ANALOG DEVICES

Blackfin Embedded Processor

Silicon Anomaly List — ADSP-BF542/BF544/BF547/BF548/BF549

ABOUT ADSP-BF542/BF544/BF547/BF548/BF549 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF542/BF544/BF547/BF548/BF549 product(s) and the functionality specified in the ADSP-BF542/BF544/BF547/BF548/BF549 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. If an 'M' suffix is affixed to the silicon revision, this indicates that the device contains a mobile DDR SDRAM controller instead of a standard DDR controller and a standard DDR controller is not available. If the 'M' suffix is not present, the silicon revision contains the standard DDR controller and is not available with a mobile DDR controller. 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>
0.3M*	0x0003
0.2	0x0002
0.1	0x0001

* - M = Mobile DDR controller instead of standard DDR controller

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
06/03/2010	J	C	Added Anomalies: 05000220, 05000473, 05000474, 05000477, 05000481, 05000483, 05000484, 05000485 Deleted Anomalies: 05000383
07/23/2009	I	B	Removed Silicon Revision 0.0 Added Anomalies: 05000434, 05000456, 05000457, 05000460, 05000461, 05000462, 05000463, 05000464, 05000465, 05000466, 05000467 Revised Anomalies: 05000365
01/16/2009	H	A	Added Silicon Revision 0.3M Added Anomalies: 05000431, 05000442, 05000443, 05000446, 05000447, 05000448, 05000449, 05000450 Deleted Anomalies: 05000377
08/07/2008	G	PrH	Added Silicon Revision 0.2 Added Anomalies: 05000416, 05000425, 05000426, 05000427, 05000429, 05000430 Revised Anomalies: 05000325, 05000365 Deleted Anomalies: 05000412 (covered by 05000325)

NR003403J

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-BF542/BF544/BF547/BF548/BF549 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.1	0.2	0.3 M
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x	x
2	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x	x
3	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x	x
4	05000220	Data Corruption/Core Hang with L2/L3 Configured in Writeback Cache Mode	x	x	x
5	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x	x
6	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x	x
7	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x	x
8	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x	x
9	05000325	FIFO Boot Mode Not Functional	x	.	.
10	05000353	bfrom_SysControl() Firmware Function Performs Improper System Reset	x	.	.
11	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	x	x
12	05000360	External Memory Read Access Hangs Core With PLL Bypass	x	x	x
13	05000365	DMAs that Go Urgent during Tight Core Writes to External Memory Are Blocked	x	x	x
14	05000369	Addressing Conflict between Boot ROM and Asynchronous Memory	x	x	x
15	05000371	Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration	x	.	.
16	05000378	Security/Authentication Speedpath Causes Authentication To Fail To Initiate	x	.	.
17	05000379	16-Bit NAND FLASH Boot Mode Is Not Functional	x	.	.
18	05000404	Lockbox SESR Disallows Certain User Interrupts	x	.	.
19	05000405	Lockbox SESR Firmware Does Not Save/Restore Full Context	x	x	x
20	05000406	Lockbox SESR Argument Checking Does Not Check L2 Memory Protection Range	x	.	.
21	05000407	Lockbox SESR Firmware Arguments Are Not Retained After First Initialization	x	.	.
22	05000408	Lockbox Firmware Memory Cleanup Routine Does not Clear Registers	x	x	x
23	05000409	Lockbox firmware leaves MDMA0 channel enabled	x	.	.
24	05000411	bfrom_SysControl() Firmware Function Cannot be Used to Enter Power Saving Modes	x	.	.
25	05000413	NAND Boot Mode Not Compatible With Some NAND Flash Devices	x	.	.
26	05000414	OTP_CHECK_FOR_PREV_WRITE Bit is Not Functional in bfrom_OtpWrite() API	x	.	.
27	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x	x
28	05000425	Multichannel SPORT Channel Misalignment Under Specific Configuration	x	x	x
29	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x	x
30	05000427	CORE_EPPI_PRIO bit and SYS_EPPI_PRIO bit in the HMDMA1_CONTROL register are not functional	x	.	.
31	05000429	WB_EDGE Bit in NFC_IRQSTAT Incorrectly Reflects Buffer Status Instead of IRQ Status	x	.	.
32	05000430	Software System Reset Corrupts PLL_LOCKCNT Register	.	x	.
33	05000431	Incorrect Use of Stack in Lockbox Firmware During Authentication	x	x	.
34	05000434	SW Breakpoints Ignored Upon Return From Lockbox Authentication	x	x	x
35	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x	x
36	05000446	CDMAPRIO and L2DMAPRIO Bits in the SYSCR Register Are Not Functional	x	x	x
37	05000447	UART IrDA Receiver Fails on Extended Bit Pulses	x	x	x
38	05000448	DDR Clock Duty Cycle Spec Violation (tCH, tCL)	x	.	.
39	05000449	Reduced Timing Margins on DDR Output Setup and Hold (tDS and tDH)	x	.	.
40	05000450	USB DMA Short Packet Data Corruption	x	x	x
41	05000456	USB Receive Interrupt Is Not Generated in DMA Mode 1	x	x	x

No.	ID	Description	0.1	0.2	0.3 M
42	05000457	Host DMA Port Responds to Certain Bus Activity Without $\overline{\text{HOST\_CE}}$ Assertion	x	x	x
43	05000460	USB DMA Mode 1 Failure When Multiple USB DMA Channels Are Concurrently Enabled	x	x	x
44	05000461	False Hardware Error when RETI Points to Invalid Memory	x	x	x
45	05000462	Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign	x	x	x
46	05000463	USB DMA RX Data Corruption	x	x	x
47	05000464	USB TX DMA Hang	x	x	x
48	05000465	USB Rx DMA Hang	x	x	x
49	05000466	TxPktRdy Bit Not Set for Transmit Endpoint When Core and DMA Access USB Endpoint FIFOs Simultaneously	x	x	x
50	05000467	Possible USB RX Data Corruption When Control & Data EP FIFOs are Accessed via the Core	x	x	x
51	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x	x	x
52	05000474	Access to DDR SDRAM Causes System Hang with Certain PLL Settings	x	x	x
53	05000477	TESTSET Instruction Cannot Be Interrupted	x	x	x
54	05000481	Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption	x	x	x
55	05000483	Possible USB Data Corruption When Multiple Endpoints Are Accessed by the Core	x	x	x
56	05000484	DDR Trim May Not Be Performed for Certain VLEV Values in OTP Page PBS00L	x	x	.
57	05000485	PLL_CTL Change Using bfrom_SysControl( ) Can Result in Processor Overclocking	.	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-BF542/BF544/BF547/BF548/BF549 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, 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):
0.1, 0.2, 0.3M

2. 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):
0.1, 0.2, 0.3M

3. 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, 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):
0.1, 0.2, 0.3M

4. 05000220 - Data Corruption/Core Hang with L2/L3 Configured in Writeback Cache Mode:

DESCRIPTION:
This problem occurs with either on-chip L2 or external L3 memory configured in Write-Back Cache mode while the other level of memory is either cached (Write-Back or Write-Through) or uncached.

Assuming L3 to be in Write-Back cache mode and L2 either cached or uncached, a specific sequence of events can result in either data corruption in memory or a core hang. The trigger for these potential failures is when the cache is writing back to L3 memory, but the write is held off due to an SDRAM row change or other activity on the External Memory Interface, such as a DMA access. If this scenario occurs and the core:

1) issues a write to L2 during the hold-off, both L2 and L3 memory will have corrupted data.

2) issues a read from L2 during the hold-off, the read never completes, resulting in an infinite core hang.

The same failures occur with the opposite configuration as well. If the cache is writing back to on-chip L2 memory, and the core attempts an access (read or write) to L3 memory, then the failures persist, albeit less frequently (due to the higher speed of accesses to L2 memory, it is more difficult to stall the write-backs).

WORKAROUND:
If either L2 or L3 accesses are restricted to DMA, and Write-Back cache is used for the other level of memory, then the failure is avoided. Failures also do not occur if there are no core accesses to one level of memory when the other level is cached.

Possible workarounds are:

1) Use Write-Through Cache Mode.

2) If the necessary accesses are infrequent and to a limited number of data locations, insert an SSYNC before the access. This will ensure that all pending cache writes have completed.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

5. 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, 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):

0.1, 0.2, 0.3M

6. 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 I2S/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):

0.1, 0.2, 0.3M

7. 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 errata does not exist.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

8. 05000310 - False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory:

DESCRIPTION:
Fetches at the boundary of either reserved memory or L1 Instruction cache memory (if instruction cache enabled) which is covered by a valid CPLB cause a false Hardware Error (External Memory Addressing Error).

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

Note that this anomaly also happens on the boundary of L1_code_cache if instruction cache is enabled.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

9. 05000325 - FIFO Boot Mode Not Functional:

DESCRIPTION:
The FIFO boot mode is not functional.

WORKAROUND:
Use one of the other boot modes.

APPLIES TO REVISION(S):
0.1

10. 05000353 - bfrom_SysControl() Firmware Function Performs Improper System Reset:

DESCRIPTION:
The *bfrom_SysControl()* firmware function does not properly execute a software reset and cannot be used to perform system reset of the processor. Consequently, not all System MMRs are reset correctly and the watchdog reset may fail.

WORKAROUND:
Use the following code to perform system reset instead of *bfrom_SysControl(SYSCTRL_SYSRESET)*. This code must execute out of non-cache L1 memory:

```
/* Issue system soft reset */
P0.L = LO(SWRST) ;
P0.H = HI(SWRST) ;
R0.L = 0x0007 ;
W[P0] = R0 ;
SSYNC ;

/* Wait for System reset to complete (needs to be 5 SCLKs).     */
/* Assuming a worst case CCLK:SCLK ratio (15:1), use 5*15 = 75  */
/* as the loop count.                                           */
P1 = 75;
LSETUP(start, end) LCO = P1 ;
   start:
   end:
      NOP ;

/* Clear system soft reset */
R0.L = 0x0000 ;
W[P0] = R0 ;
SSYNC ;

/* Core reset - forces reboot */
RAISE 1 ;
```

APPLIES TO REVISION(S):
0.1

11. 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):
0.1, 0.2, 0.3M

12. 05000360 - External Memory Read Access Hangs Core With PLL Bypass:

DESCRIPTION:
Core will hang if processor is placed in PLL bypass mode and a read operation is performed to an external memory location. This also includes fetches made to the boot ROM.

WORKAROUND:
Implement one of the following:
1) Do not bypass PLL. If CCLK = SCLK is desired, program through CSEL and SSEL.

2) After bypassing PLL, write 0x0001 to PLL_DIV register before accessing external memory.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

13. 05000365 - DMAs that Go Urgent during Tight Core Writes to External Memory Are Blocked:

DESCRIPTION:
A core request to the DDR controller can override a higher priority "Urgent DMA*" request. However, this happens only if the core is continually issuing writes to the external memory (i.e., in a tight loop), and if the peripheral is running at high speeds (close to SCLK/2) or if the instantaneous bandwidth during the run time of an application approaches the theoretical bandwidth limitations.

If the DMA Sync bit (DMAx_CONFIG[5]) is set, then this behavior can be seen even at very low peripheral speeds.

*An urgent DMA request is issued when a DMA FIFO is empty (If DMA is configured as TX) or if the FIFO is full(If DMA is configured as RX) and a peripheral requests for a DMA-FIFO slot. In case of EPPIx, the "urgent" request is issued based on the EPPI FIFO levels which can be programmed using the EPPIx_CONTROL register(FIFO_UWM bits - EPPIx_CONTROL[30:29]).

WORKAROUND:
If running peripherals at high speeds with peripheral DMA accessing DDR SDRAM memory, then avoid tight core accesses to the DDR SDRAM memory.

If the DMA Sync bit is enabled, then set the appropriate DEBx_URGENT bits in the EBIU_DDRQUE register. This will make all DMA requests coming from the DMA controller associated with DEBx urgent. The disadvantage of the workaround is that it adversely affects the throughput of the system. By making every DMA request urgent, the bus is accessed more often by the DMA controller that the peripheral is connected to, resulting in fewer accesses by the core and the other DMA controller(s). Cache flushes/misses may also be held off.

If using EPPI, then one can use the CORE_EPPI_PRIO and SYS_EPPI_PRIO bit in the HMDMA0_CONTROL register to avoid this behavior. For more information refer to anomaly 05-00-0427 and also the BF54x hardware reference manual, Chapter 26 - Enhanced Parallel Peripheral Interface, section System configuration.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

14. 05000369 - Addressing Conflict between Boot ROM and Asynchronous Memory:

DESCRIPTION:
When the Boot ROM (address range 0xEF00.0000-0xEF00.0FFF) and the Asynchronous Memory space (address range 0x2000.0000-0x2FFF.FFFF) are accessed simultaneously, either access can return incorrect data. Any combination of core read/writes, instruction fetches, cache line fills or DMA read/writes can trigger the problem. It applies to all NOR flashes, including Synchronous NOR flash, but not to NAND or ATAPI accesses on the same bus.

WORKAROUND:
1) Do not DMA values out of the Boot ROM while the core or any other DMA is accessing the Asynchronous Memory space or fetching from there.
2) Avoid any DMA from/to Asynchronous Memory space running in the background when calling any of the ROM functions.
3) Do not call ROM routines while executing out of Asynchronous Memory space
4) Do not open a memory window in an IDDE that shows the Boot ROM while accessing Asynchronous Memory space or vice versa.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

15. 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, 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):
0.1

16. 05000378 - Security/Authentication Speedpath Causes Authentication To Fail To Initiate:

DESCRIPTION:
Operating the processor above a maximum CCLK of 400 MHz and/or below a minimum VDDINT of 1.1875 V results in failure to initiate digital signature authentication. There is no dependence upon SCLK.

Outside these limits, Authentication will fail to complete successfully. The failure mode is a "-20" error code returned from the authentication routine, indicating that the processor failed to enter "Secure Entry" mode. Note that this anomaly will not compromise security nor will it allow invalid authentication to result in success.

WORKAROUND:
Authentication is possible at CCLK rates up to 400 MHz when VDDINT is at or above 1.188 V. Therefore, ensure that the regulator is programmed to 1.25 V (with -5% tolerance, this is 1.188 V) and reduce the core frequency (CCLK) to a maximum of 400 MHz prior to initiating digital signature authentication.

There is no dependence upon SCLK.

APPLIES TO REVISION(S):
0.1

17. 05000379 - 16-Bit NAND FLASH Boot Mode Is Not Functional:

DESCRIPTION:
The 16-Bit NAND FLASH boot mode is not functional.

WORKAROUND:
Use one of the other boot modes.

APPLIES TO REVISION(S):
0.1

18. 05000404 - Lockbox SESR Disallows Certain User Interrupts:

DESCRIPTION:
When the processor enters Secure Entry Mode, interrupts are shut off (i.e. IMASK is cleared except for bits [4:0] because they are hardwired). The user can select interrupts to be unmasked via an argument to the SESR. The IVTMR (core timer) and IVHW (HW error) interrupts cannot be unmasked inside the SESR Therefore, IVTMR and IVHW cannot be serviced during the digital signature authentication process (Secure Entry Mode).

WORKAROUND:
None

APPLIES TO REVISION(S):
0.1

19. 05000405 - Lockbox SESR Firmware Does Not Save/Restore Full Context:

DESCRIPTION:
Embedding `asm("raise 2;");` to call authentication in C code is not recommended. Registers R0-R3 and P0-P2 are not saved in the SESR. The compiler will assume that any C code after `asm("raise 2;");` will still be able to use these registers safely, although they are overwritten inside the SESR.

WORKAROUND:
The following C code instruction, which informs the compiler that it is not safe to use the registers R0-R3 and P0-P2, may be used to begin authentication:

```
asm("raise 2;":::"R0", "R1", "R2", "R3", "P0", "P1", "P2");
```

When using assembly code, the user must save the registers R0-R3 and P0-P2, issue the `raise 2;` instruction, then restore the registers R0-R3 and P0-P2.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

20. 05000406 - Lockbox SESR Argument Checking Does Not Check L2 Memory Protection Range:

DESCRIPTION:
Lockbox SESR firmware fails to warn user if any portion of the digitally signed message is placed outside of the range of protected memory space in on-chip L2 memory during the digital signature authentication process.

By default upon entry into Secure Entry Mode, 64KB of L2 memory is protected against DMA access (0xFEB00000 - 0xFEB0FFFF). Lockbox SESR firmware performs argument checking in order to verify that the user does not place any portion of the digitally signed message outside of protected memory space. Memory protection via DMA restricted access to the L2 memory is enabled and functional. The anomalous behavior is that the SESR will allow the user to place sensitive information outside of the protected memory space without warning and without aborting the authentication process in order to protect the user's sensitive information which lies outside of protected memory space from being compromised.

WORKAROUND:
User's who intend to place sensitive information in on-chip L2 memory must ensure that their placement is completely bounded within the protected memory space range: 64KB of memory (0xFEB00000 - 0xFEB0FFFF).

APPLIES TO REVISION(S):
0.1

21. 05000407 - Lockbox SESR Firmware Arguments Are Not Retained After First Initialization:

DESCRIPTION:
Lockbox SESR firmware arguments must be reset in subsequent digital signature authentication attempts as the arguments are not retained after first initialization.

WORKAROUND:
Lockbox SESR arguments must be programmed upon every authentication attempt.

If digital signature authentication is to be attempted following a failure or abortion of the authentication process, users must reset their SESR arguments for subsequent authentication attempts.

APPLIES TO REVISION(S):
0.1

22. 05000408 - Lockbox Firmware Memory Cleanup Routine Does not Clear Registers:

DESCRIPTION:
The security firmware memory clear routine does not clear processor registers. It only clears on-chip SRAM memory.

Sensitive information may remain in processor registers upon exiting from Secure Mode, so users must not rely on the firmware memory clear routine to clear registers.

WORKAROUND:
Users should clear out processor registers prior to exiting Secure Mode. The security firmware memory clear routine may be called to clear on-chip SRAM or users may substitute their own memory clear routine instead.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

23. 05000409 - Lockbox firmware leaves MDMA0 channel enabled:

DESCRIPTION:
During the digital signature authentication process, the security firmware uses the MDMA0 channel, but does not disable the channel after use. If the customer application made use of that same MDMA0 channel, the customer may encounter problems configuring it.

WORKAROUND:
Customer applications should first disable the MDMA0 channel used by the security firmware before attempting to reconfigure it. This is only necessary following a successful authentication process.

APPLIES TO REVISION(S):
0.1

24. 05000411 - bfrom_SysControl() Firmware Function Cannot be Used to Enter Power Saving Modes:

DESCRIPTION:
The *bfrom_SysControl()* firmware function does not manage SIC_IWRx wakeup bits properly and cannot be used to enable the processor to enter sleep or deep sleep mode.

WORKAROUND:
Write to the PLL_CTL register directly to enter sleep or deep sleep mode.

APPLIES TO REVISION(S):
0.1

25. 05000413 - NAND Boot Mode Not Compatible With Some NAND Flash Devices:

DESCRIPTION:
The NAND boot mode is not functional with a majority of the Micron NAND flash devices. The Micron NAND flash parts can have two power cycling procedures. If the device requires the issuing of the RESET command after a power cycle and the device can become busy for more than 11000 core clock cycles, then the device is not supported. Please refer to the NAND flash device datasheet for details of the required power cycling procedure.

WORKAROUND:
This workaround only applies to rev 0.1 silicon.

Use NAND flash devices that do not impose the restriction of issuing a RESET command followed by an extended busy time (greater than 11000 core clock cycles) after a power cycle.

APPLIES TO REVISION(S):
0.1

26. 05000414 - OTP_CHECK_FOR_PREV_WRITE Bit is Not Functional in bfrom_OtpWrite() API:

DESCRIPTION:
OTP_CHECK_FOR_PREV_WRITE is a flag which is provided to enable the check for unprogrammed OTP bits in the firmware OTP write operation.

If the *OTP_CHECK_FOR_PREV_WRITE* bit is set in the *dFlags* argument in the *bfrom_OtpWrite()* API, the OTP data bits are first checked for zero (unprogrammed) values before the write is allowed to proceed. This error detection for a write to a previously programmed page causes dedicated error messages and the write does not occur. If the *OTP_CHECK_FOR_PREV_WRITE* bit is cleared (default), the write proceeds without first performing this check.

The *OTP_CHECK_FOR_PREV_WRITE* bit is not functional in firmware and therefore this feature cannot be configured by the user. The firmware OTP write operation always performs the check for unprogrammed OTP bits to the data page target of the write operation. If the page is detected as being previously programmed (i.e., page contains a bit whose value is '1'), the write will not occur.

WORKAROUND:
None.

APPLIES TO REVISION(S):
0.1

27. 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):
0.1, 0.2, 0.3M

28. 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):

0.1, 0.2, 0.3M

29. 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;
```

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

30. 05000427 - CORE_EPPI_PRIO bit and SYS_EPPI_PRIO bit in the HMDMA1_CONTROL register are not functional:

DESCRIPTION:
The CORE_EPPI_PRIO bit and SYS_EPPI_PRIO bit in the HMDMA1_CONTROL register are not functional. These bits, in addition to the urgent watermark bits, help to further elevate the priority of EPPI (EPPI0/1/2) transactions at the DDR controller interface. Use of these bits is required only under high DDR activity resulting from the core and several other DMA channels (including EPPI DMA channels) simultaneously accessing DDR memory. For more information, please refer to the "System Configuration" section of the EPPI Chapter in the ADSP-BF54x Hardware Reference Manual.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.1

31. 05000429 - WB_EDGE Bit in NFC_IRQSTAT Incorrectly Reflects Buffer Status Instead of IRQ Status:

DESCRIPTION:
The NAND Flash Controller (NFC) "write buffer edge detect" (WB_EDGE) interrupt is generated when the 4-word-deep write buffer of the NFC becomes empty. The functional generation of the interrupt is correct. The functional description of the WB_EDGE bit is described as "The write buffer edge detect (WB_EDGE) is set when the write buffer transitions from 'not empty' to 'empty.'" This correctly describes the behavior through which the interrupt is generated. However, the status bit for this interrupt does not align with the functional behavior of the interrupt.
The WB_EDGE bit of NFC_IRQSTAT register currently behaves as a "write buffer empty" status bit instead of as an actual IRQ status bit. The WB_EDGE bit is set when the NFC write buffer is empty and cleared when the buffer is not empty. The IRQ is cleared correctly after the W1C operation, but the bit will remain set. This does not reflect the IRQ status, and the behavior of this bit is identical to the WB_EMPTY bit in the NFC_STAT register.

The NFC_IRQSTAT default reset value of 0x0004 indicates that an IRQ is already active. The intended default reset value is 0x0000.

WORKAROUND:
Do not depend upon the WB_EDGE bit of NFC_IRQSTAT register to determine status of the write buffer.

The status of the write buffer should be determined by checking the WB_EMPTY bit or WB_FULL bit in the NFC_STATUS register.

If software requires an interrupt handler that services WB_EDGE interrupts, the following procedure is advised:
1) Only unmask the WB_EDGE interrupt prior to performing NFC operations that are issued through the write buffer. Otherwise mask off the interrupt.
2) From within the interrupt handler, service all interrupts except for the WB_EDGE interrupt and clear them by performing the appropriate W1C operation to the NFC_IRQSTAT register.
3) Service the WB_EDGE interrupt only when the SIC_ISRx register indicates an NFC interrupt is still latched and the only bit set in the NFC_IRQSTAT register is the WB_EDGE bit.
4) Mask off the WB_EDGE interrupt within the WB_EDGE interrupt handler if the handler does not issue any NFC write buffer transactions. If the WB_EDGE IRQ handler does issue NFC write buffer transactions, ensure the current WB_EDGE interrupt is cleared before performing the transactions.

APPLIES TO REVISION(S):
0.1

32. 05000430 - Software System Reset Corrupts PLL_LOCKCNT Register:

DESCRIPTION:
Software reset (`raise 1;`, core double fault, watchdog reset, *bfrom_SysControl()* call with *SWRST* flag) will return with PLL_LOCKCNT register value set to 0x0007 (instead of intended reset value of 0x0200). This may result in failure to subsequently re-program PLL and Voltage Regulator control registers.

Hardware reset will result in proper reset value loaded into PLL_LOCKCNT and no action is required.

WORKAROUND:
Following any software reset (`raise 1;`, core double fault, watchdog reset, *bfrom_SysControl()* call with *SWRST* flag) users cannot rely upon the default reset value within the PLL_LOCKCNT register and must explicitly write a value of 0x0200 or larger to PLL_LOCKCNT register before making subsequent changes to PLL or Voltage Regulator control registers.

Do not use the Preboot mechanism for setting PLL or Voltage Regulator control registers if power settings will be changed before entering software reset. It is recommended to use application code or init code to explicitly write to PLL_LOCKCNT prior to changing the PLL or Voltage Regulator control registers as stated above.

APPLIES TO REVISION(S):
0.2

33. 05000431 - Incorrect Use of Stack in Lockbox Firmware During Authentication:

DESCRIPTION:
Some of the cryptographic routines utilized by the security firmware use the stack in ways that do not conform to the run-time execution model by using stack locations that reside both inside and outside the currently allocated stack frame. This causes problems when an interrupt occurs during authentication and the interrupt handler pushes data onto the stack thus possibly overwriting live data. When live data is overwritten by the interrupt handler, the behavior of the security firmware is undefined upon return from interrupt.

This anomaly can manifest itself as a processor hang. The corruption of the stack when an interrupt is serviced during authentication can result in the Program Counter not being properly returned from the Lockbox SESR firmware. There is no evidence of this anomaly resulting in a security compromise. Issuing reset may be required to recover from this anomalous behavior.

WORKAROUND:
There are several workarounds possible:

1) Interrupt handlers that occur during authentication should wrap themselves with the following two instructions:

```
/* Start of original interrupt handler */
SP += -60;
<body of interrupt handler>
SP += 60;
RTI;
/* End of original interrupt handler */
```

2) When using the Device Driver/System Services Libraries (DD/SS) distributed with VisualDSP++, the DD/SS libraries will have to be rebuilt from source with the fix presented above implemented. For example, to implement the fix in the version of the DD/SS distributed with VisualDSP++ 5.0 update 3, the following steps must be taken:

a) In /Blackfin/lib/src/services/int/adi_int_module.h, add the *SP += -60;* \ instruction immediately after the __STARTFUNC(Name) entry point for both ADI_INT_ISR_FUNCTION and ADI_INT_EXC_FUNCTION. For example:

```
#define ADI_INT_ISR_FUNCTION(Name,IVG,Nested) \
   __STARTFUNC(Name) \
      SP += -60; \                          // <-- ADD THIS INSTRUCTION
      [--SP] = R0; \
      [--SP] = P1; \
```

In the same file, increase the length of the adi_int_ISR_Entry array from 16 to 18.

b) In /Blackfin/lib/src/services/int/adi_int_asm.asm, the complementary stack modification must be made to the end of the handlers. The *SP += 60;* \ instruction must be inserted before the RTI in ADI_INT_ISR_EPILOG and before the RTX in ADI_INT_EXC_EPILOG. For example:

```
#define ADI_INT_EXC_EPILOG(Nested) \
   unlink;\
   SP +=  12;\
      .
      .
      .
   SP += 60; \                              // <-- ADD THIS INSTRUCTION
   RTX;\
   NOP;\
   NOP;\
   NOP;
```

c) Rebuild the DD/SS library with these changes and use them instead of the prebuilt libraries distributed with VisualDSP++.

3) Do not enable interrupts to be serviced during authentication.

APPLIES TO REVISION(S):
0.1, 0.2

34. 05000434 - SW Breakpoints Ignored Upon Return From Lockbox Authentication:

DESCRIPTION:

Upon returning from a failed Lockbox authentication attempt, software breakpoints are not able to halt the debugger until after the fourth breakpoint is executed.

This anomaly occurs when Condition #1 AND Condition #2 OR Condition #3 are met:

1) The code initiating the authentication by executing instruction "RAISE 2;" is executing from L1 Code Cache memory configured as SRAM

AND

2) The failure from authentication is due to a message-digital signature-message size mismatch.

OR

3) The failure from authentication is due to fact that the public key is not programmed and the firmware reads back all 0's.

Note that if the combination of Condition 1 and either Condition 2 or Condition 3 are not met, the anomaly will not be encountered.

WORKAROUND:

There are several workarounds possible:

1) Use a hardware breakpoint instead of a software breakpoint to break right after returning from authentication.

2) Do not initiate authentication from L1 Cache Code area of memory

3) Place multiple (at least four) NOPs after the return point of authentication and place a regular software breakpoint at each NOP. The first four will not execute as expected but the fifth (5th) breakpoint will trigger the debugger to halt. NOPs may be replaced with non-critical code if desired.

4) Do not link the calling routine that executes the instruction "RAISE 2;" that initiates authentication into L1 Cache Code space

APPLIES TO REVISION(S):

0.1, 0.2, 0.3M

35. 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
```

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

36. 05000446 - CDMAPRIO and L2DMAPRIO Bits in the SYSCR Register Are Not Functional:

DESCRIPTION:
The CDMAPRIO and L2DMAPRIO Bits in the SYSCR Register Are Not Functional. These bits provide control over the priority of DMAC0 and DMAC1 controller at the L1 and L2 memory interface. The priority order for DMAC0 and DMAC1 is intended to be configurable by the user and the order may be swapped. The priority configuration for L1 accesses is defined by the CDMAPRIO bit of the SYSCR register. For L2 accesses, the L2DMAPRIO bit in SYSCR is used in the same way.

The priorities will be as the fixed bits indicate, which is DMAC0 has higher priority over DMAC1 for L1 and L2 memory accesses. This priority can not be changed with respect to on-chip L1 and L2 memory access.

WORKAROUND:
The external memory access priorities between DMAC0 and DMAC1 can be configured but it is recommended that they be also set to the same priorities as L1 and L2 priorities for DMAC0 and DMAC1. i.e. DMAC0 higher priority than DMAC1.

Consequently, it is also recommended not to use MDMA0 and MDMA1 channels, as using a MDMA channel on a higher priority DMA controller (MDMA0 and MDMA1 are on DMAC0) can hold off peripheral-DMA accesses for channels on DMAC1. The MDMA channels can access memory every SCLK cycle which could lead to starving critical access for peripherals on DMAC1.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

37. 05000447 - UART IrDA Receiver Fails on Extended Bit Pulses:

DESCRIPTION:
The UART fails reception when the width of the receive pulse is wider than 3/16th. As defined by the standard, all IrDA transmitters assert bit pulses for exactly 3/16th of a bit period. Wired connections of an IrDA transmitter to the Blackfin IrDA receiver work properly. If the connection is not hard wired but is implemented instead via an infrared link, it has been observed that infrared tranceiver devices extend the output pulse beyond the 3/16th duration. This can cause the Blackfin UART IrDA receiver to fail at higher bit rates. The baud rate where the Blackfin UART IRDA function fails depends on the characteristics of the particular IRDA transceiver module that is used. When an infrared tranceiver that employs extended bit pulses is used, the Blackfin receiver still properly detects the start bit. However, the Blackfin may not properly latch in data. The receive interrupt count may also not match the number of transmitted bytes.

WORKAROUND:
There are several workarounds possible:
1) Add external logic ensuring IRDA RXD pulse width is always 3/16th parts of the UART bit rate.
2) Use external IRDA Encoder/Decoder (for example: HSDL-7000, MCP2122, TIR1000)
3) Use IRDA transceiver modules where tPW max = 3/16 parts of the UART Baud Rate (1.6us @ 115200 bps). ADI has not identified any devices meeting this criteria.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

38. 05000448 - DDR Clock Duty Cycle Spec Violation (tCH, tCL):

DESCRIPTION:
DDR clock duty cycle specifications tCH and tCL are not met. This can result in DDR read/write failures.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.1

39. 05000449 - Reduced Timing Margins on DDR Output Setup and Hold (tDS and tDH):

DESCRIPTION:
tDS and tDH are the DQ/DQM to DQS setup and hold times during a DDR write transaction. These specifications have a minimum value of 700ps which deviates from the 900ps values stated in the product data sheet.

If proper care is not taken to account for this issue, through tighter board design tolerances and usage of high performance memory devices, bit errors may occur during write transactions.

DDR SDRAM Write Cycle Timing Switching Characteristics
tDS (output setup) - DQ/DQM Setup to DQS - Time between DQ/DQM valid and DQS edge
tDH (output hold) - DQ/DQM Hold to DQS - Time between DQS edge and DQ/DQM invalid

WORKAROUND:
Use faster -5 (200MHz) speed grade DDR memory devices that require only 400ps DQ and DQM input setup/hold time relative to DQS. This is suggested to provide greater margin for trace length mismatch and cross talk on PCBs.

In addition, follow recommended board design guidelines in Micron TN-46-14. Increase intra-group trace match precision to +/- 30 mil. Place the DDR device as close to the Blackfin processor as possible.

APPLIES TO REVISION(S):
0.1

40. 05000450 - USB DMA Short Packet Data Corruption:

DESCRIPTION:
DMA Mode 1:

DMA mode 1 allows large size transfers to generate a single interrupt at the end of the entire transfer. The transfer is split up in packets of length specified in the Maximum Packet Size field for that endpoint. If the transfer size is not an integer multiple of the Maximum Packet Size, a short packet will be present at the end of the transfer. When using DMA mode 1 the short packet may be corrupted in the TX USB FIFO.

DMA Mode 0:

For the case where the entire transfer is done using DMA mode 0, the short packet can be corrupted if double buffering is enabled for the TX Endpoint FIFO.

Note: if the short packet is a multiple of 4 bytes, the corruption will not occur.

WORKAROUND:
Use DMA mode 1 to transfer (n* Maximum Packet Size) and schedule DMA mode 0 to transfer the short packet.

For example, if your transfer size is 33168 bytes and Maximum Packet Size equals 512, schedule [33168 - (33168 mod 512)] in DMA mode 1 and the remainder (33168 mod 512) in DMA mode 0.

When using only DMA mode 0 for the entire transfer, disable double buffering to avoid corruption of the short packet.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

41. 05000456 - USB Receive Interrupt Is Not Generated in DMA Mode 1:

DESCRIPTION:
Whether the USB is used in host or device mode, the USB receive interrupt may not be generated when DMA Mode 1 is used.

For DMA Mode 1 host mode receive operations where the transfer size is an integer multiple of MaxPacketSize, extra "in" tokens are sent out by the USB controller at the end of the DMA transfer. This causes the slave device to send an additional data packet back to the Blackfin processor, where it is received in the USB FIFO, but no USB RX interrupt is generated. Taking the Mass Storage Class as a specific example, this causes the devices to send a status packet, which should generate a USB RX interrupt, however, this interrupt may be lost.

For DMA Mode 1 device mode receive operations where the transfer size is unknown, the Short Packet Interrupt must be relied upon to indicate the end of the transfer. However, this anomaly prevents the USB controller from issuing an RX interrupt for the corresponding endpoint when a short/null packet is received.

This anomaly does not apply to device mode when the size of the receive transfer is known in advance, as the DMA Completion Interrupt is generated at the end of the transfer and the endpoint receive interrupt is not used.

This anomaly also does not apply to transmit operations.

WORKAROUND:
In all affected cases described above, use DMA Mode 0 instead of DMA Mode 1.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

42. 05000457 - Host DMA Port Responds to Certain Bus Activity Without HOST_CE Assertion:

DESCRIPTION:
The Host DMA Port (HOSTDP) responds to certain bus activity even without the assertion of its chip enable, HOST_CE. If the HOSTDP is the only slave on the bus (meaning that HOST_WR and HOST_RD are asserted by the host only for communicating with the HOSTDP), this anomaly will not be observed. There are two states in which the HOSTDP responds to bus activity (assertion of HOST_WR or HOST_RD) without HOST_CE being asserted:

1. HOSTDP is Enabled and Waiting for the Host to Configure - While waiting for configuration in both acknowledge and interrupt modes, assertions of HOST_WR for other slaves can cause the HOSTDP to be erroneously configured.

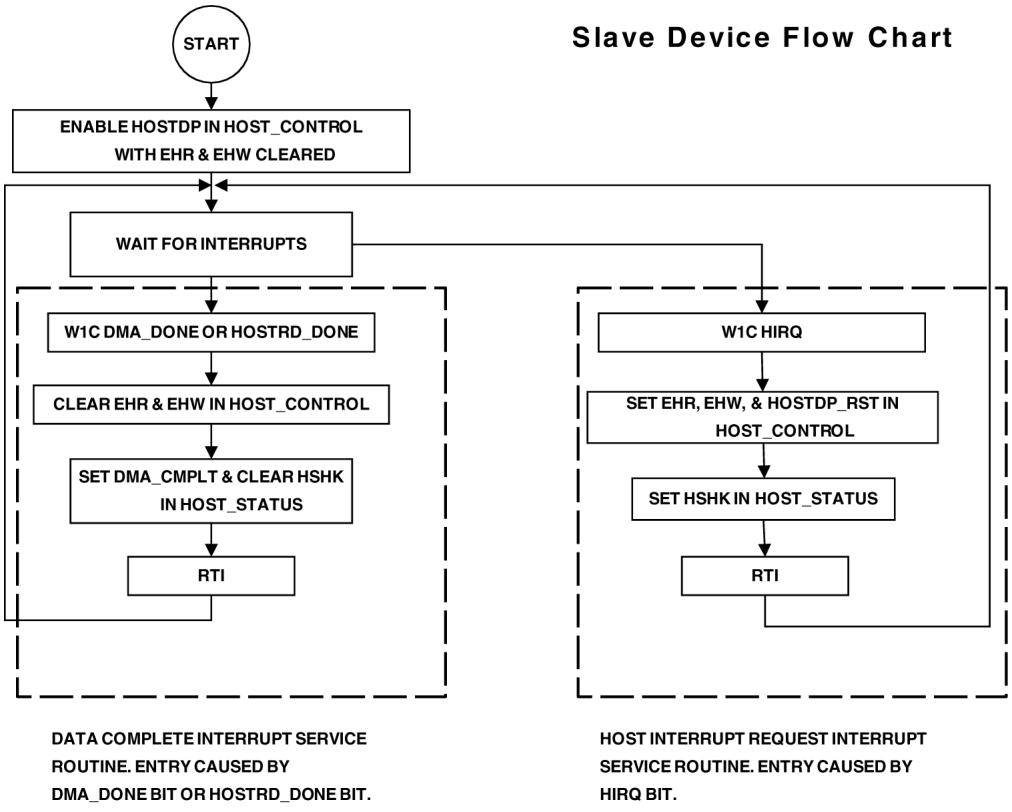
2. HOSTDP is Configured for Data Reads - While waiting for data reads in both acknowledge and interrupt modes, assertions of HOST_RD for other slaves can cause the HOSTDP to subsequently return data out of order.

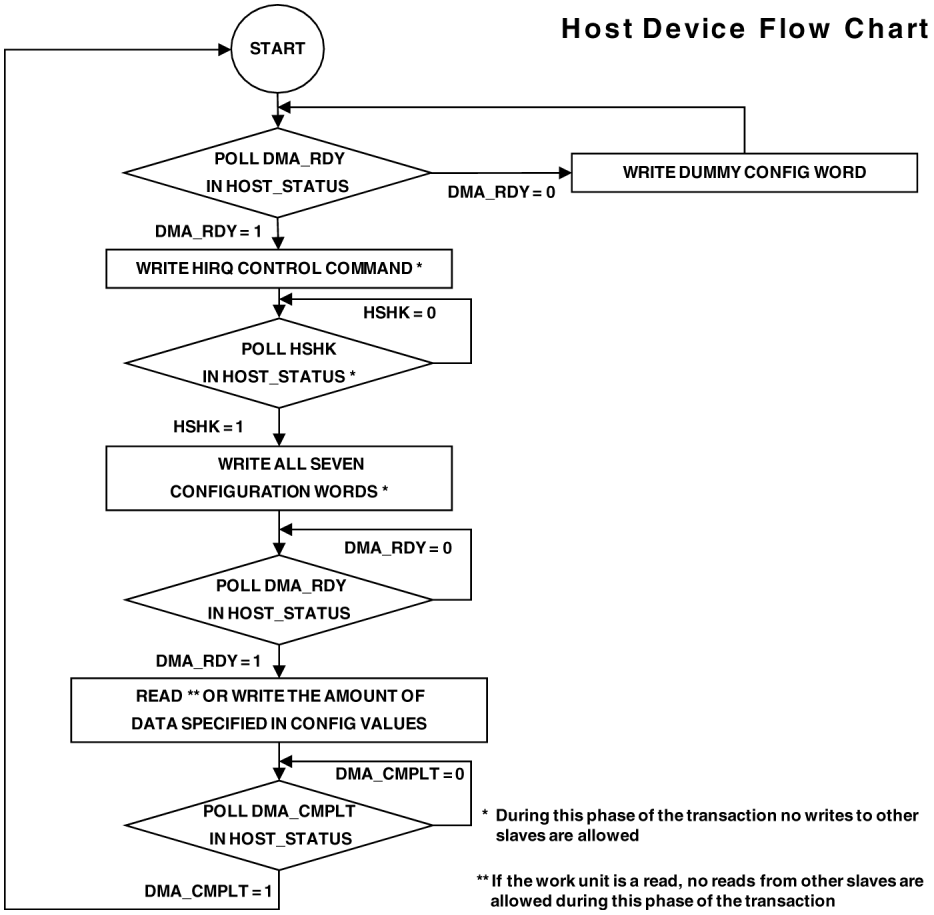
WORKAROUND:
The following workarounds can be used for state #1:

a. Add additional logic - Connect to HOST_WR through a 2-input OR gate where one input of the OR gate is the host's write signal and the other is the host's chip select.

b. Time bus accesses by the host processor - Do not write to other slaves on the bus between the time that the HOSTDP is enabled and the DMA_RDY bit in HOST_STATUS is set.

c. Change the software on both the host and the slave as shown in the following flowcharts:





The following workarounds can be used for state #2:

a. Add additional logic - Connect to $\overline{\text{HOST\_RD}}$ through a 2-input OR gate where one input of the OR gate is the host's read signal and the other is the host's chip select.

b. Time bus accesses by the host processor - Do not read from other slaves on the bus between the time that the final configuration word (YMODIFY) is written to the HOSTDP and the DMA_CMPLT bit in HOST_STATUS is set.

APPLIES TO REVISION(S):

0.1, 0.2, 0.3M

43. 05000460 - USB DMA Mode 1 Failure When Multiple USB DMA Channels Are Concurrently Enabled:

DESCRIPTION:
When multiple USB DMA Mode 1 channels are enabled and active at the same time, one of the channel's DMA Address registers may be corrupted resulting in either DMA hang or data corruption.

WORKAROUND:
Use DMA Mode 0 if the application requires multiple USB DMA channels to be concurrently enabled.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

44. 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):
0.1, 0.2, 0.3M

45. 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):
0.1, 0.2, 0.3M

46. 05000463 - USB DMA RX Data Corruption:

DESCRIPTION:
USB DMA Rx data corruption is observed when the USB buffer destination is in L1 or L2 memory and another peripheral's DMA buffers, e.g., SPORT, are also in L1 or L2 memory spaces and are accessed at the same time as the USB DMA is accessing its buffer.

WORKAROUND:
When multiple peripherals are used with buffers in L1 or L2, place USB buffers in L3 or vice versa.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

47. 05000464 - USB TX DMA Hang:

DESCRIPTION:
USB TX DMA hangs while reading data from L1 or L2 memory at the same time another peripheral e.g., SPORT0, is receiving and writing data to L1 or L2 memory.

WORKAROUND:
When multiple peripherals are used with buffers in L1 or L2 place USB buffers in L3 or vice versa.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

48. 05000465 - USB Rx DMA Hang:

DESCRIPTION:
USB Rx DMA hangs if the endpoint FIFOs are configured in double buffer mode (this is the case when MaxPacketSize in USB_EP_NIx_RXMAXP is equal or less than half the endpoint FIFO size) When double buffering is enabled, there is the possibility of a race condition where RxPktRdy is set and cleared in the same cycle. When this happens RxPktRdy will remain cleared, thus preventing the USB DMA from unloading the FIFO, resulting in a Rx DMA hang.

WORKAROUND:
Use DMA mode 0 with double buffering disabled.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

49. 05000466 - TxPktRdy Bit Not Set for Transmit Endpoint When Core and DMA Access USB Endpoint FIFOs Simultaneously:

DESCRIPTION:
TX DMA data can be lost when both the USB DMA and the core access two different USB endpoint FIFOs at the same time. The DMA pointer does not increment correctly for the last two bytes of the DMA accessed FIFO, thus preventing the USB controller from setting TXPKTRDY when AUTOSET is enabled. If TXPKTRDY is set manually, data will be sent on the bus but the last two bytes will be missing.

WORKAROUND:
Do not mix concurrent DMA and core accesses to the USB TX endpoint FIFOs.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

50. 05000467 - Possible USB RX Data Corruption When Control & Data EP FIFOs are Accessed via the Core:

DESCRIPTION:
Received data may be corrupted if the RX FIFO is accessed via the core under the following conditions:

1. Control and data USB endpoints are enabled.
2. Data has been received at the control endpoint.
3. Data is being received or has been received at the data endpoint.
4. Core reads an ODD number of bytes from the control endpoint, EP0.
5. Subsequent core read of the data endpoint's RX FIFO will return corrupted data.

WORKAROUND:
Use DMA to read data from the streaming (data) endpoint FIFO.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

51. 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):
0.1, 0.2, 0.3M

52. 05000474 - Access to DDR SDRAM Causes System Hang with Certain PLL Settings:

DESCRIPTION:
For specific CCLK::SCLK ratios, back-to-back core reads/writes (data accesses or instruction fetches) over the EBIU, followed by a third core read/write over the EBIU will result in a core hang. The EBIU access subsequent to the original back-to-back core EBIU accesses is not constrained to following immediately afterward or within a certain period of time. Data transferred in the second core EBIU access in the back-to-back accesses may also be lost as a result of this anomaly. For example, a core hang will occur as a result of the following sequences:

1) Read/Write DDR -> *Read/Write Asynchronous Memory -> Read Boot ROM
2) Read/Write Asynchronous Memory -> *Read/Write DDR -> Read/Write Asynchronous Memory
3) Read/Write DDR -> *Read/Write DDR -> Read DDR

* In addition to the core hang, this data may also be lost.

The failure occurs with a CCLK::SCLK ratio of N::M, where N is odd OR M > 1 and odd. The failure is independent of DDR timing parameters and delay line settings, and cache configuration does not influence whether or not the failure occurs.

WORKAROUND:
1) Use CCLK::SCLK ratios of N:1, where N is even, OR
2) Use DMA to perform accesses.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

53. 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.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

54. 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):
0.1, 0.2, 0.3M

55. 05000483 - Possible USB Data Corruption When Multiple Endpoints Are Accessed by the Core:

DESCRIPTION:
When using core transfers to fetch data from endpoint FIFOs with multiple endpoints enabled, data corruption can occur when the core is reading data from one endpoint FIFO and a change in code flow occurs immediately prior to this read committing in the pipeline. If this code accesses a different endpoint FIFO, the core will read the data from the different endpoint FIFO; however, when the application resumes with the read access to the previous endpoint FIFO that did not commit, the read is corrupted.

Note that this change in code flow could be due to a different endpoint interrupt, or the read could be speculatively executed in the shadow of a conditional branch. Exceptions can also cause this problem, but only in the unusual case where an access to an endpoint FIFO takes place in the exception handler. The most likely scenario for this corruption to occur is when the core is reading data from one endpoint and gets interrupted to service a different endpoint.

WORKAROUND:
1) Use the USB DMA to read from the Endpoint FIFOs.
2) When multiple Endpoint FIFOs are enabled, disable interrupts around reads from an endpoint FIFO.
3) Ensure that Endpoint FIFO reads do not occur in the shadow of a conditional branch by placing three NOPs between the branch instruction and the read.

APPLIES TO REVISION(S):
0.1, 0.2, 0.3M

56. 05000484 - DDR Trim May Not Be Performed for Certain VLEV Values in OTP Page PBS00L:

DESCRIPTION:
When using the preboot routine and OTP memory page PBS00L to define custom default values for VR_CTL and VLEV, DDR trim may not be performed if the value of the programmed VLEV in OTP is offset by less than two levels from the reset value of VLEV.

WORKAROUND:
Ensure that the VLEV value programmed into OTP memory is offset by at least 2 from the VLEV reset value.

APPLIES TO REVISION(S):
0.1, 0.2

57. 05000485 - PLL_CTL Change Using bfrom_SysControl( ) Can Result in Processor Overclocking:

DESCRIPTION:
When bfrom_SysControl() is called with both the SYSCTRL_PLLCTL and the SYSCTRL_PLLDIV flags set in dActionFlags, and the new PLL_CTL value has either the PDWN or the STOPCK bit set, then MSEL gets updated in the PLL before CSEL/SSEL, which can lead to a situation where the processor is overclocked (depending on the new MSEL value).

WORKAROUND:
If setting either the PDWN or STOPCK bits in the new value of PLL_CTL passed into bfrom_SysControl(), the possible workarounds to avoid overclocking are:

1) Set the new value of MSEL equal to the old value of MSEL, OR
2) Do not set the SYSCTL_PLLDIV flag in the same call to bfrom_SysControl().

APPLIES TO REVISION(S):
0.2, 0.3M