Blackfin Embedded Processor

Silicon Anomaly List — ADSP-BF531/BF532/BF533

ABOUT ADSP-BF531/BF532/BF533 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF531/BF532/BF533 product(s) and the functionality specified in the ADSP-BF531/BF532/BF533 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 implementation field bits <15:0> of the DSPID core MMR register can be used to differentiate the revisions as shown below.

Silicon REVISION	DSPID<15:0>
0.6	0x0006
0.5	0x0005

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
03/14/2014	H	I	Added Anomalies - 05000503, 05000506 Removed Silicon Revisions 0.3 and 0.4
05/23/2011	G	H	Added Anomalies - 05000489, 05000491, 05000494, 05000501
05/25/2010	F	F	Added Anomalies - 05000443, 05000461, 05000462, 05000471, 05000473, 05000475, 05000477, 05000481
09/18/2008	E	F	Added Anomalies - 05000425, 05000426 Revised Anomalies - 05000283, 05000315
06/18/2008	D	F	Added Silicon Revision 0.6 Added Anomalies - 05000416
02/08/2008	C	E	Added Anomalies - 05000363, 05000400, 05000402, 05000403
12/10/2007	B	E	Added Anomalies - 05000366, 05000371
09/04/2007	A	E	Initial Consolidated Revision - Replaces anomaly lists for ADSP-BF531 (Rev W), ADSP-BF532 (Rev AB) and ADSP-BF533 (Rev X) Added Anomalies - 05000357 Revised Anomalies - 05000311

NR003532H

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-BF531/BF532/BF533 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.5	0.6
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x
2	05000105	Watchpoint Status Register (WPSTAT) Bits Are Set on Every Corresponding Match	x	x
3	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x
4	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x
5	05000166	PPI Data Lengths between 8 and 16 Do Not Zero Out Upper Bits	x	x
6	05000167	Turning SPORTs on while External Frame Sync Is Active May Corrupt Data	x	x
7	05000180	PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs	x	x
8	05000208	VSTAT Status Bit in PLL_STAT Register Is Not Functional	x	x
9	05000219	NMI Event at Boot Time Results in Unpredictable State	x	x
10	05000229	SPI Slave Boot Mode Modifies Registers from Reset Value	x	x
11	05000233	PPI_FS3 Is Not Driven in 2 or 3 Internal Frame Sync Transmit Modes	x	.
12	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
13	05000254	Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock	x	x
14	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x
15	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x
16	05000273	Writes to Synchronous SDRAM Memory May Be Lost	x	.
17	05000276	Timing Requirements Change for External Frame Sync PPI Modes with Non-Zero PPI_DELAY	x	x
18	05000277	Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt	x	.
19	05000278	Disabling Peripherals with DMA Running May Cause DMA System Instability	x	.
20	05000281	False Hardware Error when ISR Context Is Not Restored	x	.
21	05000282	Memory DMA Corruption with 32-Bit Data and Traffic Control	x	.
22	05000283	System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage	x	.
23	05000288	SPORTs May Receive Bad Data If FIFOs Fill Up	x	.
24	05000301	Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space	x	.
25	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
26	05000311	Erroneous Flag (GPIO) Pin Operations under Specific Sequences	x	.
27	05000312	Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted	x	.
28	05000313	PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes	x	.
29	05000315	Killed System MMR Write Completes Erroneously on Next System MMR Access	x	.
30	05000319	Internal Voltage Regulator Values of 1.05V, 1.10V and 1.15V Not Allowed for LQFP Packages	x	.
31	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	.
32	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x
33	05000371	Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration	x	.
34	05000400	PPI Does Not Start Properly In Specific Mode	x	.
35	05000402	SSYNC Stalls Processor when Executed from Non-Cacheable Memory	x	.
36	05000403	Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall	x	x
37	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x
38	05000425	Multichannel SPORT Channel Misalignment Under Specific Configuration	x	x
39	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
40	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x
41	05000461	False Hardware Error when RETI Points to Invalid Memory	x	x
42	05000462	Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign	x	x

No.	ID	Description	0.5	0.6
43	05000471	Boot Failure When SDRAM Control Signals Toggle Coming Out Of Reset	x	x
44	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x	x
45	05000475	Possible Lockup Condition when Modifying PLL from External Memory	x	x
46	05000477	TESTSET Instruction Cannot Be Interrupted	x	x
47	05000481	Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption	x	x
48	05000489	PLL May Latch Incorrect Values Coming Out of Reset	x	x
49	05000491	Instruction Memory Stalls Can Cause IFLUSH to Fail	x	x
50	05000494	EXCPT Instruction May Be Lost If NMI Happens Simultaneously	x	x
51	05000501	RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes	x	x
52	05000503	SPORT Sign-Extension May Not Work	x	x
53	05000506	Hardware Loop Can Underflow Under Specific Conditions	x	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-BF531/BF532/BF533 including a description, workaround, and identification of applicable silicon revisions.

1. 05000074 - 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 (VisualDSP++, VDK, 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):
0.5, 0.6

2. 05000105 - Watchpoint Status Register (WPSTAT) Bits Are Set on Every Corresponding Match:

DESCRIPTION:
Even when the Watchpoint Data Address Counters (WPDACTL:WPDCNTENx) are enabled, the corresponding Watchpoint Status Register bits (WPSTAT:STATDAx) will be set on every match, not just on the expiration of the counter.

The same is true for the Watchpoint Instruction Address Counters (WPIACTL:WPICNTENx) and Status Bits (WPSTAT:STATIAx).

WORKAROUND:
When a watchpoint interrupt occurs, you must validate the set WPSTAT bits with their counter enable bits and counter register values (WPIACNTn or WPDACTn).

Note: Because the Counter Register only decrements to 0x0000, its value will equal 0x0000 when the counter has expired AND when it is 1 match away from its counter expiring.

APPLIES TO REVISION(S):
0.5, 0.6

3. 05000119 - DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops:

DESCRIPTION:
After completion of a Peripheral Receive DMA, the DMAx_IRQ_STATUS:DMA_RUN bit will be in an undefined state.

WORKAROUND:
The DMA interrupt and/or the DMAx_IRQ_STATUS:DMA_DONE bits should be used to determine when the channel has completed running.

APPLIES TO REVISION(S):
0.5, 0.6

4. 05000122 - Rx.H Cannot Be Used to Access 16-bit System MMR Registers:

DESCRIPTION:
When accessing 16-bit system MMR registers, the high half of the data registers may not be used. If a high half register is used, incorrect data will be written to the system MMR register, but no exception will be generated. For example, this access would fail:

```
W[P0] = R5.H;   // P0 points to a 16-bit System MMR
```

WORKAROUND:
Use other forms of 16-bit transfers when accessing 16-bit system MMR registers. For example:

```
W[P0] = R5.L;   // P0 points to a 16-bit System MMR
R4.L = W[P0];
R3 = W[P0](Z);
W[P0] = R3;
```

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 (VisualDSP++, VDK, 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):
0.5, 0.6

5. 05000166 - PPI Data Lengths between 8 and 16 Do Not Zero Out Upper Bits:

DESCRIPTION:
For PPI data lengths greater than 8 and less than 16, the upper bits received into memory that are not part of the PPI data should be zero. For example, if the user is using 10-bit PPI data length, the upper 6 bits in memory should be zero. Instead, the PPI captures whatever data is on the upper 6 PPI data pins (muxed as PFx pins).

WORKAROUND:
The software workaround is to mask out the upper 6 bits when processing received data.

APPLIES TO REVISION(S):
0.5, 0.6

6. 05000167 - Turning SPORTs on while External Frame Sync Is Active May Corrupt Data:

DESCRIPTION:
The SPORTs are level sensitive to External Frame Syncs. If a SPORT is configured for External Frame Syncs and the frame sync is active when the SPORT is first enabled, the SPORT will start receiving data immediately when enabled. This may occur in the middle of a frame, causing incorrect data to be received.

This anomaly also applies to Stereo Serial Modes (I^2S and variants), except in the case where either the LRFS or the RRFST bit is set (not both).

WORKAROUND:
Hold off external Frame syncs until the SPORT is fully enabled. If you use a serial device with external frame syncs that can't be held off until the SPORT is enabled, a programmable flag pin can be connected to the Frame Sync. The PFx pin can be programmed to continually sample the SPORT Frame Sync and then enable the SPORT when the RFS / TFS signals are in the inactive state.

For Stereo Serial Modes, either set the LRFS or the RRFST bit (not both), if possible.

APPLIES TO REVISION(S):
0.5, 0.6

7. 05000180 - PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs:

DESCRIPTION:
In self-triggered, continuous sampling operation of the PPI, the delay count specified in the PPI_DELAY register is ignored. As soon as this mode is enabled, data is transferred.

WORKAROUND:
If a delay is needed, either ignore received data in software or use a mode with at least one frame sync.

APPLIES TO REVISION(S):
0.5, 0.6

8. 05000208 - VSTAT Status Bit in PLL_STAT Register Is Not Functional:

DESCRIPTION:
The VSTAT status bit in the PLL_STAT register does not function. Relying on its value to determine whether the internal voltage regulator has settled is not recommended.

WORKAROUND:
When changing the voltage via the internal voltage regulator, allow at least 40usec for the voltage change to take place. After 40usec, the new value will be set, regardless of the state of the VSTAT bit.

APPLIES TO REVISION(S):
0.5, 0.6

9. 05000219 - NMI Event at Boot Time Results in Unpredictable State:

DESCRIPTION:
If the NMI pin is asserted at boot time, the boot process will fail because there is no handler in the boot ROM. The behavior is not predictable.

WORKAROUND:
Do not assert the NMI pin during a boot sequence.

APPLIES TO REVISION(S):
0.5, 0.6

10. 05000229 - SPI Slave Boot Mode Modifies Registers from Reset Value:

DESCRIPTION:
In this Boot Mode, the DMA5_CONFIG and SPI_CTL registers are not restored to their default (reset) states before executing the user's application code. The DMA5 channel remains enabled in stop mode and the SPI remains enabled in RX DMA mode.

WORKAROUND:
The user's application must reset these registers before either the SPI or DMA channel 5 can be used.

APPLIES TO REVISION(S):
0.5, 0.6

11. 05000233 - PPI_FS3 Is Not Driven in 2 or 3 Internal Frame Sync Transmit Modes:

DESCRIPTION:
In this mode, if the PORT_CFG field in the PPI_CONTROL register is set to #b11 (Sync PPI_FS3 to PPI_FS2), the PPI_FS3 frame sync signal is not driven to the PF3 flag pin. It is, however, correctly driven to PF3 when the PORT_CFG field is set to #b01 (Sync PPI_FS3 to PPI_FS1).

WORKAROUND:
None

APPLIES TO REVISION(S):
0.5

12. 05000245 - 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 (VisualDSP++, VDK, 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):

0.5, 0.6

13. 05000254 - Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock:

DESCRIPTION:
If a Timer is in PWM_OUT mode AND is clocked by an external clock as opposed to the system clock (i.e., clocked by a signal applied to either PPI_CLK or a flag pin) AND is in single-pulse mode (PERIOD_CNT = 0), then the generated pulse width may be off by +1 or -1 count. All other modes are not affected by this anomaly.

WORKAROUND:
The suggested workaround is to use continuous mode instead of the single-pulse mode. You may enable the timer and immediately disable it again. The timer will generate a single pulse and count to the end of the period before effectively disabling itself. The generated waveform will be of the desired length.

If PULSEWIDTH is the desired width, the following sequence will produce a single pulse:

```
TIMERx_CONFIG = PWM_OUT|CLK_SEL|PERIOD_CNT|IRQ_ENA;  // Optional: PULSE_HI|TIN_SEL|EMU_RUN
TIMERx_PERIOD = PULSEWIDTH + 2;                      // Slightly bigger than the width
TIMERx_WIDTH  = PULSEWIDTH;
TIMER_ENABLE  = TIMENx;
TIMER_DISABLE = TIMDISx;
<wait for interrupt (at end of period)>
```

APPLIES TO REVISION(S):
0.5, 0.6

14. 05000265 - Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks:

DESCRIPTION:

A noisy board environment combined with slow input edge rates on external SPORT receive (RSCLK) and transmit clocks (TSCLK) may cause a variety of observable problems. Unexpected high frequency transitions on the RSCLK/TSCLK can cause the SPORT to recognize an extra noise-induced glitch clock pulse.

The high frequency transitions on the RSCLK/TSCLK are most likely to be caused by noise on the rising or falling edge of external serial clocks. This noise, coupled with a slowly transitioning serial clock signal, can cause an additional bit-clock with a short period due to high sensitivity of the clock input. A slow slew rate input allows any noise on the clock input around the switching point to cause the clock input to cross and re-cross the switching point. This oscillation can cause a glitch clock pulse in the internal logic of the serial port.

Problems which may be observed due to this glitch clock pulse are:

- In stereo serial modes, this will show up as missed frame syncs, causing left/right data swaps.
- In multichannel mode, this will show up as MFD counts appearing inaccurate or skipped frames.
- In Normal (Early) Frame sync mode, data words received will be shifted right one bit. The MSB may be incorrectly captured in sign extension mode.
- In any mode, received or transmitted data words may appear to be partially right shifted if noise occurs on any input clocks between the start of frame sync and the last bit to be received or transmitted.

In Stereo Serial mode (bit 9 set in SPORTx_RCR2), unexpected high frequency transitions on RSCLK/TSCLK can cause the SPORT to miss rising or falling edges of the word clock. This causes left or right words of Stereo Serial data to be lost. This may be observed as a Left/Right channel swap when listening to stereo audio signals. The additional noise-induced bit-clock pulse on the SPORT's internal logic results in the FS edge-detection logic generating a pulse with a smaller width and, at the same time, prevents the SPORT from detecting the external FS signal during the next 'normal' bit-clock period. The FS pulse with smaller width, which is the output of the edge-detection logic, is ignored by the SPORT's sequential logic. Due to the fact that the edge detection part of the FS-logic was already 'triggered', the next 'normal' RSCLK will not detect the change in RFS anymore. In I^2S/EIAJ mode, this results in one stereo sample being detected/transferred as two left/right channels, and all subsequent channels will be word-swapped in memory.

In multichannel mode, the mutlichannel frame delay (MFD) logic receives the extra sync pulse and begins counting early or double counting (if the count has already begun). A MFD of zero can roll over to 15, as the count begins one cycle early.

In early frame sync mode, if the noise occurs on the driving edge of the clock the same cycle that FS becomes active, the FS logic receives the extra runt pulse and begins counting the word length one cycle early. The first bit will be sampled twice and the last bit will be skipped.

In all modes, if the noise occurs in any cycle after the FS becomes active, the bit counting logic receives the extra runt pulse and advances too rapidly. If this occurs once during a work unit, it will finish counting the word length one cycle early. The bit where the noise occurs will be sampled twice, and the last bit will be skipped.

WORKAROUND:

1) Decrease the sensitivity to noise by increasing the slew rate of the bit clock or make the rise and fall times of serial bit clocks short, such that any noise around the transition produces a short duration noise-induced bit-clock pulse. This small high-frequency pulse will not have any impact on the SPORT or on the detection of the frame-sync. Sharpen edges as much as possible, if this is suitable and within EMI requirements.
2) If possible, use internally generated bit-clocks and frame-syncs.
3) Follow good PCB design practices. Shield RSCLK with respect to TSCLK lines to minimize coupling between the serial clocks.
4) Separate RSCLK, TSCLK, and Frame Sync traces on the board to minimize coupling which occurs at the driving edge when FS switches.

A specific workaround for problems observed in Stereo Serial mode is to delay the frame-sync signal such that noise-induced bit-clock pulses do not start processing the frame-sync. This can be achieved if there is a larger serial resistor in the frame-sync trace than the one in the bit-clock trace. Frame-sync transitions should not cross the 50% point until the bit-clock crosses the 10% of VDD threshold (for a falling edge bit-clock) or the 90% threshold (for a rising edge bit-clock).

APPLIES TO REVISION(S):

0.5, 0.6

15. 05000272 - Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V:

DESCRIPTION:
Data can become corrupted if data cache is enabled in write through mode and the AOW bit of the DCPLB is not set and Vddint is 0.9V or less.

WORKAROUND:
When Vddint <= 0.9V, either operate data cache in write back mode or set the AOW bit of the DCPLB when operating in write through mode. When Vddint is greater than 0.9V, the anomaly does not exist.

APPLIES TO REVISION(S):
0.5, 0.6

16. 05000273 - Writes to Synchronous SDRAM Memory May Be Lost:

DESCRIPTION:
When the Core Clock is not at least twice as fast as the the System Clock, 32-bit or wider writes to SDRAM memory may be lost. Note that since cache victims are effectively 256 bit wide writes, cache victimization will also trigger this anomaly.

WORKAROUND:
Either:
1) Make sure that the Core Clock (CCLK) is at least twice as fast as the System Clock (SCLK)
or
2) Make sure all external memory writes are 16 bits wide or less:

```
W[P2] = R0;   // 16-bit write
B[P2] = R0;   //  8-bit write
```

If using data cache, the Write Through policy should be used since there is no cache victimization in this mode.

APPLIES TO REVISION(S):
0.5

17. 05000276 - Timing Requirements Change for External Frame Sync PPI Modes with Non-Zero PPI_DELAY:

DESCRIPTION:
The PPI timing diagrams in the processor data sheet only apply to PPI modes where the PPI_DELAY register is set to zero.

WORKAROUND:
For non-zero values of the PPI_DELAY register, the following information applies:

In the data sheet, when POLC = 0, the frame sync is sampled on the falling edge of the PPI clock and the corresponding setup time is shown relative to this edge. When the PPI_DELAY register is a non-zero value, the frame sync setup time increases by one half the period of the PPI clock. The delay starts counting at the point on the existing diagrams where data is shown to be sampled.

In the data sheet, when POLC = 1, the frame sync is sampled on the rising edge of the PPI clock and the corresponding setup time is shown relative to this edge. When the PPI_DELAY register is a non-zero value, the frame sync setup time increases by one half the period of the PPI clock. The delay starts counting at the point on the existing diagrams where data is shown to be sampled.

APPLIES TO REVISION(S):
0.5, 0.6

18. 05000277 - Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt:

DESCRIPTION:
If a write to any I/O data register (data, clear, set and toggle registers) occurs one system clock cycle after an edge is detected on an edge-triggered interrupt, then the bit may be cleared one system clock cycle after it has been set.

If the bit has been programmed to generate an interrupt, then the interrupt will occur, but there will be no indication of which bit signalled the interrupt. The interrupt will be lost if the core clock is not running or if the SIC_IMASK bit is not set to enable the interrupt.

WORKAROUND:
If only one edge-sensitive source is assigned to one interrupt, it can be assumed to be the source of the interrupt and a read instruction of SIC_ISR and the I/O registers is not required. Note that all interrupts are properly executed, when enabled.

Use level-sensitive interrupts instead of edge-sensitive interrupts. Toggle the polarity between received edges to prevent re-entry of the interrupt service routine and to sensitize for the next edge. This is applicable when the latency between two edges is sufficient to serve the interrupt service routine or can be used for request lines. Toggling polarity can be used when looking for both edges. For only one edge, however, the other interrupt must be ignored.

APPLIES TO REVISION(S):
0.5

19. 05000278 - Disabling Peripherals with DMA Running May Cause DMA System Instability:

DESCRIPTION:
If a peripheral (PPI, SPORT, SPI, etc.) is disabled while DMA is running and before the associated DMA channel is disabled, the DMA system may be corrupted. In applications with multiple DMA channels running concurrently, this anomaly manifests itself with missing data or shuffled data being transferred. Although the anomaly also affects applications with a single DMA channel, its effects may not be visible if the peripheral is being shut down by the user code.

WORKAROUND:
If the DMA channel is running, disable the peripheral's associated DMA channel before disabling the peripheral itself.

If the DMA channel is stopped, the peripheral must be disabled before the associated DMA channel is disabled. When a channel is disabled, the DMA unit ignores the peripheral interrupt and passes it directly to the interrupt controller, thus generating unwanted interrupts.

APPLIES TO REVISION(S):
0.5

20. 05000281 - False Hardware Error when ISR Context Is Not Restored:

DESCRIPTION:
In some instances, exiting an interrupt service routine (ISR) without restoring context may be desired. Consider the following sequence:

```
ISR_Exit:
   RAISE 14;   // instruction A
   RTI;        // instruction B
```

This sequence will return from the current interrupt level and then immediately execute the level 14 interrupt service routine. Ideally, the latter would then restore the context before returning to user level, thus saving time in the first ISR.

In order to describe the problem, assume that the first interrupt occurs at an instruction like:

```
Rx = [Py];   // instruction C
```

or any similar instruction.

The processor will jump to the ISR (RETI will contain the address of instruction C). If the ISR changes Py, when the processor reaches instruction B above, it will speculatively fetch instruction C, which could now point to an invalid address. Because of instruction A, instruction B will not be executed, however, the hardware error condition will be latched. The hardware exception will then be triggered at the next system MMR read.

WORKAROUND:
Load the RETI register (before the above "`raise; rti;`" sequence) with a location where speculative fetches will not cause hardware errors.

APPLIES TO REVISION(S):
0.5

21. 05000282 - Memory DMA Corruption with 32-Bit Data and Traffic Control:

DESCRIPTION:
This anomaly applies to cases where:
1) Memory DMA (MDMA) channels are used in 32-bit mode (WDSIZE in MDMA_yy_CONFIG = 0b10).
and
2) Traffic Control is enabled to group accesses of the same direction together (DMA_TC_PER register contains non-zero fields).

In this particular case, high and low words may be inverted and/or interrupts may be lost.

WORKAROUND:
This anomaly is avoided if MDMA channels are used in 16-bit mode or if traffic control is disabled (DMA_TC_PER = 0x0000).

Note: on this device, the 16-bit MDMA is more efficient than the 32-bit mode for transfers from L1 to external memory and vice versa.

APPLIES TO REVISION(S):
0.5

22. 05000283 - System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage:

DESCRIPTION:

Consider the following sequence:

1) System MMR write is stalled.
2) Interrupt/Exception occurs while the System MMR write is stalled (thus killing the write).
3) Interrupt/Exception Service Routine performs an SSYNC instruction.

In order for this anomaly to happen, the change in program flow must kill the write in one particular stage of the execution pipeline. In this case, the anomaly will cause the MMR logic to think that the killed System MMR access is still valid. The SSYNC will therefore stall the processor indefinitely or until it is interrupted itself by a higher priority interrupt or event.

Similarly, if the System MMR write is killed by an instruction itself, such as a conditional branch, the infinite stall can happen if the store buffer is full and emptying out to slow external memory.

```
cc = r0 == r0;   // always true
if cc jump skip;
W[p0] = r1.l;    // System MMR access is fetched and killed
skip: ...
```

NOTE: if a user tries to halt the processor in the handler via the debugging tools, the infinite stall will also lock out the Emulation event.

WORKAROUND:

The workaround is to reset the MMR logic with another killed System MMR access that has no other side-effects on the application. For instance, read from the CHIPID register. The following code snippet, executed at the beginning of each interrupt/exception handler, will work around this anomaly:

```
cc = r0 == r0;   // always true
p0.h = 0xffc0;   // System MMR space CHIPID
p0.l = 0x0014;
if cc jump skip; // always skip MMR access, but MMR access is fetched and killed
r0 = [p0];       // bogus System MMR read to work around the anomaly
skip: ...        // continue with handler code
```

In the case of MMR writes being killed by the conditional branches, it is sufficient to insert 2 NOPs or any other non-MMR instructions in the location immediately after the conditional branch.

NOTE: in order to prevent lock-ups during debugging sessions, always set a breakpoint after the above code snippet if you need to halt the processor in the handler code.

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 (VisualDSP++, VDK, 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):

0.5

23. 05000288 - SPORTs May Receive Bad Data If FIFOs Fill Up:

DESCRIPTION:
The SPORT receives incorrect data if it is configured as follows:

1) The secondary receive data is enabled (RXSE=1) or the word length > 16 bits.
and
2) The RX FIFO is filled with 8 words of data.
and
3) An additional word is clocked into the SPORT.

In this case, the overflow does not assert because there is room to hold the data. The overflow will assert if the next piece of data is received without removing data from the FIFO.

This anomaly will cause one piece of primary data to be received in place of secondary data (RxSEC=1) or word swap (SLEN>0xF). Subsequent words will be received correctly.

WORKAROUND:
Avoid the conditions described in the problem description.

Operating so closely to a FIFO overflow should be avoided.

APPLIES TO REVISION(S):
0.5

24. 05000301 - Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space:

DESCRIPTION:
When MemDMA source and destination descriptors are in different memory spaces (one in internal memory and one in external memory), and if the traffic control is turned on, then the source descriptor count of descriptor words currently fetched can get corrupted by the value in the current destination descriptor count (which can be greater or less than the original source descriptor count). This will make the source fetch more/less descriptor elements than intended.

One possible result is that some elements of the descriptor may not be loaded. Another possible result is that extra descriptor element fetches may be performed. The descriptor element pointer may also overflow and wrap back to the start of the register set if too many extra fetches occur, thus overwriting good data with bad data in the first few registers (e.g., Next Descriptor Pointer). In this last case, the DMA may not appear to fail until the next descriptor fetch, when it fetches an invalid pointer.

WORKAROUND:
Place source and destination descriptors in the same memory space. Both should be located either in external or internal memory.

APPLIES TO REVISION(S):
0.5

25. 05000310 - 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.

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 (VisualDSP++, VDK, 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):
0.5, 0.6

26. 05000311 - Erroneous Flag (GPIO) Pin Operations under Specific Sequences:

DESCRIPTION:

When an access to a GPIO / Flag IO System MMR (any register with a PORTFIO_ or FIO_ prefix) is followed by another System MMR access that is not in the GPIO / Flag IO block, the GPIO's input driver can become active for a moment. As a result, the output values held in the Port F latch may clear erroneously, causing unwanted transitions in pin state on the output pins.

Only certain combinations of MMR accesses can trigger this failure, and they vary with the type of GPIO register access (i.e., write, read, aborted read). Some failures may occur very rarely and are unlikely to be detected during system evaluation. Because of this, it must be assumed that any MMR combination where the address bits 4, 5, or 6 differ between the GPIO register and the subsequently accessed MMR can generate this failure. Furthermore, the two MMR accesses need not occur in consecutive instructions for the problem to occur. The accesses can be separated by an unlimited number of cycles/instructions.

Accesses to multiple GPIO/FIO registers do not disturb each other, and GPIO flag pins configured as inputs are not impacted.

WORKAROUND:

Every sequence of accesses to GPIO registers must be terminated by a safe register read. A "safe register" is defined as any non-GPIO system MMR that has the same address bits 4, 5, and 6 as the last accessed GPIO register. The workaround must ensure this rule is not violated by non-linear program flow, such as conditional jumps or interrupts. As a welcomed side-effect, aborted GPIO MMR reads are avoided entirely. For example, the following sequence is safe:

```
P5.H = HI(PORTFIO);  P5.L = LO(PORTFIO);  /* PORTFIO is the same as FIO_FLAG_D */
P4.H = HI(SYSCR);    P4.L = LO(SYSCR);

CLI R7;           /* avoid interrupts */
NOP; NOP; NOP;    /* three cycles after CLI before 1st FIO read access  */

/* any GPIO sequence */
R6 = W[P5](Z);
R5 = W[P5+PORTFIO_MASKA-PORTFIO](Z);  /* PORTFIO_MASKA_D read */
R6 = R5 & R6;

W[P5+PORTFIO_CLEAR-PORTFIO] = R6;   /* last GPIO access to PORTFIO_CLEAR (0xFFC00704) */
R5 = W[P4](Z);                      /* dummy read from SYSCR (0xFFC00104) */
STI R7;           /* restore interrupts */
```

Note that address bits 4, 5 and 6 of SYSCR and PORTFIO_CLEAR are b#000. Therefore, a SYSCR read safely resolves the critical situation introduced by the PORTFIO_CLEAR access.

The following is a comprehensive list of safe registers for each GPIO/FIO register.

If the last GPIO access was to...	Then the "safe registers" are...
PORTFIO/_CLEAR/_SET/_TOGGLE (FIO_FLAG_D/C/S/T)	SYSCR, PPI_STATUS, or SPI_STAT
PORTFIO_MASKA/_CLEAR/_SET/_TOGGLE (FIO_MASKA_D/C/S/T)	UART_SCR, TIMER1_CONFIG, or EBIU_SDSTAT
PORTFIO_MASKB/_CLEAR/_SET/_TOGGLE (FIO_MASKB_D/C/S/T)	UART_GCTL, TIMER2_CONFIG, or DMA0_IRQ_STATUS
PORTFIO_DIR/POLAR/EDGE/BOTH (FIO_DIR/POLAR/EDGE/BOTH)	SPORT0_STAT, SPORT1_STAT, or DMA0_CURR_X_COUNT
PORTFIO_INEN (FIO_INEN)	TIMER_ENABLE, TIMER_STATUS, or DMA1_CONFIG

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 (VisualDSP++, VDK, 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):

0.5

27. 05000312 - Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted:

DESCRIPTION:
When instruction cache is enabled, invalid code may be executed when any of the following instructions are interrupted:

- **CSYNC**
- **SSYNC**
- **LCx =**
- **LTx =** (only when LCx is non-zero)
- **LBx =** (only when LCx is non-zero)

When this problem occurs, a variety of incorrect things could happen, including an illegal instruction exception. Additional errors could show up as an exception, a hardware error, or an instruction that is valid but different than the one that was expected.

WORKAROUND:
Place a **cli** before all **SSYNC**, **CSYNC**, "**LCx =**", "**LTx =**", and "**LBx =**" instructions to disable interrupts, and place an **sti** after each of these instructions to re-enable interrupts. When these instructions are executed in code that is already non-interruptible, the problem will not occur.

In an interrupt service routine that will enable interrupt nesting, be sure to push the LCx, LTx, and LBx registers before pushing RETI, which enables interrupt nesting. Following the inverse during the ISR context restore will guarantee that RETI is popped before the loop registers are loaded, thus disabling nested interrupts and protecting the loads from this anomaly situation. For example:

```
INT_HANDLER:
   [--sp] = astat;
   [--sp] = lc0; // push loop registers before pushing RETI
   [--sp] = lt0;
   [--sp] = lb0;
   [--sp] = lc1;
   [--sp] = lt1;
   [--sp] = lb1;
   [--sp] = reti; // push RETI to enable nested interrupts
   [--sp] = ...
      // body of interrupt handler
   ...  = [sp++];
   reti = [sp++]; // pop RETI to disable interrupts
   lb1 = [sp++];  // it is now safe to load the loop registers
   lt1 = [sp++];
   lc1 = [sp++];
   lb0 = [sp++];
   lt0 = [sp++];
   lc0 = [sp++];
   astat = [sp++];
```

Finally, as the workaround involves Supervisor Mode instructions to disable and enable interrupts, this does not apply to User Mode. In user space, do not use **CSYNC** or **SSYNC** instructions. Also, do not load the loop registers directly. Instead, utilize hardware loops which can be implemented with the **LSETUP** instruction, which limits loop ranges to 2046 bytes.

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 (VisualDSP++, VDK, 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):
0.5

28. 05000313 - PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes:

DESCRIPTION:
When the PPI is configured to trigger on a single external frame sync, all of the transfers require an edge on the frame sync except for the first transfer. For the first transfer only, the frame sync input is level-sensitive. This will make the PPI begin a transfer if the frame sync is at the active state, which can cause the PPI to start prematurely.

This anomaly does not apply when the PPI uses 2 or 3 frame syncs.

WORKAROUND:
When using a single external frame sync with the PPI, ensure that the frame sync is in the inactive state when the PPI is enabled.

APPLIES TO REVISION(S):
0.5

29. 05000315 - Killed System MMR Write Completes Erroneously on Next System MMR Access:

DESCRIPTION:
Consider the following sequence:
1) System MMR write is stalled.
2) Interrupt/Exception occurs while the System MMR write is stalled (thus killing the write).
3) Interrupt/Exception Service Routine accesses (either read or write) any system MMR.

In order for this anomaly to happen, the change in program flow must kill the write in one particular stage of the execution pipeline. In this case, the anomaly will cause the MMR logic to think that the killed System MMR access is still valid. The following access (read/write) to the System MMR in the handler will cause the previously stalled write to complete erroneously.

Similarly, if the System MMR write is killed by an instruction itself, such as a conditional branch, the erroneous write can happen if the store buffer is full and emptying out to slow external memory.

```
cc = r0 == r0;   // always true
if cc jump skip;
W[p0] = r1.l;    // System MMR access is fetched and killed
skip: ...
```

NOTE: if the processor is halted in the handler before the next System MMR access via the debugging tools, the processor will stall indefinitely waiting for the write to complete, thus locking out the Emulation event.

WORKAROUND:
The workaround is to reset the MMR logic with another killed System MMR access in the branch's shadow. For example, setting up a read from the System MMR CHIPID register and subsequently killing it will create a killed access that has no other side-effects on the system. Therefore, the following code snippet, executed at the beginning of each handler routine, will work around this anomaly:

```
cc = r0 == r0;   // always true
p0.h = 0xffc0;   // System MMR space CHIPID
p0.l = 0x0014;
if cc jump skip; // always skip System MMR access, but it is fetched and killed
r0 = [p0];       // bogus System MMR read to work around the anomaly
skip: ...        // continue with handler code
```

In the case of System MMR writes being killed by the conditional branches, it is sufficient to insert 2 NOPs or any other non-MMR instructions in the location immediately after the conditional branch.

NOTE: in order to prevent lock-ups during debug sessions, always insert a desired breakpoint *after* the above code snippet if you need to halt the processor in the handler.

APPLIES TO REVISION(S):
0.5

30. 05000319 - Internal Voltage Regulator Values of 1.05V, 1.10V and 1.15V Not Allowed for LQFP Packages:

DESCRIPTION:
When the VR_CTL register is programmed to contain VLEV values of 0xA, 0xB, or 0xC (1.05V, 1.10V, and 1.15V, respectively), the actual Vddint applied to the core through the regulator may drop below the specified tolerance of -5%.

This issue only occurs on parts in LQFP packages.

WORKAROUND:
Either avoid programming these values or program the regulator to the next highest setting to ensure that the Vddint remains above the minimum threshold for the core clock that the application is running.

APPLIES TO REVISION(S):
0.5

31. 05000357 - Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled:

DESCRIPTION:
When configured in multi-channel mode with channel 0 disabled, DMA transmit data will be sent to the wrong SPORT channel if all of the following criteria are met:

1) External Receive Frame Sync (IRFS = 0 in SPORTx_RCR1)
2) Window Offset = 0 (WOFF = 0 in SPORTx_MCMC1)
3) Multichannel Frame Delay = 0 (MFD = 0 in SPORTx_MCMC2)
4) DMA Transmit Packing Disabled (MCDTXPE = 0 in SPORTx_MCMC2)

When this specific configuration is used, the multi-channel transmit data gets corrupted because whatever is in the channel 0 placeholder in non-packed mode gets sent first, even though channel 0 is disabled. The result is a one-word data shift in the output window, which repeats for each subsequent window in the serial stream. For example, if the non-packed transmit buffer is {0, 1, 2, 3, 4, 5, 6, 7}, and the window size is 8 channels with channel 0 disabled and channels 1-7 enabled to transmit, the expected data sequence in a series of output windows is:

1234567--1234567--1234567--1234567

With this anomaly, the output looks like this instead:

0123456--7012345--6701234--5670123

WORKAROUND:
There are several possible workarounds to this:

1) Disable Multichannel Mode
2) Use Internal Receive Frame Syncs
3) Use a Multichannel Frame Delay > 0
4) Use a Window Offset > 0
5) Enable DMA Transmit Packing
6) Do not disable Channel 0

APPLIES TO REVISION(S):
0.5

32. 05000366 - PPI Underflow Error Goes Undetected in ITU-R 656 Mode:

DESCRIPTION:
If the PPI port is configured in ITU-R 656 Output Mode, the FIFO Underrun bit (UNDR in PPI_STATUS) does not get set when a PPI FIFO underrun occurs. An underrun can happen due to limited bandwidth or the PPI DMA failing to gain access to the bus due to arbitration latencies.

WORKAROUND:
None.

APPLIES TO REVISION(S):
0.5, 0.6

33. 05000371 - Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration:

DESCRIPTION:
The RTS instruction can fail to return correctly if placed within four execution cycles of the beginning of a subroutine. For example:

```
CALL STUB_CODE;
...
...
...
STUB_CODE:
   RTS;
```

When this happens, potential bit failures in RETS will cause the processor to vector to the wrong address, which can cause invalid code to be executed.

WORKAROUND:
If there are at least four execution cycles in the subroutine before the RTS, the CALL and RTS instructions can never align in the manner required to encounter this problem. Since a NOP is a 1-cycle instruction, the following is a safe workaround for all potential failure cases:

```
CALL STUB_CODE;
...
...
...
STUB_CODE:
   NOP;      // These 4 NOPs can be any combination of instructions
   NOP;      // that results in at least 4 core clock cycles.
   NOP;
   NOP;
   RTS;
```

Branch prediction does not factor into this scenario. Conditional jumps within the subroutine that arrive at the RTS instruction inside of 4 cycles will not result in the scenario required to cause this failure. Asynchronous events (interrupts, exceptions, and NMI) are also not susceptible to this failure.

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 (VisualDSP++, VDK, 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):
0.5

34. 05000400 - PPI Does Not Start Properly In Specific Mode:

DESCRIPTION:
When the PPI port is configured in transmit mode with two internal frame syncs, the PPI will not start properly if the PPI Frame Sync 3 (PPI_FS3) pin is left floating.

WORKAROUND:
The PPI_FS3 pin must be pulled down when the PPI is configured in transmit mode with 2 internal frame syncs.

APPLIES TO REVISION(S):
0.5

35. 05000402 - SSYNC Stalls Processor when Executed from Non-Cacheable Memory:

DESCRIPTION:
Executing an SSYNC instruction from non-cacheable L2 memory with interrupts disabled can cause the processor to stall.

WORKAROUND:
If any interrupts are enabled, the stall will still occur, but it will be broken by the asynchronous event. If no interrupts are enabled or no interrupts are being generated, the stall is indefinite and the processor must be reset.

To avoid the stall condition, the following conditions must be met.

1) The SSYNC is in L1 memory or in cacheable L2 memory.
2) The SSYNC is not at a loop bottom where the loop top is located in non-cacheable L2 memory.
3) If the SSYNC is located in a cacheable L2 page, it is at least eight 64-bit words away from the bottom of the page (as specified by a CPLB) if the following (address sequential) page is either L1 or non-cacheable L2 memory.

If any of the above conditions is not met, another workaround would be to configure one of the timers prior to the SSYNC instruction with a time-out period to generate an interrupt and break the stall.

APPLIES TO REVISION(S):
0.5

36. 05000403 - Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall:

DESCRIPTION:
When level-sensitive GPIO events are used to wake the processor from the low-power sleep mode of operation, the processor may stall indefinitely if the width of the wakeup pulse is too short. When this occurs, the PLL begins transitioning from the sleep mode due to the level sensed on the GPIO pin, but then reverts back to the sleep mode if the trigger level is removed before the core has had sufficient time to break the idle state to resume execution.

As a result, the processor does not wake up properly, at which point only a hardware reset can exit the resulting stall condition.

WORKAROUND:
There are two ways to avoid this anomaly:

1) Use edge-sensitivity for the pin(s) being used to generate the wakeup event.
2) Ensure that the edge on the wakeup signal is clean and held at the trigger level for at least 3 system clock (SCLK) cycles.

APPLIES TO REVISION(S):
0.5, 0.6

37. 05000416 - 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):
0.5, 0.6

38. 05000425 - Multichannel SPORT Channel Misalignment Under Specific Configuration:

DESCRIPTION:
When using the Serial Port in Multi-Channel Mode, the transmit and receive channels can get misaligned if a very specific configuration for the SPORT is met, as follows:

1) Window Offset (WOFF) = 0.
2) Window Size is an odd multiple of 8 (i.e., WSIZE is an even number > 0).
3) The time between RFS pulses is exactly equal to the window duration.

Note: The anomaly does NOT apply when WSIZE = 0.

When this exact configuration is used, the multi-channel mode channel enable registers are mislatched after the first window concludes, which results in the TDV signal being driven according to incorrect channel assignments and receive data being sampled on the wrong channels. So, the first window will send and receive properly, but all windows after the first will be misaligned, and data sent and received will be corrupted.

This error occurs for external and internal clocks and RFS.

WORKAROUND:
There are several workarounds possible:

1) Use a window offset other than 0.
2) Use a window size that is an even multiple of 8.
3) For internal RFS, make sure that SPORTx_RFSDIV is at least equal to the window size (# of enabled channels * SLEN).

APPLIES TO REVISION(S):
0.5, 0.6

39. 05000426 - 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;
```

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 (VisualDSP++, VDK, 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):

0.5, 0.6

40. 05000443 - 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
```

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 (VisualDSP++, VDK, 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):
0.5, 0.6

41. 05000461 - 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):

0.5, 0.6

42. 05000462 - Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign:

DESCRIPTION:
When the SPORT is configured in multichannel mode with an external SPORT clock, a synchronization problem may occur when the SPORT is enabled. This synchronization issue manifests when the skew between the external SPORT clock and the Blackfin processor's internal System Clock (SCLK) causes the channel counters inside the SPORT to get out-of-sync. When this occurs, a "dead" channel is inserted at the beginning of the window, and the rest of the transmit channels are right-shifted one location throughout the active window. The last channel data will be sent as the first enabled transmit channel data in the second window after another "dead" channel is inserted. All data will be sent sequentially and in its entirety, but it is transmitted on the wrong channels with respect to the frame sync and will never recover.

WORKAROUND:
When this error occurs, the SPORT must be restarted and checked again for this error. The failure is extremely rare to begin with, so the probability of seeing consecutive restarts showing the failure is infinitesimally small.

A software solution is possible based on the timing of the SPORT interrupt. In the SPORT ISR, the CYCLES register can be set to zero the first time the interrupt occurs and then read back the second time the interrupt occurs. This will provide a time reference in core clocks for the frequency of the SPORT interrupt itself. If the value read the second time exceeds the duration of the multichannel window (in core clocks), then a "dead" channel was inserted into the stream, and the SPORT must be restarted.

Hardware workarounds are going to be heavily dependent on how the multichannel mode SPORT is configured. In multichannel mode, TFS functions as a Transmit Data Valid (TDV) signal and will always be driven to the active state (as governed by the LTFS bit in the SPORTx_TCR1 register) during transmit channels. Therefore, the TDV signal can be routed to one of the GPIO pins configured to generate an interrupt upon detection of the TDV pin changing states, based upon how the application configures the channels within the active frame, to detect the "dead" channel. If all the channels in the window are configured as transmit channels and there is no window offset and no multichannel frame delay, then TDV should go active as soon as the RFS pulse is received. If the period of the RFS pulse is exactly the window size (i.e., there are no extra clocks after the active window before the next RFS is detected), then TDV will remain active throughout operation. Therefore, if TDV goes inactive while the SPORT is on, the failure happened and the SPORT must be restarted and run again with this test in place until the failure is not detected.

For applications that have a window offset, a multichannel frame delay, extra clocks between the end of the active window and the next frame sync, and/or non-transmit channels inside the active window, the first TDV assertion would need to be tracked manually to detect the "dead" channel. One idea might be to do the following:

1) Connect TFS (TDV) to a GPIO interrupt and configure the interrupt to occur when TDV goes active.
2) Connect RFS to a GPIO interrupt and configure the interrupt to occur when RFS goes active.
3) Connect the SPORT receive clock to a TMRx pin configured in EXT_CLK mode.

When the GPIO interrupt for the active RFS pulse signifying the start of the window occurs, enable the Timer that is being used to track the SPORT receive clock. When the GPIO interrupt for the TDV signal transition occurs, check the TIMERx_COUNTER register to determine how many SPORT clocks have passed since the frame started. If it is one channel's worth over the expected value, the error occurred and the SPORT must be restarted and tested again. The GPIO interrupts should also be disabled if the startup condition is not detected.

APPLIES TO REVISION(S):
0.5, 0.6

43. 05000471 - Boot Failure When SDRAM Control Signals Toggle Coming Out Of Reset:

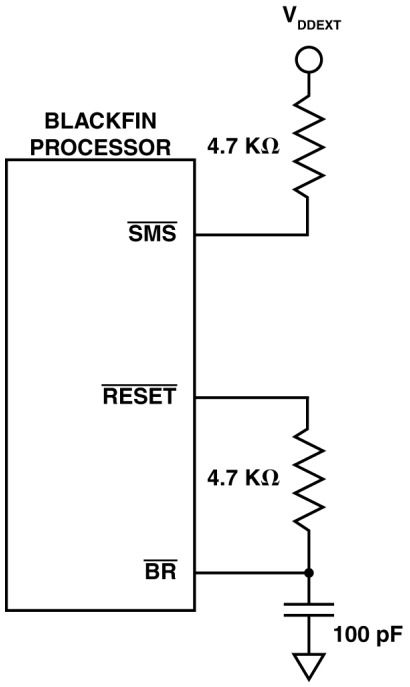
DESCRIPTION:
When $\overline{\text{RESET}}$ is de-asserted at the conclusion of the power-on reset sequence (not applicable to warm resets), there is a very small chance that the SDRAM control signals ($\overline{\text{SRAS}}$, $\overline{\text{SCAS}}$, $\overline{\text{SWE}}$, $\overline{\text{SMS}}$, SCKE, and SA10) may be driven low for a single CLKOUT cycle. This activity can cause certain SDRAM devices to enter engineering test modes that result in the SDRAM driving data onto the shared EBIU data lines that the Blackfin processor is going to read from when configured to boot or execute from parallel flash.

When booting from parallel flash (BMODE = b#01), this contention corrupts the boot stream and causes the processor to hang. At this point, power-cycling the processor is required to clear the condition.

When executing from parallel flash (BMODE = b#00), this contention corrupts the op-codes and causes the processor to execute incorrect or illegal instructions, which may result in improper application execution or unexpected exceptions.

This anomaly does not apply when booting from serial memory, as the EBIU is not used by the boot process for SPI boot modes until code or data needs to be resolved to SDRAM. Since the first write command issued by the SDRAM controller causes the SDRAM to exit the engineering test mode, no data corruption occurs.

WORKAROUND:
A hardware workaround can be implemented in the form of an RC delay circuit introduced between the $\overline{\text{RESET}}$ and bus request ($\overline{\text{BR}}$) pins:



The delayed version of $\overline{\text{RESET}}$ seen on $\overline{\text{BR}}$ causes the Blackfin processor to continue to three-state the SDRAM control signals after the rising edge of $\overline{\text{RESET}}$ for longer than one CLKOUT cycle, which over-rides the glitch behavior described by this anomaly. The pull-up resistor on the $\overline{\text{SMS}}$ pin ensures that the memory-select signal remains de-asserted while the processor is three-stating the $\overline{\text{SMS}}$ output. Additional pull-ups are needed on the $\overline{\text{AMS}}$x signals as well. This drawing contains the local connections to the Blackfin processor only. The $\overline{\text{RESET}}$ input pin is connected to the reset supervisor circuit, and the $\overline{\text{SMS}}$ output pin connects to the SDRAM memory select pin.

APPLIES TO REVISION(S):
0.5, 0.6

44. 05000473 - Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15:

DESCRIPTION:
A SPORT receive underflow error can be erroneously triggered when the SPORT serial length is greater than 16 bits and an interrupt occurs as the access is initiated to the 32-bit SPORTx_RX register. Internally, two accesses are required to obtain the 32-bit data over the internal 16-bit Peripheral Access Bus, and the anomaly manifests when the first half of the access is initiated but the second is held off due to the interrupt. Application code vectors to service the interrupt and then issues the read of the SPORTx_RX register again when it subsequently resumes execution after the interrupt has been serviced. The previous read that was interrupted is still pending awaiting the second half of the 32-bit access, but the SPORT erroneously sends out two requests again. The first access completes the previous transaction, and the second access generates the underflow error, as it is now attempting to make a read when there is no new data present.

WORKAROUND:
The anomaly does not apply when using valid serial lengths up to 16 bits, so setting SLEN < 16 is one workaround.

When the length of the serial word is 17-32 bits (16 <= SLEN < 32), accesses to the SPORTx_RX register must not be interrupted, so interrupts must be disabled around the read. In C:

```
int temp_IMASK;

temp_IMASK = cli();
RX_Data = *pSPORT0_RX;
sti(temp_IMASK);
```

In assembly:

```
P0.H = HI(SPORT0_RX);
P0.L = LO(SPORT0_RX);

CLI R0;
R1 = [P0];
STI R0;
```

APPLIES TO REVISION(S):
0.5, 0.6

45. 05000475 - Possible Lockup Condition when Modifying PLL from External Memory:

DESCRIPTION:
Synchronization logic in the EBIU can get corrupted if PLL alterations are made by code that resides in external memory. When this occurs, an infinite stall will occur, and the part will need to be reset. The lockup is dependent on what the original ratio was, what the new ratio is, and other factors, thus making it impossible to specify any cases where this is safe.

WORKAROUND:
The CCLK::SCLK ratio should not be changed via the external interface, whether it's from asynchronous memory or SDRAM. Only make modifications to the PLL_CTL and PLL_DIV registers from code executing in on-chip memory.

APPLIES TO REVISION(S):
0.5, 0.6

46. 05000477 - 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.

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.

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 (VisualDSP++, VDK, 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):
0.5, 0.6

47. 05000481 - Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption:

DESCRIPTION:
Reading the ITEST_COMMAND or ITEST_DATA registers will erroneously trigger a write to these registers in addition to reading the current contents of the register. The erroneous write does not update the read state of the register, however, the data written to the register is acquired from the most recent MMR write request, whether the most recent MMR write request was committed or speculatively executed. The bogus write can set either register to perform unwanted operations that could result in:

1) Corrupted instruction L1 memory and/or instruction TAG memory.
and/or
2) Garbled instruction fetch stream (stale data used in place of new fetch data).

WORKAROUND:
Never read ITEST_COMMAND or ITEST_DATA. The only exception to this strict workaround is in the case of performing the read atomically and immediately after a write to the same register. In this case, the erroneous write will still occur, but it will be with the exact same data as the intentional write that preceded it.

APPLIES TO REVISION(S):
0.5, 0.6

48. 05000489 - PLL May Latch Incorrect Values Coming Out of Reset:

DESCRIPTION:
It is possible that the PLL can latch incorrect SSEL and CSEL values during reset when VDDINT is powered before VDDEXT. If this problem occurs, the PLL_DIV register will show the correct default value when read via software, but the actual SSEL and CSEL values being provided to the PLL may be incorrect. This results in different values for the core and system clocks from what the default values would be coming out of reset. If this problem occurs, the most likely result will be system and core clocks that are not the default (CCLK = 10xCLKIN, SCLK = 2xCLKIN), which will be corrected when the application programs the PLL to the desired frequencies. However, the random nature of the values latched could lead to the PLL getting illegally programmed, which can cause the boot process to fail.

WORKAROUND:
There are a few workarounds for this issue. Any one of the following will avoid the issue:

1) Use the on-chip regulator.
2) Issue a second hardware reset after the power-on reset.
3) Ensure that VDDEXT reaches at least the Vddext minimum specification before turning on VDDINT.
4) If powering VDDINT first, keep $\overline{\text{RESET}}$ de-asserted until after VDDEXT has been established, then assert $\overline{\text{RESET}}$ per the power-on reset specification.

It is extremely unlikely that this anomaly will occur. If it has not been observed in existing designs, it is recommended that one of the above workarounds be implemented at the next logical point of the design cycle. For systems in development, implementing one of the above workarounds is strongly encouraged.

APPLIES TO REVISION(S):
0.5, 0.6

49. 05000491 - 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):
0.5, 0.6

50. 05000494 - EXCPT Instruction May Be Lost If NMI Happens Simultaneously:

DESCRIPTION:

A software exception raised by issuing the EXCPT instruction may be lost if an NMI event occurs simultaneous to execution of the EXCPT instruction. When this precise timing is met, the program sequencer believes it is going to service the EXCPT instruction and prepares to write the address of the next sequential instruction after the EXCPT instruction to the RETX register. However, the NMI event takes priority over the Exception event, and this address erroneously goes to the RETN register. As such, when the NMI event is serviced, program execution incorrectly resumes at the instruction after the EXCPT instruction rather than at the EXCPT instruction itself, so the software exception is lost and is not recoverable.

WORKAROUND:

Either do not use NMI or protect against this lost exception by forcing the exception to be continuously re-raised and verified in the exception handler itself. For example:

```
EXCPT 0;
JUMP -2;  // add this jump -2 after every EXCPT instruction
```

Then, in the exception handler code, read the EXCAUSE field of the SEQSTAT register to determine the cause of the exception.
If EXCAUSE < 16, the handler was invoked by execution of the EXCPT instruction, so the RETX register must then be modified to skip over the JUMP -2 that was inserted in the workaround code:

```
R2 = SEQSTAT;
R2 <<= 0x1A;
R2 >>= 0x1A;   // Mask Everything Except SEQSTAT[5:0] (EXCAUSE)
R1 = 0xF (Z);
CC = R2 <= R1; // Check for EXCAUSE < 16
IF !CC JUMP CONTINUE_EX_HANDLER;
R2 = RETX;
R2 += 2;       // Modify RETX to Point to Instruction After Inserted JUMP -2;
RETX = R2;
JUMP END_EX_HANDLER;

CONTINUE_EX_HANDLER: // Rest of Exception Handler Code Goes Here
   .
   .
   .
END_EX_HANDLER: RTX;
```

In this fashion, the JUMP -2 guarantees that the soft exception is re-raised when this anomaly occurs. When the NMI does not occur, the above exception handler will redirect the application code to resume after the JUMP -2 workaround code that re-raises the exception.

A 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 (VisualDSP++, VDK, 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):

0.5, 0.6

51. 05000501 - RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes:

DESCRIPTION:
When in SPI receive DMA modes, the RXS bit in SPI_STAT can get set and erroneously get stuck high if the SPI port is disabled as hardware is updating the status of the RXS bit. When in RX DMA mode, RXS will set as a word is transferred from the shift register to the internal FIFO, but it is then automatically cleared immediately by the hardware as DMA drains the FIFO. However, there is an internal 2 system clock (SCLK) latency for the status register to properly reflect this. If software disables the SPI port in exactly this window of time before RXS is cleared, the RXS bit doesn't get cleared and will remain set, even after the SPI is disabled. If the SPI port is subsequently re-enabled, the set RXS bit will cause one of two problems to occur:

1) If enabled in core RX mode, the SPI RX interrupt request will be raised immediately even though there is no new data in the SPI_RDBR register.
2) If enabled in RX DMA mode, DMA requests will be issued, which will cause the processor to DMA data from the SPI FIFO even though there is actually no new data present.

In master mode, the SPI will continue issuing clocks after RX DMA is completed until the SPI port is disabled. If any SPI word is received exactly as software disables the SPI port, the problem will occur.

In slave mode, the host would have to continue providing clocks and the chip-select for this possibility to occur.

WORKAROUND:
Reading the SPI_RDBR register while the SPI is disabled will clear the stuck RXS condition and not trigger any other activity. If using RX DMA mode, be sure to include this dummy read after the SPI port disable.

APPLIES TO REVISION(S):
0.5, 0.6

52. 05000503 - SPORT Sign-Extension May Not Work:

DESCRIPTION:
In multichannel receive mode, the SPORT sign-extension feature (RDTPYE=b#01 in SPORTx_RCR1) is not reliable for channel 0 data when configured for MSB-first data reception. This is regardless of any channel offset and/or multichannel frame delay.

WORKAROUND:
1) If possible, use receive bit order of LSB-first.
2) Do not use channel 0.
3) Ignore channel 0 data.
4) Use software to manually apply sign extension to the channel 0 data before processing.

APPLIES TO REVISION(S):
0.5, 0.6

53. 05000506 - Hardware Loop Can Underflow Under Specific Conditions:

DESCRIPTION:
When two consecutive hardware loops are separated by a single instruction, and the two hardware loops use the same loop registers, and the first loop contains a conditional jump to its loop bottom, the first hardware loop can underflow. For example:

```
P0 = 16;
LSETUP(loop_top1, loop_bottom1), LC0 = P0;
   loop_top1:    nop;
                 if CC JUMP loop_bottom1;
                 nop;
                 nop;
   loop_bottom1: nop;

nop;                          // Any single instruction

LSETUP(loop_top2, loop_bottom2), LC0 = P0;
   loop_top2:    nop;
   loop_bottom2: nop;
```

If a stall occurs on the instruction that is between the two loops, the top loop can decrement its loop count from 0 to 0xFFFFFFFF and continue looping with the incorrect loop count.

WORKAROUND:
There are several workarounds to this issue:

1) Do not use the same loop register set in consecutive hardware loops.
2) Ensure there is not exactly one instruction between consecutive hardware loops.
3) Ensure the first loop does not conditionally jump to its loop bottom.

APPLIES TO REVISION(S):
0.5, 0.6