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>
0.4	0x0004
0.3M*	0x0003

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

APPLICABILITY

Each anomaly applies to specific silicon revisions. See Summary or Detailed List for affected revisions. Additionally, not all processors described by this anomaly list have the same feature set. Therefore, peripheral-specific anomalies may not apply to all processors. See the below table for details. An "x" indicates that anomalies related to this peripheral apply only to the model indicated, and the list of specific anomalies for that peripheral appear in the rightmost column.

Peripheral	ADSP-BF549	ADSP-BF548	ADSP-BF547	ADSP-BF544	ADSP-BF542	Anomalies
SDIO	x	x	x		x	None
ATAPI	x	x	x		x	None
USB	x	x	x		x	05000450, 05000456, 05000460, 05000463, 05000464, 05000465, 05000466, 05000467, 05000483, 05000510
Keypad	x	x	x		x	None
MXVR	x					None
CAN	x	x		x	x	None
HDMA	x	x	x	x		05000457

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
01/25/2016	L	E	Added Anomalies: 05000329, 05000503, 05000506, 05000508, 05000510, 05000511 Revised Anomaly: 05000265 Removed Silicon Revision(s) 0.1, 0.2

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

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	Rev 0.3 M	Rev 0.4
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x
2	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x
3	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x
4	05000220	Data Corruption/Core Hang with L2/L3 Configured in Writeback Cache Mode	x	.
5	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
6	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x
7	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x
8	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
9	05000329	Synchronous Burst Flash Boot Mode Is Not Functional	x	x
10	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	x
11	05000360	External Memory Read Access Hangs Core With PLL Bypass	x	x
12	05000365	DMAs that Go Urgent during Tight Core Writes to External Memory Are Blocked	x	x
13	05000369	Addressing Conflict between Boot ROM and Asynchronous Memory	x	x
14	05000405	Lockbox SESR Firmware Does Not Save/Restore Full Context	x	x
15	05000408	Lockbox Firmware Memory Cleanup Routine Does not Clear Registers	x	x
16	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x
17	05000425	Multichannel SPORT Channel Misalignment Under Specific Configuration	x	.
18	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
19	05000434	SW Breakpoints Ignored Upon Return From Lockbox Authentication	x	x
20	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x
21	05000446	CDMAPRIO and L2DMAPRIO Bits in the SYSCR Register Are Not Functional	x	x
22	05000447	UART IrDA Receiver Fails on Extended Bit Pulses	x	x
23	05000450	USB DMA Short Packet Data Corruption	x	.
24	05000456	USB Receive Interrupt Is Not Generated in DMA Mode 1	x	x
25	05000457	Host DMA Port Responds to Certain Bus Activity Without $\overline{\text{HOST\_CE}}$ Assertion	x	x
26	05000460	USB DMA Mode 1 Failure When Multiple USB DMA Channels Are Concurrently Enabled	x	.
27	05000461	False Hardware Error when RETI Points to Invalid Memory	x	x
28	05000462	Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign	x	.
29	05000463	USB DMA RX Data Corruption	x	.
30	05000464	USB TX DMA Hang	x	.
31	05000465	USB Rx DMA Hang	x	x
32	05000466	Simultaneous Core/DMA Access to USB Endpoint FIFOs Doesn't Set TX Endpoint TxPktRdy Bit	x	.
33	05000467	Possible USB RX Data Corruption When Control & Data EP FIFOs are Accessed via the Core	x	.
34	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x	x
35	05000474	Access to DDR SDRAM Causes System Hang with Certain PLL Settings	x	.
36	05000477	TESTSET Instruction Cannot Be Interrupted	x	x
37	05000481	Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption	x	x
38	05000483	Possible USB Data Corruption When Multiple Endpoints Are Accessed by the Core	x	x
39	05000485	PLL_CTL Change Using bfrom_SysControl( ) Can Result in Processor Overclocking	x	.
40	05000489	PLL May Latch Incorrect Values Coming Out of Reset	x	x

No.	ID	Description	Rev 0.3 M	Rev 0.4
41	05000490	SPI Master Boot Can Fail Under Certain Conditions	x	x
42	05000491	Instruction Memory Stalls Can Cause IFLUSH to Fail	x	x
43	05000494	EXCPT Instruction May Be Lost If NMI Happens Simultaneously	x	x
44	05000498	CNT_COMMAND Functionality Depends on CNT_IMASK Configuration	x	x
45	05000500	NFC Hang When AMC Requests Async Pins During Last 16 Bytes of Page Write	x	x
46	05000501	RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes	x	x
47	05000502	Async Memory Writes May Be Skipped When Using Odd Clock Ratios	x	x
48	05000503	SPORT Sign-Extension May Not Work	x	x
49	05000506	Hardware Loop Can Underflow Under Specific Conditions	x	x
50	05000508	UART Receive DMA Hangs under Certain Conditions	x	x
51	05000510	USB Wakeup from Hibernate State Requires Re-Enumeration	x	x
52	05000511	Lower 16 Bits of CNT_COUNTER Register Do Not Update Properly	.	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, 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):
0.3M, 0.4

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):
0.3M, 0.4

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, 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):
0.3M, 0.4

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):
0.3M

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, 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):

0.3M, 0.4

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. When excessive noise occurs during high-frequency transitions on a slowly ramping RSCLK/TSCLK signal, it 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, which can result in numerous operational failures.

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

- In stereo serial modes, a frame sync may be missed, causing left/right data swapping.
- In multi-channel mode, multi-channel frame delay (MFD) counts may appear inaccurate or frames may be skipped.
- In normal (early) frame sync mode, received data words could 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 frame sync logic was already triggered, the next "normal" RSCLK will not detect the change in RFS. 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 multi-channel mode, the 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.

While the above audio failures are possible signatures associated with this anomaly, numerous other failures are possible due to the internal logic being subjected to what amounts to an out-of-spec clock signal. Even though the external signal is within specification, the noise causes multiple transitions to be sensed where only one transition actually occurred, resulting in an out-of-spec clock being presented to the internal logic. This can lead to illegal logic states and/or incorrect advancement of state machines, which adversely affects the SPORT itself and synchronization with other logic units like the DMA engine. A number of failure scenarios may result from this, including misreported SPORT/DMA errors and unexpected DMA halts.

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):

0.3M, 0.4

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 anomaly does not exist.

APPLIES TO REVISION(S):
0.3M, 0.4

8. 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):
0.3M, 0.4

9. 05000329 - Synchronous Burst Flash Boot Mode Is Not Functional:

DESCRIPTION:
The synchronous burst flash mode is not functional. Due to a missing mux on PORTI, the NORCLK does not appear on PI15.

WORKAROUND:
Use asynchronous flash mode for booting from NOR flash.

APPLIES TO REVISION(S):
0.3M, 0.4

10. 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):
0.3M, 0.4

11. 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):
0.3M, 0.4

12. 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):
0.3M, 0.4

13. 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):
0.3M, 0.4

14. 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):
0.3M, 0.4

15. 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):
0.3M, 0.4

16. 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):

0.3M, 0.4

17. 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):
0.3M

18. 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):

0.3M, 0.4

19. 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):
0.3M, 0.4

20. 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):
0.3M, 0.4

21. 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):
0.3M, 0.4

22. 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):
0.3M, 0.4

23. 05000450 - USB DMA Short Packet Data Corruption:

DESCRIPTION:
When DMA mode 0 is used for the entire transfer, the short packet can be corrupted if double buffering is enabled for the TX Endpoint FIFO.

When DMA mode 1 is used, large transfers generate a single interrupt at the end of the entire transfer. The transfer itself is split up into packets with a 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, thus making it susceptible to the same short packet corruption.

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

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

When using only DMA mode 0 for the entire transfer, there are two workarounds:
1) Disable double buffering.
2) Ensure that the short packet's length is a multiple of 4 bytes.

APPLIES TO REVISION(S):
0.3M

24. 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):
0.3M, 0.4

25. 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.

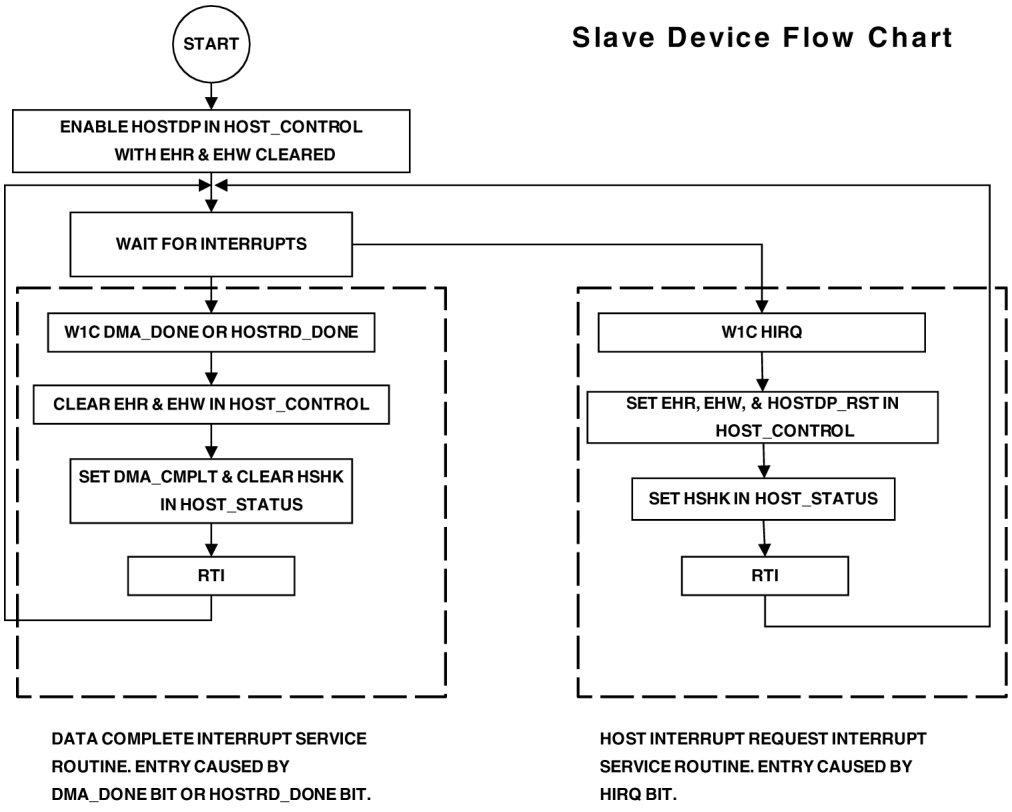
2) HOSTDP is Configured for Data Reads - While waiting for data reads in both acknowledge and interrupt modes, assertions of HOST_RD for other slaves can cause the HOSTDP to subsequently return data out of order.

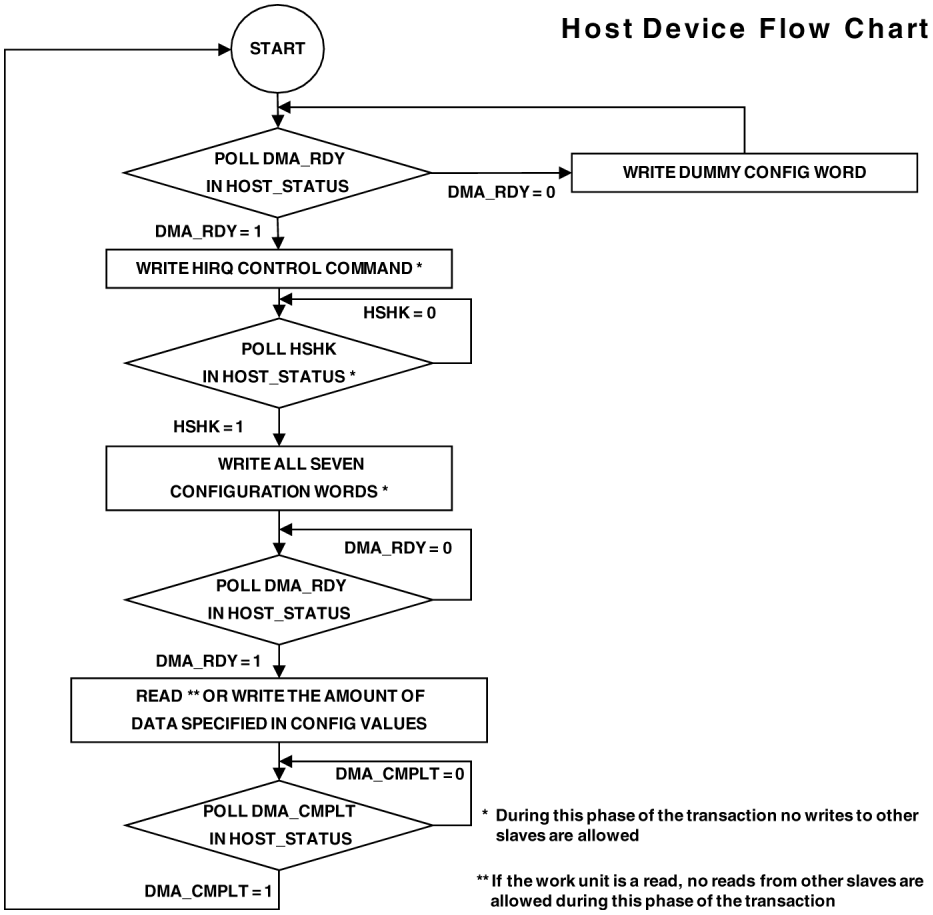
WORKAROUND:
The following workarounds can be used for state #1:

1) 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.

2) 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.

3) 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:

1) 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.

2) 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):

0.3M, 0.4

26. 05000460 - USB DMA Mode 1 Failure When Multiple USB DMA Channels Are Concurrently Enabled:

DESCRIPTION:

When multiple USB DMA channels are enabled/active at the same time and any enabled channel uses DMA Mode 1, one of the channel's DMA Address registers may be corrupted, resulting in either a 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):

0.3M

27. 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):

0.3M, 0.4

28. 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):
0.3M

29. 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):
0.3M

30. 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):
0.3M

31. 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):
0.3M, 0.4

32. 05000466 - Simultaneous Core/DMA Access to USB Endpoint FIFOs Doesn't Set TX Endpoint TxPktRdy Bit:

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):
0.3M

33. 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):
0.3M

34. 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):
0.3M, 0.4

35. 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):
0.3M

36. 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):
0.3M, 0.4

37. 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):
0.3M, 0.4

38. 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):
0.3M, 0.4

39. 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):
0.3M

40. 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):

0.3M, 0.4

41. 05000490 - SPI Master Boot Can Fail Under Certain Conditions:

DESCRIPTION:

Master Mode SPI Booting can fail under certain combinations of SPI_BAUD, CCLK::SCLK ratio, boot block size, and SDRAM Refresh Rate. The root cause for this problem is described in anomaly 05-00-0501. This is the manifestation of that anomaly due to the boot ROM sequence, which does not incorporate the software workaround to the underlying hardware problem of a stuck RXS bit in the SPI_STAT register.

When the RXS bit gets stuck as a result of anomaly 05-00-0501, a subsequent re-enabling of the SPI port results in DMA requests to a FIFO that has not yet been populated. This causes bogus data retrieved at the end of the previous block to be interpreted by the boot ROM as an invalid block header for the next block, which causes the boot to abort. There are two places in the boot ROM where the device is susceptible to this, manifesting in one of three ways:

1) When a bootable image block size exceeds 64K, it is broken into multiple DMA work units. In the DMA handler invoked between the work units, the anomaly can be encountered.

2) When SPI_BAUD = 2, the maximum SPI baud rate of SCLK/4 aligns exactly with the boot ROM execution frequency, which allows for the SPI disable to align exactly with a word being received (as a result of the SPI's behavior to continue issuing clocks even after the RX DMA is completed). At this particular baud rate, the SPI issues exactly 40 clocks between when the DMA completes and when the SPI is disabled in the ROM. This equates to exactly 5 additional received bytes, which completely fills the 4-deep SPI RX FIFO and the shift register, which asserts the RXS bit as the SPI is shut down.

3) When system timing parameters allow for any single word to get transferred from the shift register to the SPI FIFO exactly as the SPI port is being shut down, the anomaly can theoretically be encountered, though it has not been observed on silicon or in simulations. All of the system timing parameters mentioned above must combine to cause the timing that triggers the anomaly.

WORKAROUND:

For case 1), do not allow block sizes over 64K.

For case 2), using any SPI_BAUD setting other than 2 avoids the timing required to encounter the anomaly.

For case 3), if the problem were to occur, the behavior would be consistent and repeatable. Changing any of the SPI_BAUD, CCLK::SCLK ratio, block sizes, and/or SDRAM refresh rate will alter the timing (as aligned to boot ROM execution) such that the problem can be avoided.

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, 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):

0.3M, 0.4

42. 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):
0.3M, 0.4

43. 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):
0.3M, 0.4

44. 05000498 - CNT_COMMAND Functionality Depends on CNT_IMASK Configuration:

DESCRIPTION:
The counter's "Zero Once" mode is only functional if at least one of the CZMZIE (Counter Zeroed by Zero Marker Interrupt Enable), CZMEIE (Zero Marker Error Interrupt), and/or CZMIE (CZM Pin Interrupt Enable/Push-Button Interrupt) bits in the CNT_IMASK register is set.

WORKAROUND:
If the counter is to be reset only on the first active level on the CZM pin, do all of the following:

1) Clear the ZMZC bit in the CNT_CONFIG register.
2) Set the W1ZMONCE bit in the CNT_COMMAND register.
3) Set the CZMZIE, CZMEIE, and/or CZMIE bits in the CNT_IMASK register.

As long as the SIC_IMASK register doesn't enable the counter interrupt, no action will be taken by the processor as a result of enabling one of these counter interrupts. If an alternate interrupt from the same counter is desired, software must ignore the extra interrupts resulting from the enable bit being set. To this regard, the CZMZIE interrupt is the most convenient to choose for this workaround.

Note that the Zero-marker-zeros-counter (ZMZC) mode is not affected by this anomaly.

APPLIES TO REVISION(S):
0.3M, 0.4

45. 05000500 - NFC Hang When AMC Requests Async Pins During Last 16 Bytes of Page Write:

DESCRIPTION:
The NFC controller on the BF54x shares certain pins with the asynchronous memory controller (AMC). The AMC pins have higher priority in a static arbitration scheme controlled by the APCM module of the EBIU. During an NFC page transfer, the AMC can acquire the shared pins at any point and release them after the AMC transfers are completed and when the NFC requests the pins again.

The problem occurs when the AMC requests the shared pins during the last 16-bytes of a NFC DMA page write transfer. If control of the pins are relinquished by the NFC in this window, the NFC never requests for the pins again and the DMA transfer consequently hangs. This is true regardless of whether the NFC is configured for 256-byte or 512-byte page writes. Simply detecting and re-enabling the DMA will not work since only 256 or 512 byte transfers are permitted.

The problem can occur irrespective of whether the AMC access is done using the core or DMA, or whether it is a read or a write access. Also, this problem applies only to page write transfers, and does not apply to page read transfers from the NAND flash.

WORKAROUND:
If simultaneous page writes through the NFC and accesses to the asynchronous memory banks are not required, or can be prevented in software, this problem will not occur.

However, there are some workarounds available if both NFC and AMC accesses are required -

1) If the access to the AMC takes place over DMA, these accesses can be held off till the NFC transfer is complete. This can be done by setting the TESTSETLOCK bit in the EBIU_FCTL register. This bit prevents DMA from getting access to the asynchronous memory banks. This bit can be cleared in the NFC DMA ISR to subsequently allow DMA accesses to the asynchronous memory banks. Keep in mind that NFC transfers can be megabytes in size. So this method should only be used if the AMC DMA accesses are low priority. Also ensure that this register is not modified when the AMC is in use.

2) Use core accesses to write to the NFC. Only DMA transfers stall due to this anomaly. However, ECC functionality of the NFC is lost when using core transfers for page writes.

3) This workaround is applicable only for NFC page size of 256, and only if the hang can be detected by some mechanism in software. Once the hang is detected, the NFC FIFO can be refilled by DMA if the page size setting in the NFC is switched to 512. The NFC page size can be reverted to 256 once the DMA count is past the critical count required for the failure to occur. For example, with a NFC page size of 256, the hang can only occur at DMA counts of 0xF4, 0xF8, and 0xFC. To implement this workaround, once application detects the hung state of NFC, change NFC page size from 256 to 512 and wait till NFC count reaches 0xFD, at which point NFC page size can be change back to 256.

APPLIES TO REVISION(S):
0.3M, 0.4

46. 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):
0.3M, 0.4

47. 05000502 - Async Memory Writes May Be Skipped When Using Odd Clock Ratios:

DESCRIPTION:
For certain CCLK::SCLK ratios, read or write accesses to asynchronous memory banks may be randomly dropped. The failure occurs with a CCLK::SCLK ratio of N::M, where N is odd OR M > 1 and odd. The only exception is when the CCLK::SCLK ratio is 1::1, and the failure will not occur at this operating point.

The failure is independent of other system parameters, or cache settings.

DMA accesses are not impacted by this anomaly.

WORKAROUND:
The possibly workarounds for this anomaly are:

1) Use only even CCLK::SCLK (2::1, 4::1 etc) ratios if core accesses to asynchronous memory banks are needed.
2) Use DMA accesses for the asynchronous memory banks.

APPLIES TO REVISION(S):
0.3M, 0.4

48. 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):
0.3M, 0.4

49. 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):
0.3M, 0.4

50. 05000508 - UART Receive DMA Hangs under Certain Conditions:

DESCRIPTION:
When the UART is in Receive DMA mode and the receive FIFO is empty, a permanent lock-up can occur if new UART RX data is received into the FIFO in the same SCLK cycle as the arrival of the internal DMA grant signal from the previous DMA read to get the last data from the UART_RBR register. When this precise timing is met, the UART does not clear the old data from the UART_RBR register, but it does properly clear the UART_LSR.DR status bit and the internal DMA grant signal. When this occurs, new data entering the UART RX FIFO will not be forwarded to the UART_RBR register, which prevents the UART from raising DMA data read requests and causes the UART RX DMA to hang.

WORKAROUND:
The cause for the internal DMA grant to the UART being delayed long enough for this condition to occur is the fact that the UART channel is the lowest priority in the DMA arbitration scheme and can be held off by other DMA channels enabled in the system. If the UART channel is given higher DMA priority, this condition can be avoided.

If prioritizing the UART DMA over other DMA activity is not an option, then the UART hang condition can be detected and corrected in software by checking the UART_MSR.RFCS and UART_LSR.DR status bits for an illegal combination that results from this anomaly. When the lock-up occurs, the UART_LSR.DR bit is properly cleared by the UART but the data from the FIFO isn't loaded to the UART_RBR register, thus forcing the UART RX FIFO to fill as new data comes in, eventually causing the UART_MSR.RFCS bit to set. Since the UART_MSR.RFCS bit indicates the status of the RX FIFO behind the UART_RBR register, it cannot be set if UART_LSR.DR = 0 (indicating that the UART_RBR register is empty). Once this invalid status is detected, a dummy core read of the UART_RBR register will clear the old data in the UART_RBR register, and the FIFO data will be forwarded properly. Hence, pending DMA read requests will be issued again, and the UART resumes operation:

```
short temp_data;

// Two reads with the following status is the hang condition
if((*pUART_MSR & RFCS) && (!(*pUART_LSR & DR)))   // RFCS = 1, DR = 0
{
  // Read status a 2nd time, perform corrective action if still true
  if((*pUART_MSR & RFCS) && (!(*pUART_LSR & DR)))
    temp_data = *pUART_RBR;  // dummy core read access to UART_RBR register
}
```

APPLIES TO REVISION(S):
0.3M, 0.4

51. 05000510 - USB Wakeup from Hibernate State Requires Re-Enumeration:

DESCRIPTION:
Applications may choose to place the processor in hibernate state when the USB suspend is seen on the bus. Logic was added at the PHY level to maintain the state of the USB data lines so that the processor will appear to still be connected to the host during the USB suspend state. This would prevent re-enumeration when the USB bus was subsequently resumed. However, when in hibernate, the USB controller also loses its logic state because power is removed. As a result, re-enumeration is unavoidable when coming out of the hibernate state.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3M, 0.4

52. 05000511 - Lower 16 Bits of CNT_COUNTER Register Do Not Update Properly:

DESCRIPTION:
If the CNT_COUNTER register is read in the same SCLK cycle that the counter is being incremented by hardware, the MMR read may incorrectly return the previous value in the lower 16 bits. Under most circumstances, this is nearly negligible, as the value of the 32-bit read would be only one less than the correct value. However, in the case of a 16-bit overflow, the value read from the counter will be off by 0xffff from the expected value, as the upper 16 bits do get properly updated. For example, if the current counter value is 0x0000ffff, the next counter increment should result in CNT_COUNTER containing the value 0x00010000. When this anomaly manifests, the value read will incorrectly be 0x0001ffff.

WORKAROUND:
There is no workaround for this anomaly when it occurs any time other than when the counter wraps at 16 bits. For the case where the value in the lower 16 bits is 0xFFFF, a second read of the CNT_COUNTER register will ensure that the correct value is read.

APPLIES TO REVISION(S):
0.4