers	x	.
16	16000019	SPI Master Boot Device Auto-detection Frequency is Set Incorrectly	x	.
17	16000020	Clock Buffer Disable Bits in DPM_RESTORE0 are not Functional	x	.
18	16000021	Rom_SysControl Save/Restore DMC Registers Feauture Not Functional	x	.
19	16000022	Boot Code Fails to Enable Parity Fault Detection	x	.
20	16000023	Rom_SysControl Does not Update CGU0_CLKOUTSEL	x	.
21	16000024	Spurious Fault Signaled After Clearing an Externally Generated Fault	x	x
22	16000025	SPORT May Drive Data Pins During Inactive Channels in Multichannel Mode	x	x
23	16000027	WUA_BCODE Bit in DPM_RESTORE0 Not Functional	x	.
24	16000028	Default SPI Master Boot Mode Setting is Incorrect	x	.
25	16000029	PPI tDFSPI Timing Does Not Meet Data Sheet Specification	x	.
26	16000030	Interrupted Core Reads of MMRs May Cause Data Loss	x	.
27	16000031	Incorrect Default USB_PLL_OSC.PLLM Value	x	.
28	16000032	Core Reads of System MMRs May Cause the Core to Hang	x	.
29	16000033	PPI Data Underflow on First Word Not Reported in Certain Modes	x	x
30	16000034	CNV1 Red Pixel Substitution feature not functional in the PVP	x	.
31	16000035	IPF0 Output Port Color Separation feature not functional	x	.
32	16000036	Spurious USB Wake From Hibernate May Occur When USB_VBUS is Low	x	.
33	16000037	Core RAISE 2 Instruction Not Latched When Executed at Priority Level 0, 1, or 2	x	.
34	16000038	Spurious Unhandled NMI or L1 Memory Parity Error Interrupt May Occur Upon Entering the NMI ISR	x	.
35	16000039	CGU_STAT.PLOCKERR Bit May be Unreliable	x	x
36	16000040	JTAG Emulator Reads of SDU_IDCODE Alter Register Contents	x	x
37	16000041	IFLUSH Instruction Causes Parity Error When Parity Is Enabled	x	x
38	16000042	Instruction Cache Failure When Parity Is Enabled	.	x

Key: x = anomaly exists in revision
. = Not applicable

DETAILED LIST OF SILICON ANOMALIES

The following list details all known silicon anomalies for the ADSP-BF606/BF607/BF608/BF609 including a description, workaround, and identification of applicable silicon revisions.

1. 16000003 - TRU_STAT.ADDRERR and TRU_ERRADDR.ADDR May Not Reflect the Correct Status:

DESCRIPTION:
TRU_STAT.ADDRERR and TRU_ERRADDR.ADDR show an error even if a valid address is written to the TRU address range.

WORKAROUND:
Do not depend on the values in TRU_STAT.ADDRERR and TRU_ERRADDR.ADDR.

APPLIES TO REVISION(S):
0.0, 0.1

2. 16000004 - The EPPI Data Enable (DEN) Signal is Not Functional:

DESCRIPTION:
The Data Enable (DEN) feature of the EPPI is not functional.

In GP 2 FS transmit mode with internally generated frame syncs, PPIx_FS3 does not function as a data enable (DEN) pin.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

3. 16000005 - Using L1 Instruction Cache with Parity Enabled is Unreliable:

DESCRIPTION:
Using L1 instruction cache with parity enabled is unreliable and may cause unpredictable results.

WORKAROUND:
If L1 instruction cache needs to be used, don't enable parity.

APPLIES TO REVISION(S):
0.0

4. 16000006 - SEQSTAT.SYSNMI Clears Upon Entering the NMI ISR:

DESCRIPTION:
SEQSTAT.SYSNMI indicates that a system NMI has occured. However, SEQSTAT.SYSNMI gets cleared as soon as the NMI ISR (Interrupt Service Routine) is entered.

WORKAROUND:
At the beginning of the NMI ISR, check all sources of the NMI in the SEQSTAT register. If none of the sources are active it can be assumed that the system NMI is the cause.

APPLIES TO REVISION(S):
0.0

5. 16000007 - DDR2 Memory Reads May Fail Intermittently:

DESCRIPTION:
DDR2 memory reads may fail intermittently using the DMC_TR2.TRTP (tRTP) minimum setting recommended in the Hardware Reference manual.

WORKAROUND:
Increase DMC_TR2.TRTP to a minimum of 5. If the DMC_TR2.TRTP calculation (using the memory manufacturer tRTP specification and the DCLK frequency) yields a result above 5, this higher DMC_TR2.TRTP value should be used.

APPLIES TO REVISION(S):
0.0, 0.1

6. 16000008 - Instruction Memory Stalls Can Cause IFLUSH to Fail:

DESCRIPTION:
When an instruction memory stall occurs when executing an IFLUSH instruction, the instruction may fail to invalidate a cache line. This could be a problem when replacing instructions in memory and could cause stale, incorrect instructions in cache to be executed rather than initiating a cache line fill.

WORKAROUND:
Instruction memory stalls must be avoided when executing an IFLUSH instruction. By placing the IFLUSH instruction in L1 memory, the prefetcher will not cause instruction cache misses that could cause memory stalls. In addition, padding the IFLUSH instruction with NOPs will ensure that subsequent IFLUSH instructions do not interfere with one another, and wrapping SSYNCs around it ensures that any fill/victim buffers are not busy. The recommended routine to perform an IFLUSH is:

```
SSYNC;          // Ensure all fill/victim buffers are not busy
LSETUP (LS, LE)
LS:  IFLUSH;
     NOP;
     NOP;
LE:  NOP;
SSYNC;          // Ensure all fill/victim buffers are not busy
```

Since this loop is four instructions long, the entire loop fits within one loop buffer, thereby turning off the prefetcher for the duration of the loop and guaranteeing that successive IFLUSH instructions do not interfere with each other.

APPLIES TO REVISION(S):
0.0, 0.1

7. 16000009 - TESTSET Instruction Cannot Be Interrupted:

DESCRIPTION:
When the TESTSET instruction gets interrupted, the write portion of the TESTSET may be stalled until after the interrupt is serviced. After the ISR completes, application code continues by reissuing the previously interrupted TESTSET instruction, but the pending write operation is completed prior to the new read of the TESTSET target data, which can lead to deadlock conditions.

For example, in a multi-threaded system that utilizes semaphores, thread A checks the availability of a semaphore using TESTSET. If this original TESTSET operation tested data with a low byte of zero (signifying that the semaphore is available), then the write portion of TESTSET sets the MSB of the low byte to 1 to lock the semaphore. When this anomaly occurs, the write doesn't happen until TESTSET is re-issued after the interrupt is serviced. Therefore, thread A writes the byte back out with the lock bit set and then immediately reads that value back, now erroneously indicating that the semaphore is locked. Provided the semaphore was actually still free when TESTSET was reissued, this means that the semaphore is now permanently locked because thread A thinks it was locked already, and any other threads that subsequently pend on the same semaphore are being locked out by thread A, which will now never release it. The same applies to a semaphore that is shared between multiple cores within the same device.

WORKAROUND:
The TESTSET instruction must be made uninterruptible to avoid this condition:

```
CLI R0;
TESTSET(P0);
STI R0;
```

There is no workaround other than this, so events that cannot be made uninterruptible, such as an NMI or an Emulation event, will always be sensitive to this issue. Additionally, due to the need to disable interrupts, User Mode code cannot implement this workaround.

APPLIES TO REVISION(S):
0.0, 0.1

8. 16000010 - IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall:

DESCRIPTION:
If the IFLUSH instruction is placed on a loop end, the processor will stall indefinitely. For example, the following two code examples will never exit the loop:

```
P1 = 2;
LSETUP (LOOP1_S, LOOP1_E) LC1 = P1;
LOOP1_S: NOP;
LOOP1_E: IFLUSH[P0++];

LSETUP (LOOP2_S, LOOP2_E) LC1 = P1;
LOOP2_S: NOP; NOP; NOP; NOP;       // Any number of instructions...
LOOP2_E: IFLUSH[P0++];
```

WORKAROUND:
Do not place the IFLUSH instruction at the bottom of a hardware loop. If the IFLUSH is padded with any instruction at the bottom of the loop, the problem is avoided:

```
LSETUP (LOOP_S, LOOP_E) LC1 = P1;
LOOP_S: IFLUSH[P0++];
LOOP_E: NOP;                    // Pad the loop end
```

APPLIES TO REVISION(S):
0.0, 0.1

9. 16000011 - False Hardware Error when RETI Points to Invalid Memory:

DESCRIPTION:

When using CALL/JUMP instructions targeting memory that does not exist, a hardware error condition will be triggered. If interrupts are enabled, the Hardware Interrupt (IRQ5) will fire. Since the RETI register will have an invalid location in it, it must be changed before executing the RTI instruction, even if servicing a different interrupt. Consider the following sequence:

```
P2.L = LO (0xFFAFFFFC);  // Load Address in Illegal Memory to P2
P2.H = HI (0xFFAFFFFC);
CALL(P2);                // Call to Bad Address Generates Hardware Error IRQ5
....

IRQ5_code:               // Hardware Error Interrupt Routine
RAISE 14;                // (1)
RTI;                     // (2)

IRQ14_code:
[--SP] = ( R7:0, P5:0 ); // (3)
[--SP] = RETI;           // (4)
....
```

When the hardware error occurs, the program counter points to the invalid location 0xFFAFFFFC, which is loaded into the RETI register during the service of the IRQ5 hardware error event. When the RTI instruction (2) is executed, a fetch of the instruction pointed to by the RETI register, which is an illegal address, is requested before hardware sees the level 14 interrupt pending. This fetch causes another hardware error to be latched, even though this instruction is not executed. Execution will go to IRQ14 (3). As soon as interrupts are re-enabled (4), the pending hardware error will fire.

WORKAROUND:

1. Ensure that code doesn't jump to or call bad pointers.
2. Always set the RETI register when returning from a hardware error to something that will not cause a hardware error on the memory fetch.

APPLIES TO REVISION(S):

0.0, 0.1

10. 16000012 - Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors:

DESCRIPTION:
A false hardware error is generated if there is an indirect jump or call through a pointer which may point to reserved or illegal memory on the opposite control flow of a conditional jump to the taken path. This commonly occurs when using function pointers, which can be invalid (e.g., set to -1). For example:

```
    CC = P2 == -0x1;
    IF CC JUMP skip;
    CALL (P2);
 skip:
    RTS;
```

Before the IF CC JUMP instruction can be committed, the pipeline speculatively issues the instruction fetch for the address at -1 (0xffffffff) and causes the false hardware error. It is a false hardware error because the offending instruction is never actually executed. This can occur if the pointer use occurs within two instructions of the conditional branch (predicted not taken), as follows:

```
    BRCC X [predicted not taken]
    Y: JUMP (P-reg);  // If either of these two p-regs describe non-existent
       CALL (P-reg);  // memory, such as external SDRAM when the SDRAM
    X: RTS;           // controller is off, then a hardware error will result.
```

WORKAROUND:
If instruction cache is on or the ICPLBs are enabled, this anomaly does not apply.

If instruction cache is off and ICPLBs are disabled, the indirect pointer instructions must be 2 instructions away from the branch instruction, which can be implemented using NOPs:

```
    BRCC X [predicted not taken]
    Y: NOP;          // These two NOPs will properly pad the indirect pointer
       NOP;          // used in the next line.
       JUMP (P-reg);
       CALL (P-reg);
    X: RTS;
```

APPLIES TO REVISION(S):
0.0, 0.1

11. 16000013 - False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory:

DESCRIPTION:
Due to fetches near boundaries of reserved memory, a false Hardware Error (External Memory Addressing Error) is generated under the following conditions:

1. A single valid CPLB spans the boundary of the reserved space. For example, a CPLB with a start address at the beginning of L1 instruction memory and a size of 4MB will include the boundary to reserved memory.

2. Two separate valid CPLBs are defined, one that covers up to the byte before the boundary and a second that starts at the boundary itself. For example, one CPLB is defined to cover the upper 1kB of L1 instruction memory before the boundary to reserved memory, and a second CPLB is defined to cover the reserved space itself.

As long as both sides of the boundary to reserved memory are covered by valid CPLBs, the false error is generated. Note that this anomaly also affects the boundary of the L1_code_cache region if instruction cache is enabled. In other words, the boundary to reserved memory, as described above, moves to the start of the cacheable region when instruction cache is turned on.

WORKAROUND:
Leave at least 76 bytes free before any boundary with a reserved memory space. This will prevent false hardware errors from occurring.

APPLIES TO REVISION(S):
0.0, 0.1

12. 16000014 - False Hardware Error from an Access in the Shadow of a Conditional Branch:

DESCRIPTION:

If a load accesses reserved or illegal memory on the opposite control flow of a conditional jump to the taken path, a false hardware error will occur.

The following sequences demonstrate how this can happen:

Sequence #1:
For the "predicted not taken" branch, the pipeline will load the instructions that sequentially follow the branch instruction that was predicted not taken. By the pipeline design, these instructions can be speculatively executed before they are aborted due to the branch misprediction. The anomaly occurs if any of the three instruction slots following the branch contain loads which might cause a hardware error:

```
BRCC X [predicted not taken]
R0 = [P0];     // If any of these three loads accesses non-existent
R1 = [P1];     // memory, such as external SDRAM when the SDRAM
R2 = [P2];     // controller is off, then a hardware error will result.
```

Sequence #2:
For the "predicted taken" branch, the one instruction slot at the destination of the branch cannot contain an access which might cause a hardware error:

```
BRCC X (BP)
Y: ...
   ...
X: R0 = [P0];  // If this instruction accesses non-existent memory,
               // such as external SDRAM when the SDRAM controller
               // is off, then a hardware error will result.
```

WORKAROUND:

If you are programming in assembly, it is necessary to avoid the conditions described above.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):

0.0, 0.1

13. 16000015 - Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported:

DESCRIPTION:

A multi-issue instruction with dsp32shiftimm in slot 1 and a P register store in slot 2 is not supported. It will cause an exception.

The following type of instruction is not supported because the P3 register is being stored in slot 2 with a dsp32shiftimm in slot 1:

```
R0 = R0 << 0x1 || [ P0 ] = P3 || NOP;      // Not Supported - Exception
```

This also applies to rotate instructions:

```
R0 = ROT R0 by 0x1 || [ P0 ] = P3 || NOP;  // Not Supported - Exception
```

Examples of supported instructions:

```
R0 = R0 << 0x1 || [ P0 ] = R1 || NOP;
R0 = R0 << 0x1 || R1 = [ P0 ] || NOP;
R0 = R0 << 0x1 || P3 = [ P0 ] || NOP;
R0 = ROT R0 by R0.L || [ P0 ] = P3 || NOP;
```

WORKAROUND:

In assembly programs, separate the multi-issue instruction into 2 separate instructions.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):

0.0, 0.1

14. 16000017 - Speculative Fetches Can Cause Undesired External FIFO Operations:

DESCRIPTION:
When an external FIFO device is connected to an asynchronous memory bank, memory accesses can be performed by the processor speculatively, causing improper operations because the FIFO will provide data to the Blackfin, and the data will be dropped whenever the fetch is made speculatively or if the speculative access is canceled. "Speculative" fetches are reads that are started and killed in the pipeline prior to completion. They are caused by either a change of flow (including an interrupt or exception) or when performing an access in the shadow of a branch. This behavior is described in the Blackfin Programmer's Reference.

Another case that can occur is when the access is performed as part of a hardware loop, where a change of flow occurs from an exception. Since exceptions can't be disabled, the following example shows how an exception can cause a speculative fetch, even with interrupts disabled:

```
CLI R3;                             /* Disable Interrupts */
LSETUP( loop_s, loop_e) LC0 = P2;
   loop_s: R0 = W[P0];              /* Read from a FIFO Device  */
   loop_e: W[P1++] = R0;            /* Write that Generates a Data CPLB Page Miss */
STI R3;                             /* Enable Interrupts */
RTS;
```

In this example, the read inside the hardware loop is made to a FIFO with interrupts disabled. When the write inside the loop generates a data CPLB exception, the read inside the loop will be done speculatively.

WORKAROUND:
First, if the access is being performed with a core read, turn off interrupts prior to doing the core read. The read phase of the pipeline must then be protected from seeing the read instruction before interrupts are turned off:

```
CLI R0;
NOP; NOP; NOP;  /* Can Be Any 3 Instructions */
R1 = [P0];
STI R0;
```

To protect against an exception causing the same undesired behavior, the read must be separated from the change of flow:

```
CLI R3;                             /* Disable Interrupts */
LSETUP( loop_s, loop_e) LC0 = P2;
   loop_s: NOP;                     /* 2 NOPs to Pad Read */
           NOP;
           R0 = W[P0];
   loop_e: W[P1++] = R0;
STI R3;                             /* Enable Interrupts */
RTS;
```

The loop could also be constructed to place the NOP padding at the end:

```
LSETUP( .Lword_loop_s, .Lword_loop_e) LC0 = P2;
   .Lword_loop_s: R0 = W[P0];
                  W[P1++] = R0;
                  NOP;          /* 2 NOPs to Pad Read */
   .Lword_loop_e: NOP;
```

Both of these sequences prevent the change of flow from allowing the read to execute speculatively. The 2 inserted NOPs provide enough separation in the pipeline to prevent a speculative access. These NOPs can be any two instructions.

Reads performed using a DMA transfer do not need to be protected from speculative accesses.

APPLIES TO REVISION(S):
0.0, 0.1

15. 16000018 - RSI Boot Cleanup Routine Does Not Clear Registers:

DESCRIPTION:
Upon completion of RSI boot, the RSI Cleanup routine does not correctly clear the following registers:

RSI0_CLK DMA10_CFG DMA10_XCNT DMA10_XMOD DMA10_ADDRSTART
The registers are instead initialized with the return result of the previous function call. The previous function call returns the contents of the RSI0_XFRSTAT register which (assuming a successful boot and reset operation) contains the value 0x00000080.

For a successful boot operation with no errors, the value written to these registers does not result in any errors. However if an RSI error occurred during the final stages of boot when the boot device is reset a different value may be written depending on the RSI failure as contained in the RSI0_XFRSTAT register.

WORKAROUND:
Users may disable the execution of the cleanup function if required by setting BITM_ROM_BFLAG_NORESTORE in the boot structure flags within initialization code if required.

APPLIES TO REVISION(S):
0.0

16. 16000019 - SPI Master Boot Device Auto-detection Frequency is Set Incorrectly:

DESCRIPTION:
The SPI master boot device auto-detection frequency is set higher than the default boot mode operation speed.

The default SPI master boot mode SPI clock divider setting is 0x1F. However, the device auto-detection routine uses a clock divider setting of 0x19.

The result is that the SPI master boot device must be able to support a minimum SPI operating frequency of SCLK1/26 instead of SCLK1/32.

WORKAROUND:
Use a SPI master boot device that supports a SPI operating frequency of at least SCLK1/26.

APPLIES TO REVISION(S):
0.0

17. 16000020 - Clock Buffer Disable Bits in DPM_RESTORE0 are not Functional:

DESCRIPTION:
The clock buffer disable bits in the DPM_RESTORE0 register (WUA_SCBF1DIS, WUA_SCBF2DIS, WUA_SCBF3DIS, and WUA_CCBF1DIS) are not functional.

WORKAROUND:
If clock gating coming out of hibernate is desired the code to do so must be present in the boot source.

APPLIES TO REVISION(S):
0.0

18. 16000021 - Rom_SysControl Save/Restore DMC Registers Feauture Not Functional:

DESCRIPTION:
The Rom_SysControl DMC save (DDR read) and DMC restore (DDR write) features are not functional. Consequently, if DDR2/LPDDR memory is placed into self-refresh during hibernate, code must be present in the boot source to re-initialize DDR2/LPDDR after wake from hibernate. Additionally, the DPM_RESTORE0.WUA_DDR bit should not be set.

WORKAROUND:
Place code in the boot source to re-initialize DDR2/LPDDR after wake from hibernate. If the contents of the DMC registers must be preserved during hibernate then (before entering hibernate) store the contents of the DMC registers in registers or memory that are preserved during hibernate.

APPLIES TO REVISION(S):
0.0

19. 16000022 - Boot Code Fails to Enable Parity Fault Detection:

DESCRIPTION:
The boot code fails to enable read parity checking in the IMEM_CONTROL and DMEM_CONTROL registers after the memory initialization procedure has completed. Consequently, parity errors are not detected and flagged as faults during the boot process. In addition, if cache is enabled after boot without first initializing the cache parity tags no parity error will result.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

20. 16000023 - Rom_SysControl Does not Update CGU0_CLKOUTSEL:

DESCRIPTION:
The rom_SysControl API does not update CGU0_CLKOUTSEL even when the correct flag (ROM_SYSCTRL_CGU_CLKOUTSEL) has been set.

WORKAROUND:
Write CGU0_CLKOUTSEL directly if it needs to be updated.

APPLIES TO REVISION(S):
0.0

21. 16000024 - Spurious Fault Signaled After Clearing an Externally Generated Fault:

DESCRIPTION:
A spurious fault will be indicated for a single SCLK0 period on SYS_FAULT and $\overline{\text{SYS\_FAULT}}$ under the following conditions:
1. SEC_FCTL.FIEN and SEC_FCTL.FOEN are set.
2. An external device signals a fault on SYS_FAULT or $\overline{\text{SYS\_FAULT}}$.
3. The external fault is cleared by writing 0x00010000 to SEC_FEND.

WORKAROUND:
Clear SEC_FCTL.FIEN before writing to SEC_FEND to clear an externally signaled fault:
1. SEC_FCTL.FIEN and SEC_FCTL.FOEN are set.
2. An external device signals a fault on SYS_FAULT or $\overline{\text{SYS\_FAULT}}$.
3. Clear SEC_FCTL.FIEN
4. Clear the external fault by writing 0x00010000 to SEC_FEND.
5. Set SEC_FCTL.FIEN

APPLIES TO REVISION(S):
0.0, 0.1

22. 16000025 - SPORT May Drive Data Pins During Inactive Channels in Multichannel Mode:

DESCRIPTION:
When a SPORT is operating in multichannel mode, the transmitter tri-states the data pins during the inactive channels. When SPMUX functionality is enabled, under specific conditions, one SPORT half may continue to drive on the inactive channels. This happens when all the below conditions are true.

1. SPORT half "x" is configured as transmitter (SPORT_CTL_x.SPTRAN = 1)
2. Imports Frame sync internally from the pairing half SPORT (SPORT_CTL2_x.FSMUXSEL = 1).
3. Multichannel Frame Delay is zero (SPORT_MCTL_x.MFD = 0)
4. Window Offset is zero (SPORT_MCTL_x.OFFSET = 0)
5. Channel-0 of multichannel frame is enabled for transmission (SPORT_CS0_x.CH0 = 1)
6. Frame sync is active low (SPORT_xCTL.LFS = 1)
7. Frame Sync edge Detect bit is 0 (SPORT_CTL_x.FSED = 0)

If any of these conditions is false, this anomaly does not occur.

When this exact configuration is used, after completion of all the active channels, the SPORT half transmitter drives the first bit of next word to be transmitted once the number of channels specified in WSIZE expires. Therefore the SPORT half may drive on inactive channels which can cause contention when other transmitters configured to drive on these inactive channels.

WORKAROUND:
Avoid any one of the above conditions. For example:

1. Set the Frame Sync edge Detect bit (SPORT_CTL_x.FSED = 1).
2. Use Window Offset other than zero.
3. Use Multichannel Frame Delay other than zero.

APPLIES TO REVISION(S):
0.0, 0.1

23. 16000027 - WUA_BCODE Bit in DPM_RESTORE0 Not Functional:

DESCRIPTION:
The WUA_BCODE bit in the DPM_RESTORE0 register is not functional. This prevents rom_SysControl() from saving the RCU_BCODE register when entering hibernate and from restoring RCU_BCODE when returning from hibernate.

WORKAROUND:
If the contents of RCU_BCODE must be preserved during hibernate:

1. Before entering hibernate store the contents of RCU_BCODE in a register or memory that is preserved during hibernate.
2. After returning from hibernate read back the stored contents of RCU_BCODE and write it to the RCU_BCODE register.

APPLIES TO REVISION(S):
0.0

24. 16000028 - Default SPI Master Boot Mode Setting is Incorrect:

DESCRIPTION:
The SPI0 master boot incorrectly defaults to SPI mode 0 (SPI_CTL.CPOL=0 and SPI_CTL.CPHA=0) instead of SPI Mode 3 (SPI_CTL.CPOL=1 and SPI_CTL.CPHA=1). Consequently bus contention may occur if multi-bit SPI boot is used.

WORKAROUND:
Change to SPI mode 3 before attempting multi-bit SPI transactions.

APPLIES TO REVISION(S):
0.0

25. 16000029 - PPI tDFSPI Timing Does Not Meet Data Sheet Specification:

DESCRIPTION:
The PPI delay specification tDFSPI may be exceeded on the first PPI transfer. In the worst case a 5.5ns delay could be seen.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

26. 16000030 - Interrupted Core Reads of MMRs May Cause Data Loss:

DESCRIPTION:

Certain system MMRs have read side effects, meaning that the contents of the MMRs are altered by reading them. If a core read of one of these system MMRs is interrupted by a maskable interrupt before the contents have been written to the destination register, the system MMR contents are altered and the original data is not preserved. When the core attempts to read the system MMR again, the altered data will be read. The following system MMRs have read side effects:

Register Name	Register Width	Def file macros
RSI_FIFO	32 bits	REG_RSI0_FIFO
CRC_RESULT_CUR	32 bits	REG_CRC0_RESULT_CUR REG_CRC1_RESULT_CUR
CRC_DFIFO	32 bits	REG_CRC0_DFIFO REG_CRC1_DFIFO
TWI_RXDATA8	16 bits	REG_TWI0_RXDATA8 REG_TWI1_RXDATA8
TWI_RXDATA16	16 bits	REG_TWI0_RXDATA16 REG_TWI1_RXDATA16
UART_RBR	32 bits	REG_UART0_RBR REG_UART1_RBR
LP_RX	32 bits	REG_LP0_RX REG_LP1_RX REG_LP2_RX REG_LP3_RX
SDU_DMARD	32 bits	REG_SDU0_DMARD
SPORT_RXPRI_A	32 bits	REG_SPORT0_RXPRI_A REG_SPORT1_RXPRI_A REG_SPORT2_RXPRI_A
SPORT_RXSEC_A	32 bits	REG_SPORT0_RXSEC_A REG_SPORT1_RXSEC_A REG_SPORT2_RXSEC_A
SPORT_RXPRI_B	32 bits	REG_SPORT0_RXPRI_B REG_SPORT1_RXPRI_B REG_SPORT2_RXPRI_B
SPORT_RXSEC_B	32 bits	REG_SPORT0_RXSEC_B REG_SPORT1_RXSEC_B REG_SPORT2_RXSEC_B
SPI_RFIFO	32 bits	REG_SPI0_RFIFO REG_SPI1_RFIFO
USB_INTRRX	16 bits	REG_USB0_INTRRX
USB_INTRTX	16 bits	REG_USB0_INTRTX
USB_IRQ	16 bits	REG_USB0_IRQ
USB_LPM_IRQ	16 bits	REG_USB0_LPM_IRQ
USB_DMA_IRQ	16 bits	REG_USB0_DMA_IRQ
EMAC_DMA_MISS_FRM	32 bits	REG_EMAC0_DMA_MISS_FRM REG_EMAC1_DMA_MISS_FRM
EMAC_ISTAT	32 bits	REG_EMAC0_ISTAT REG_EMAC1_ISTAT
EMAC_TM_STMPSTAT	32 bits	REG_EMAC0_TM_STMPSTAT REG_EMAC1_TM_STMPSTAT
EMAC_TXOCTCNT_GB	32 bits	REG_EMAC0_TXOCTCNT_GB REG_EMAC1_TXOCTCNT_GB
EMAC_TXFRMCNT_GB	32 bits	REG_EMAC0_TXFRMCNT_GB REG_EMAC1_TXFRMCNT_GB
EMAC_TXBCASTFRM_G	32 bits	REG_EMAC0_TXBCASTFRM_G REG_EMAC1_TXBCASTFRM_G
EMAC_TXMCASTFRM_G	32 bits	REG_EMAC0_TXMCASTFRM_G REG_EMAC1_TXMCASTFRM_G
EMAC_TX64_GB	32 bits	REG_EMAC0_TX64_GB REG_EMAC1_TX64_GB
EMAC_TX65TO127_GB	32 bits	REG_EMAC0_TX65TO127_GB REG_EMAC1_TX65TO127_GB
EMAC_TX128TO255_GB	32 bits	REG_EMAC0_TX128TO255_GB REG_EMAC1_TX128TO255_GB
EMAC_TX256TO511_GB	32 bits	REG_EMAC0_TX256TO511_GB REG_EMAC1_TX256TO511_GB
EMAC_TX512TO1023_GB	32 bits	REG_EMAC0_TX512TO1023_GB REG_EMAC1_TX512TO1023_GB
EMAC_TX1024TOMAX_GB	32 bits	REG_EMAC0_TX1024TOMAX_GB REG_EMAC1_TX1024TOMAX_GB
EMAC_TXUCASTFRM_GB	32 bits	REG_EMAC0_TXUCASTFRM_GB REG_EMAC1_TXUCASTFRM_GB
EMAC_TXMCASTFRM_GB	32 bits	REG_EMAC0_TXMCASTFRM_GB REG_EMAC1_TXMCASTFRM_GB
EMAC_TXBCASTFRM_GB	32 bits	REG_EMAC0_TXBCASTFRM_GB REG_EMAC1_TXBCASTFRM_GB
EMAC_TXUNDR_ERR	32 bits	REG_EMAC0_TXUNDR_ERR REG_EMAC1_TXUNDR_ERR
EMAC_TXSNGCOL_G	32 bits	REG_EMAC0_TXSNGCOL_G REG_EMAC1_TXSNGCOL_G
EMAC_TXMULTCOL_G	32 bits	REG_EMAC0_TXMULTCOL_G REG_EMAC1_TXMULTCOL_G
EMAC_TXDEFERRED	32 bits	REG_EMAC0_TXDEFERRED REG_EMAC1_TXDEFERRED
EMAC_TXLATECOL	32 bits	REG_EMAC0_TXLATECOL REG_EMAC1_TXLATECOL

Register Name	Register Width	Def file macros
EMAC_TXEXCESSCOL	32 bits	REG_EMAC0_TXEXCESSCOL REG_EMAC1_TXEXCESSCOL
EMAC_TXCARR_ERR	32 bits	REG_EMAC0_TXCARR_ERR REG_EMAC1_TXCARR_ERR
EMAC_TXOCTCNT_G	32 bits	REG_EMAC0_TXOCTCNT_G REG_EMAC1_TXOCTCNT_G
EMAC_TXFRMCNT_G	32 bits	REG_EMAC0_TXFRMCNT_G REG_EMAC1_TXFRMCNT_G
EMAC_TXEXCESSDEF	32 bits	REG_EMAC0_TXEXCESSDEF REG_EMAC1_TXEXCESSDEF
EMAC_TXPAUSEFRM	32 bits	REG_EMAC0_TXPAUSEFRM REG_EMAC1_TXPAUSEFRM
EMAC_TXVLANFRM_G	32 bits	REG_EMAC0_TXVLANFRM_G REG_EMAC1_TXVLANFRM_G
EMAC_RXFRMCNT_GB	32 bits	REG_EMAC0_RXFRMCNT_GB REG_EMAC1_RXFRMCNT_GB
EMAC_RXOCTCNT_GB	32 bits	REG_EMAC0_RXOCTCNT_GB REG_EMAC1_RXOCTCNT_GB
EMAC_RXOCTCNT_G	32 bits	REG_EMAC0_RXOCTCNT_G REG_EMAC1_RXOCTCNT_G
EMAC_RXBCASTFRM_G	32 bits	REG_EMAC0_RXBCASTFRM_G REG_EMAC1_RXBCASTFRM_G
EMAC_RXMCASTFRM_G	32 bits	REG_EMAC0_RXMCASTFRM_G REG_EMAC1_RXMCASTFRM_G
EMAC_RXCRC_ERR	32 bits	REG_EMAC0_RXCRC_ERR REG_EMAC1_RXCRC_ERR
EMAC_RXALIGN_ERR	32 bits	REG_EMAC0_RXALIGN_ERR REG_EMAC1_RXALIGN_ERR
EMAC_RXRUNT_ERR	32 bits	REG_EMAC0_RXRUNT_ERR REG_EMAC1_RXRUNT_ERR
EMAC_RXJAB_ERR	32 bits	REG_EMAC0_RXJAB_ERR REG_EMAC1_RXJAB_ERR
EMAC_RXUSIZE_G	32 bits	REG_EMAC0_RXUSIZE_G REG_EMAC1_RXUSIZE_G
EMAC_RXOSIZE_G	32 bits	REG_EMAC0_RXOSIZE_G REG_EMAC1_RXOSIZE_G
EMAC_RX64_GB	32 bits	REG_EMAC0_RX64_GB REG_EMAC1_RX64_GB
EMAC_RX65TO127_GB	32 bits	REG_EMAC0_RX65TO127_GB REG_EMAC1_RX65TO127_GB
EMAC_RX128TO255_GB	32 bits	REG_EMAC0_RX128TO255_GB REG_EMAC1_RX128TO255_GB
EMAC_RX256TO511_GB	32 bits	REG_EMAC0_RX256TO511_GB REG_EMAC1_RX256TO511_GB
EMAC_RX512TO1023_GB	32 bits	REG_EMAC0_RX512TO1023_GB REG_EMAC1_RX512TO1023_GB
EMAC_RX1024TOMAX_GB	32 bits	REG_EMAC0_RX1024TOMAX_GB REG_EMAC1_RX1024TOMAX_GB
EMAC_RXUCASTFRM_G	32 bits	REG_EMAC0_RXUCASTFRM_G REG_EMAC1_RXUCASTFRM_G
EMAC_RXLEN_ERR	32 bits	REG_EMAC0_RXLEN_ERR REG_EMAC1_RXLEN_ERR
EMAC_RXOORTYPE	32 bits	REG_EMAC0_RXOORTYPE REG_EMAC1_RXOORTYPE
EMAC_RXPAUSEFRM	32 bits	REG_EMAC0_RXPAUSEFRM REG_EMAC1_RXPAUSEFRM
EMAC_RXFIFO_OVF	32 bits	REG_EMAC0_RXFIFO_OVF REG_EMAC1_RXFIFO_OVF
EMAC_RXVLANFRM_GB	32 bits	REG_EMAC0_RXVLANFRM_GB REG_EMAC1_RXVLANFRM_GB
EMAC_RXWDOG_ERR	32 bits	REG_EMAC0_RXWDOG_ERR REG_EMAC1_RXWDOG_ERR
EMAC_IPC_RXIMSK	32 bits	REG_EMAC0_IPC_RXIMSK REG_EMAC1_IPC_RXIMSK
EMAC_IPC_RXINT	32 bits	REG_EMAC0_IPC_RXINT REG_EMAC1_IPC_RXINT
EMAC_RXIPV4_GD_FRM	32 bits	REG_EMAC0_RXIPV4_GD_FRM REG_EMAC1_RXIPV4_GD_FRM
EMAC_RXIPV4_HDR_ERR_FRM	32 bits	REG_EMAC0_RXIPV4_HDR_ERR_FRM REG_EMAC1_RXIPV4_HDR_ERR_FRM
EMAC_RXIPV4_NOPAY_FRM	32 bits	REG_EMAC0_RXIPV4_NOPAY_FRM REG_EMAC1_RXIPV4_NOPAY_FRM
EMAC_RXIPV4_FRAG_FRM	32 bits	REG_EMAC0_RXIPV4_FRAG_FRM REG_EMAC1_RXIPV4_FRAG_FRM
EMAC_RXIPV4_UDSBL_FRM	32 bits	REG_EMAC0_RXIPV4_UDSBL_FRM REG_EMAC1_RXIPV4_UDSBL_FRM
EMAC_RXIPV6_GD_FRM	32 bits	REG_EMAC0_RXIPV6_GD_FRM REG_EMAC1_RXIPV6_GD_FRM
EMAC_RXIPV6_HDR_ERR_FRM	32 bits	REG_EMAC0_RXIPV6_HDR_ERR_FRM REG_EMAC1_RXIPV6_HDR_ERR_FRM
EMAC_RXIPV6_NOPAY_FRM	32 bits	REG_EMAC0_RXIPV6_NOPAY_FRM REG_EMAC1_RXIPV6_NOPAY_FRM
EMAC_RXUDP_GD_FRM	32 bits	REG_EMAC0_RXUDP_GD_FRM REG_EMAC1_RXUDP_GD_FRM
EMAC_RXUDP_ERR_FRM	32 bits	REG_EMAC0_RXUDP_ERR_FRM REG_EMAC1_RXUDP_ERR_FRM
EMAC_RXTCP_GD_FRM	32 bits	REG_EMAC0_RXTCP_GD_FRM REG_EMAC1_RXTCP_GD_FRM
EMAC_RXTCP_ERR_FRM	32 bits	REG_EMAC0_RXTCP_ERR_FRM REG_EMAC1_RXTCP_ERR_FRM
EMAC_RXICMP_GD_FRM	32 bits	REG_EMAC0_RXICMP_GD_FRM REG_EMAC1_RXICMP_GD_FRM

Register Name	Register Width	Def file macros
EMAC_RXICMP_ERR_FRM	32 bits	REG_EMAC0_RXICMP_ERR_FRM REG_EMAC1_RXICMP_ERR_FRM
EMAC_RXIPV4_GD_OCT	32 bits	REG_EMAC0_RXIPV4_GD_OCT REG_EMAC1_RXIPV4_GD_OCT
EMAC_RXIPV4_HDR_ERR_OCT	32 bits	REG_EMAC0_RXIPV4_HDR_ERR_OCT REG_EMAC1_RXIPV4_HDR_ERR_OCT
EMAC_RXIPV4_NOPAY_OCT	32 bits	REG_EMAC0_RXIPV4_NOPAY_OCT REG_EMAC1_RXIPV4_NOPAY_OCT
EMAC_RXIPV4_FRAG_OCT	32 bits	REG_EMAC0_RXIPV4_FRAG_OCT REG_EMAC1_RXIPV4_FRAG_OCT
EMAC_RXIPV4_UDSBL_OCT	32 bits	REG_EMAC0_RXIPV4_UDSBL_OCT REG_EMAC1_RXIPV4_UDSBL_OCT
EMAC_RXIPV6_GD_OCT	32 bits	REG_EMAC0_RXIPV6_GD_OCT REG_EMAC1_RXIPV6_GD_OCT
EMAC_RXIPV6_HDR_ERR_OCT	32 bits	REG_EMAC0_RXIPV6_HDR_ERR_OCT REG_EMAC1_RXIPV6_HDR_ERR_OCT
EMAC_RXIPV6_NOPAY_OCT	32 bits	REG_EMAC0_RXIPV6_NOPAY_OCT REG_EMAC1_RXIPV6_NOPAY_OCT
EMAC_RXUDP_GD_OCT	32 bits	REG_EMAC0_RXUDP_GD_OCT REG_EMAC1_RXUDP_GD_OCT
EMAC_RXUDP_ERR_OCT	32 bits	REG_EMAC0_RXUDP_ERR_OCT REG_EMAC1_RXUDP_ERR_OCT
EMAC_RXTCP_GD_OCT	32 bits	REG_EMAC0_RXTCP_GD_OCT REG_EMAC1_RXTCP_GD_OCT
EMAC_RXTCP_ERR_OCT	32 bits	REG_EMAC0_RXTCP_ERR_OCT REG_EMAC1_RXTCP_ERR_OCT
EMAC_RXICMP_GD_OCT	32 bits	REG_EMAC0_RXICMP_GD_OCT REG_EMAC1_RXICMP_GD_OCT
EMAC_RXICMP_ERR_OCT	32 bits	REG_EMAC0_RXICMP_ERR_OCT REG_EMAC1_RXICMP_ERR_OCT

WORKAROUND:
Masking all maskable interrupts prior to reading system MMRs with read side effects will prevent data loss. Interrupts can be re-enabled subsequently.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices (CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details. For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
0.0

27. 16000031 - Incorrect Default USB_PLL_OSC.PLLM Value:

DESCRIPTION:
The default value of USB_PLL_OSC.PLLM is incorrectly set to 20 (decimal) instead of 10, which is the only supported value for USB operation.

WORKAROUND:
Program USB_PLL_OSC.PLLM to 10 before using USB.

APPLIES TO REVISION(S):
0.0

28. 16000032 - Core Reads of System MMRs May Cause the Core to Hang:

DESCRIPTION:
System MMR reads by a core in parallel with or immediately preceding an instruction that can cause an extended memory stall can leave the core in a state of lockup.

WORKAROUND:
Both workarounds must be implemented to guarantee that the problem will not occur.

1. Ensure there are no system MMR reads in parallel.
2. Place a NOP instruction after each system MMR read instruction.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices (CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details. For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
0.0

29. 16000033 - PPI Data Underflow on First Word Not Reported in Certain Modes:

DESCRIPTION:
In PPI transmit modes with external frame sync(s) PPI_STAT.CFIFOERR and PPI_STAT.YFIFOERR aren't reported reliably if the first data word underflows. If the second data word to be transmitted also underflows, PPI reports the underflow error correctly.

WORKAROUND:
Use the second data word underflow notification to take any required action or use a different PPI mode.

APPLIES TO REVISION(S):
0.0, 0.1

30. 16000034 - CNV1 Red Pixel Substitution feature not functional in the PVP:

DESCRIPTION:
The Red Pixel Substitution feature is not functional in PVP Convolution Block CNV1. The RFRMT bit field in PVP_CNV1_CTL[13:12] must be programmed with zeroes. As a consequence full frame resolution is not available from IPF0 when processing Bayer and RCCC streams.

WORKAROUND:
1. IPF0 can be used to either extract red (PVP_IPF0_CTL.EXTRED=1) or green (PVP_IPF0_CTL.EXTRED=0) pixels, if color modes CFRMT=5 or CFRMT=6 are chosen.
2. Bayer data can be converted to monochrome by Convolution blocks CNV0 to CNV3, when their 2:1 down scaling functionality is enabled.

APPLIES TO REVISION(S):
0.0

31. 16000035 - IPF0 Output Port Color Separation feature not functional:

DESCRIPTION:
The Output Port Color Separation feature is not supported in the PVP camera pipe input formatter (IPF0). The BFRMT bit field in PVP_IPF0_CTL[30:28] must be programmed with zeroes.

WORKAROUND:
Individual color components can be extracted by convolution blocks CNV0 to CNV3, when their 2:1 down scaling functionality is enabled.

APPLIES TO REVISION(S):
0.0

32. 16000036 - Spurious USB Wake From Hibernate May Occur When USB_VBUS is Low:

DESCRIPTION:
If USB is enabled and the USB_PHY_CTL.HIBER bit is set when the processor enters hibernate with USB_VBUS removed, the processor will wake up spuriously. The USB wakeup is the source of the wakeup event even though there's no USB connection. Note that if the VBUS was on when entering hibernate, removal of VBUS during hibernate will cause the processor to wake up, since any changes in VBUS or the data lines D+/D- trigger the USB wakeup event. This anomaly only applies to the case where VBUS is removed prior to entering hibernate.

WORKAROUND:
There are two possible workarounds:

1. Do not enable USB as a wake up source in the DPM0_WAKE_EN register if the application may go into hibernate when the USB connection is removed.

2. Ensure that VBUS is present before going to hibernate by doing the following in the routine that enters hibernate:

```
Wait 50 ms to ensure that USB_DEV_CTL reflects accurately the VBUS level.
Read VBUS level (bits 4:3) in USB_DEV_CTL register:
   If 11b --> vbus is on, Enter hibernate
   Else --> vbus is off, jump back to Wait 50ms
Clear bit 0 of USB_PHY_CTL (hibernate bit)
Enter hibernate
```

APPLIES TO REVISION(S):
0.0

33. 16000037 - Core RAISE 2 Instruction Not Latched When Executed at Priority Level 0, 1, or 2:

DESCRIPTION:
When a core executes the RAISE 2 instruction at priority level 0 (Reset), 1 (Emulation), or 2 (NMI) the RAISE 2 request is not latched. Consequently, the core will not vector to the level 2 ISR after returning from the priority level 0, 1, or 2 routine which contained the RAISE 2 instruction.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

34. 16000038 - Spurious Unhandled NMI or L1 Memory Parity Error Interrupt May Occur Upon Entering the NMI ISR:

DESCRIPTION:
The Unhandled NMI or L1 Memory Parity Error Interrupt (ID 10 for core 0 and ID 11 for core 1) may spuriously occur upon entering an NMI ISR. Notwithstanding this item, the documented functionality of the interrupt works as intended.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

35. 16000039 - CGU_STAT.PLOCKERR Bit May be Unreliable:

DESCRIPTION:
The CGU_STAT.PLOCKERR bit may not be set in the case of the PLL failing to lock at the programmed frequency. However, as expected, CGU_STAT.PLOCKERR will not assert when a PLL lock is successful.

WORKAROUND:
Do not depend on CGU_STAT.PLOCKERR to detect failure of the PLL to lock. In the case of safety-critical systems SYS_CLKOUT or any other internally clocked output may be monitored to ensure that no software or hardware issue has prevented the PLL from locking at the programmed frequency.

APPLIES TO REVISION(S):
0.0, 0.1

36. 16000040 - JTAG Emulator Reads of SDU_IDCODE Alter Register Contents:

DESCRIPTION:
After connecting to the processor using an emulator debug session, the value stored in SDU_IDCODE will be read as 0x00000000 if the JTAG emulator does not reload the correct value after reading.

WORKAROUND:
There are three possible workarounds:
1. Read SDU_IDCODE before connecting with an emulator debug session.
2. Use a development tool chain that reloads the SDU_IDCODE after reading. For tool chains and operating systems supported by Analog Devices (CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details. For all other tool chains and operating systems, see the appropriate supporting documentation for details.
3. Read a different register to distinguish between silicon revisions, if such register differences (shown in this list) exist between the silicon revisions of interest.

APPLIES TO REVISION(S):
0.0, 0.1

37. 16000041 - IFLUSH Instruction Causes Parity Error When Parity Is Enabled:

DESCRIPTION:
Executing IFLUSH with instruction cache and parity enabled (assuming the valid bit is active on the cache line in question) will result in incorrect parity being generated due to the clearing of the valid bit in the cache TAG without changing the parity bit appropriately. The next time that location is accessed, a parity error will be generated.

This anomaly does not apply to data cache.

WORKAROUND:
The ITEST_COMMAND functionality can be used as a substitute for the IFLUSH instruction, as the cache tag parity is recalculated in hardware when the tag is written via the ITEST_COMMAND register. For example, consider the pseudo-code:

```
IFLUSH(flush_addr);
```

The above functionality can be achieved by manually snooping the cache tag arrays to locate where in the cache it resides and invalidating that line:

```
volatile char i;
int itest_cmd, itest_data0;

for(i = 0; i < 4; i++)
{
   itest_cmd = ((i<<26) | ((flush_addr & 0x00003000)<<4) | (flush_addr & 0x000003E0));
         /* ITEST_COMMAND[27:26] = WAY # (loop count i) */
         /* ITEST_COMMAND[17:16] = flush_addr[13:12]    */
         /* ITEST_COMMAND[09:05] = flush_addr[09:05]    */

   *pITEST_COMMAND = itest_cmd;       /* write to command register */
   ssync();                           /* must be followed by ssync */

   itest_data0 = *pITEST_DATA0;       /* Get Cache Tag */
   ssync();

   /* Check for valid bit AND address match (bits 31:14 and bits 11:10) */
   if( (itest_data0 & 1) && ( (itest_data0 & 0xFFFFCC00) == (flush_addr & 0xFFFFCC00) ) )
   {
      /* Clear bit 0 to invalidate the line */
      *pITEST_DATA0 = itest_data0 & 0xFFFFFFFE;
      ssync();

      /* Change command to cache tag write */
      *pITEST_COMMAND = itest_cmd | 0x00000002;
      ssync();
      break;    /* We've done what we need to do, so exit */
   } /* if no tag match or not valid, do nothing */
} /* if no ways hit, do nothing */
```

APPLIES TO REVISION(S):
0.0, 0.1

38. 16000042 - Instruction Cache Failure When Parity Is Enabled:

DESCRIPTION:
When parity checking is enabled, an instruction cache line fill can be abandoned if a change in control occurs on the processor when the cache line fill is initiated with a particular cycle alignment relative to the cache fill initiation. When this failure occurs, the cache line is marked valid for the corresponding cache TAG, but the cache line fill itself does not happen. As a result, the processor executes the stale cached instruction rather than the one that should have been there had the new cache line been filled properly.

This anomaly does not apply to data cache.

WORKAROUND:
There is no workaround possible to keep both instruction cache and parity enabled. If either is disabled, this anomaly will not occur.

APPLIES TO REVISION(S):
0.1