SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF561 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.3	0.5
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x
2	05000099	UART Line Status Register (UART_LSR) Bits Are Not Updated at the Same Time	x	.
3	05000120	TESTSET Instructions Restricted to 32-Bit Aligned Memory Locations	x	x
4	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x
5	05000127	SIGNBITS Instruction Not Functional under Certain Conditions	x	.
6	05000149	IMDMA S1/D1 Channel May Stall	x	x
7	05000156	Timers in PWM-Out Mode with PPI GP Receive (Input) Mode with 0 Frame Syncs	x	.
8	05000166	PPI Data Lengths between 8 and 16 Do Not Zero Out Upper Bits	x	x
9	05000167	Turning SPORTs on while External Frame Sync Is Active May Corrupt Data	x	x
10	05000168	Undefined Behavior when Power-Up Sequence Is Issued to SDRAM during Auto-Refresh	x	.
11	05000169	DATA CPLB Page Miss Can Result in Lost Write-Through Data Cache Writes	x	.
12	05000171	Boot-ROM Modifies SICA_IWRx Wakeup Registers	x	.
13	05000174	Cache Fill Buffer Data lost	x	.
14	05000175	Overlapping Sequencer and Memory Stalls	x	.
15	05000176	Overflow Bit Asserted when Multiplication of -1 by -1 Followed by Accumulator Saturation	x	.
16	05000179	PPI_COUNT Cannot Be Programmed to 0 in General Purpose TX or RX Modes	x	.
17	05000180	PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs	x	x
18	05000181	Disabling the PPI Resets the PPI Configuration Registers	x	.
19	05000182	Internal Memory DMA Does Not Operate at Full Speed	x	x
20	05000184	Timer Pin Limitations for PPI TX Modes with External Frame Syncs	x	.
21	05000185	Early PPI Transmit when FS1 Asserts before FS2 in TX Mode with 2 External Frame Syncs	x	.
22	05000186	Upper PPI Pins Driven when PPI Packing Enabled and Data Length >8 Bits	x	.
23	05000187	IMDMA Corrupted Data after a Halt	x	x
24	05000188	IMDMA Restrictions on Descriptor and Buffer Placement in Memory	x	.
25	05000189	False Protection Exceptions when Speculative Fetch Is Cancelled	x	.
26	05000190	PPI Not Functional at Core Voltage < 1Volt	x	x
27	05000193	False I/O Pin Interrupts on Edge-Sensitive Inputs When Polarity Setting Is Changed	x	.
28	05000194	Restarting SPORT in Specific Modes May Cause Data Corruption	x	.
29	05000198	Failing MMR Accesses when Preceding Memory Read Stalls	x	.
30	05000199	Current DMA Address Shows Wrong Value During Carry Fix	x	.
31	05000200	SPORT TFS and DT Are Incorrectly Driven During Inactive Channels in Certain Conditions	x	.
32	05000202	Possible Infinite Stall with Specific Dual-DAG Situation	x	.
33	05000204	Incorrect Data Read with Writethrough "Allocate Cache Lines on Reads Only" Cache Mode	x	.
34	05000205	Specific Sequence that Can Cause DMA Error or DMA Stopping	x	.
35	05000207	Recovery from "Brown-Out" Condition	x	.
36	05000208	VSTAT Status Bit in PLL_STAT Register Is Not Functional	x	x
37	05000209	Speed Path in Computational Unit Affects Certain Instructions	x	.
38	05000215	UART TX Interrupt Masked Erroneously	x	.
39	05000219	NMI Event at Boot Time Results in Unpredictable State	x	.
40	05000220	Data Corruption/Core Hang with L2/L3 Configured in Writeback Cache Mode	x	.
41	05000225	Incorrect Pulse-Width of UART Start Bit	x	.
42	05000227	Scratchpad Memory Bank Reads May Return Incorrect Data	x	.

No.	ID	Description	0.3	0.5
43	05000230	UART Receiver is Less Robust Against Baudrate Differences in Certain Conditions	x	.
44	05000231	UART STB Bit Incorrectly Affects Receiver Setting	x	.
45	05000232	SPORT Data Transmit Lines Are Incorrectly Driven in Multichannel Mode	x	.
46	05000242	DF Bit in PLL_CTL Register Does Not Respond to Hardware Reset	x	.
47	05000244	If I-Cache Is On, CSYNC/SSYNC/IDLE Around Change of Control Causes Failures	x	.
48	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
49	05000248	TESTSET Operation Forces Stall on the Other Core	x	.
50	05000250	Incorrect Bit Shift of Data Word in Multichannel (TDM) Mode in Certain Conditions	x	.
51	05000251	Exception Not Generated for MMR Accesses in Reserved Region	x	.
52	05000253	Maximum External Clock Speed for Timers	x	.
53	05000254	Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock	.	x
54	05000257	Interrupt/Exception During Short Hardware Loop May Cause Bad Instruction Fetches	x	.
55	05000258	Instruction Cache Is Corrupted When Bits 9 and 12 of the ICPLB Data Registers Differ	x	.
56	05000260	ICPLB_STATUS MMR Register May Be Corrupted	x	.
57	05000261	DCPLB_FAULT_ADDR MMR Register May Be Corrupted	x	.
58	05000262	Stores To Data Cache May Be Lost	x	.
59	05000263	Hardware Loop Corrupted When Taking an ICPLB Exception	x	.
60	05000264	CSYNC/SSYNC/IDLE Causes Infinite Stall in Penultimate Instruction in Hardware Loop	x	.
61	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x
62	05000266	IMDMA Destination IRQ Status Must Be Read Prior to Using IMDMA	.	x
63	05000267	IMDMA May Corrupt Data under Certain Conditions	x	x
64	05000269	High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Increase	x	.
65	05000270	High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Decrease	x	.
66	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x
67	05000274	Data Cache Write Back to External Synchronous Memory May Be Lost	x	x
68	05000275	PPI Timing and Sampling Information Updates	x	x
69	05000276	Timing Requirements Change for External Frame Sync PPI Modes with Non-Zero PPI_DELAY	x	x
70	05000277	Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt	x	.
71	05000278	Disabling Peripherals with DMA Running May Cause DMA System Instability	x	.
72	05000281	False Hardware Error when ISR Context Is Not Restored	x	.
73	05000283	System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage	x	x
74	05000287	Reads Will Receive Incorrect Data under Certain Conditions	x	.
75	05000288	SPORTs May Receive Bad Data If FIFOs Fill Up	x	.
76	05000301	Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space	x	x
77	05000302	SSYNCs after Writes to DMA MMR Registers May Not Be Handled Correctly	x	x
78	05000305	SPORT_HYS Bit in PLL_CTL Register Is Not Functional	x	.
79	05000307	SCKELOW Bit Does Not Maintain State Through Hibernate	x	.
80	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
81	05000312	Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted	x	x
82	05000313	PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes	x	x
83	05000315	Killed System MMR Write Completes Erroneously on Next System MMR Access	x	x
84	05000320	PF2 Output Remains Asserted after SPI Master Boot	.	x
85	05000323	Erroneous GPIO Flag Pin Operations under Specific Sequences	x	x
86	05000326	SPORT Secondary Receive Channel Not Functional when Word Length >16 Bits	.	x
87	05000331	24-Bit SPI Boot Mode Is Not Functional	x	.

No.	ID	Description	0.3	0.5
88	05000332	Slave SPI Boot Mode Is Not Functional	x	.
89	05000333	Flag Data Register Writes One SCLK Cycle after Edge Is Detected May Clear Interrupt Status	x	.
90	05000339	ALT_TIMING Bit in PLL_CTL Register Is Not Functional	x	.
91	05000343	Memory DMA FIFO Causes Throughput Degradation on Writes to External Memory	x	.
92	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	x
93	05000362	Conflicting Column Address Widths Causes SDRAM Errors	x	x
94	05000363	UART Break Signal Issues	x	.
95	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x
96	05000371	Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration	x	x
97	05000403	Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall	x	x
98	05000412	TESTSET Instruction Causes Data Corruption with Writeback Data Cache Enabled	x	x
99	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x
100	05000425	Multichannel SPORT Channel Misalignment Under Specific Configuration	x	x
101	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
102	05000428	Lost/Corrupted L2/L3 Memory Write after Speculative L2 Memory Read by Core B	.	x
103	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x
104	05000458	SCKELOW Feature Is Not Functional	x	x
105	05000461	False Hardware Error when RETI Points to Invalid Memory	x	x
106	05000462	Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign	x	x
107	05000471	Boot Failure When SDRAM Control Signals Toggle Coming Out Of Reset	x	x
108	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x	x
109	05000475	Possible Lockup Condition when Modifying PLL from External Memory	x	x
110	05000477	TESTSET Instruction Cannot Be Interrupted	x	x
111	05000481	Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption	x	x
112	05000489	PLL May Latch Incorrect Values Coming Out of Reset	x	x
113	05000491	Instruction Memory Stalls Can Cause IFLUSH to Fail	x	x
114	05000494	EXCPT Instruction May Be Lost If NMI Happens Simultaneously	x	x
115	05000501	RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes	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-BF561 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):
0.3, 0.5

2. 05000099 - UART Line Status Register (UART_LSR) Bits Are Not Updated at the Same Time:

DESCRIPTION:

UART Line Status Register (UART_LSR) bits are not updated at the same time. Direct polling of the UART_LSR register may miss any of the Line Status conditions. This is because the UART_LSR register is read-to-clear, therefore, status bits may be cleared before the polling condition (typically DR is set) is met.

WORKAROUND:

When using polling mode, the SIC_ISR register should be polled to determine when it is safe to read UART_LSR/UART_RBR and write UART_THR. Below is a code example for determining when received data is ready and determining if there was a receive error. The UART_TX, UART_RX, and UART Error Interrupt are masked in SIC_IMASK for polling mode (no interrupt occurs).

```
#define IRQ_UART_RX 0x4000
#define IRQ_UART_ERROR 0x40

    p0.l = lo(UART_GCTL);
    p0.h = hi(UART_GCTL);

    p2.l = lo(SIC_ISR);
    p2.h = hi(SIC_ISR);

    r1 = PEN | WLS(8) (z);
    w[p0+UART_LCR-UART_GCTL] = r1;

    r1 = ERBFI | ELSI (z);
    w[p0+UART_IER-UART_GCTL] = r1;

receive_polling:
    r2 = w[p2] (z);
    CC = bittst (r2, bitpos (IRQ_UART_RX));
    if !CC jump receive_polling;

data_ready:
    csync;
    r1 = w[p0+UART_LSR-UART_GCTL] (z);
    r0 = w[p0+UART_RBR-UART_GCTL] (z);

    CC = bittst (r2, bitpos (IRQ_UART_ERROR));
    if CC jump error_handler;

    [i0++] = r0;
    jump receive_polling;
```

APPLIES TO REVISION(S):

0.3

3. 05000120 - TESTSET Instructions Restricted to 32-Bit Aligned Memory Locations:

DESCRIPTION:
TESTSET instructions on 16-bit aligned addresses produce wrong results.

WORKAROUND:
Use TESTSET only on memory locations that are 32-bit aligned. Use the ALIGN(4) directive if needed to place variables on 32-bit boundaries.

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):
0.3, 0.5

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

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

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

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

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

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices (VisualDSP++, VDK, 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):
0.3, 0.5

5. 05000127 - SIGNBITS Instruction Not Functional under Certain Conditions:

DESCRIPTION:
The SIGNBITS instruction, when operating on a 32-bit value, can sometimes operate incorrectly when the instruction preceding the SIGNBITS is creating the operand for the SIGNBITS, as in the following sequence:

```
R0 = ASHIFT R2 BY R3.L;
R1.L = SIGNBITS R0;
```

WORKAROUND:
There are two workarounds, which will avoid the problem:

1) Precede SIGNBITS instruction with a NOP:

```
R0 = ASHIFT R2 BY R3.L;
NOP;
R1.L = SIGNBITS R0;
```

2) Make sure the operand register for the SIGNBITS instruction is not dependent on the previous instruction:

```
R0 = ASHIFT R2 BY R3.L;
// **** another useful instruction that is not updating R0 ****;
R1.L = SIGNBITS R0;
```

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):
0.3

6. 05000149 - IMDMA S1/D1 Channel May Stall:

DESCRIPTION:
The Internal Memory DMA (IMDMA) engine has two pairs of Memory DMA channels called S0/D0 (higher priority) and S1/D1 (lower priority). The lower priority destination channel D1 can be held off by D0. The problem occurs when D0 is expecting data but never receives it because, for instance, an error condition in S0 or S0 not being enabled. In these cases, D0 will never get data, but it will also stall D1 indefinitely.

WORKAROUND:
Make sure that S0/D0 always completes without errors (for instance, with frequent checks on its status). If errors occur, disable both S0 and D0.

APPLIES TO REVISION(S):
0.3, 0.5

7. 05000156 - Timers in PWM-Out Mode with PPI GP Receive (Input) Mode with 0 Frame Syncs:

DESCRIPTION:
When PPI0 is in General Purpose Receive (Input) mode with 0 Frame Syncs, Timer 8 cannot be used in PWM-out mode.
When PPI1 is in General Purpose Receive (Input) modewith 0 Frame Syncs, Timer 10 cannot be used in PWM-out mode.

WORKAROUND:
You may use any other timer (0-7, 9, 11) instead for PWM output signals, or the timers associated with any unused PPI interface.

APPLIES TO REVISION(S):
0.3

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

9. 05000167 - Turning SPORTs on while External Frame Sync Is Active May Corrupt Data:

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

This anomaly also applies to Stereo Serial Modes (I2S and variants), except in the case where either the LRFS or the RRFST bit is set (not both).

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

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

APPLIES TO REVISION(S):
0.3, 0.5

10. 05000168 - Undefined Behavior when Power-Up Sequence Is Issued to SDRAM during Auto-Refresh:

DESCRIPTION:
Once the SDRAM has been powered up the first time and is in auto-refresh mode, the auto-refresh will run independently of subsequent power ups. This can cause an auto-refresh command to be sent to the SDRAM device at the same time as a power up (reprogramming of the SDRAM mode register). Undefined behavior can occur after this point.

Note: For most applications, the SDRAM powerup sequence and writing of the MODE register needs to be done only once. Once the powerup sequence has completed, the PSSE bit (SDRAM powerup sequence enable) should not be set again unless a change to the SDRAM Mode register is desired.

WORKAROUND:
If the SDRAM mode register must be reprogrammed (subsequent power up), the SDRAM should first be programmed in self-refresh mode.

General Procedure:
1) Issue an SSYNC instruction to ensure all pending memory operations have completed.
2) Set SDRAM to Self-Refresh mode by setting SRFS (SDRAM self-refresh enable) in EBIU_SDGCTL.
3) Issue an SSYNC instruction to ensure the write to the Self-Refresh bit has completed.
4) Reprogram EBIU_SDRRC and EBIU_SDBCTL, as needed.
5) In EBIU_SDGCTL, set PSSE (SDRAM powerup sequence enable) and clear SRFS (SDRAM self-refresh disable).
6) Issue an SSYNC instruction.

APPLIES TO REVISION(S):
0.3

11. 05000169 - DATA CPLB Page Miss Can Result in Lost Write-Through Data Cache Writes:

DESCRIPTION:
When Data Cache is enabled and a CPLB miss occurs during a write to a cache line in write-through (not write-back) mode, the line is erroneously marked as "dirty". This is never expected in write-through mode, since the data will immediately be written to the memory.

This erroneous marking may cause a missed write if
1) the erroneously marked line is victimized
AND
2) a write to the same line occurs before the victimized line is sent to L2 memory
AND
3) the write in 2) is delivered to L2 before the victimized line in 1). This may happen due to scheduling priorities of the transfers.

WORKAROUND:
1) Avoid DATA CPLB page misses when writing to write-through cacheable L2 locations currently cached
OR
2) Before swapping out a DATA CPLB page configured as write-through, FLUSHINV (with a final SSYNC) all write-through cached lines losing their CPLB descriptor.

This problem does not occur with write-back mode of the cache system.

APPLIES TO REVISION(S):
0.3

12. 05000171 - Boot-ROM Modifies SICA_IWRx Wakeup Registers:

DESCRIPTION:
The Boot ROM is executed at power up/reset and changes the value of the SICA_IWR registers from their default reset value of 0xFFFF, but does not restore them.

WORKAROUND:
User code should not rely on the default value of these registers. Set the desired values explicitly.

APPLIES TO REVISION(S):
0.3

13. 05000174 - Cache Fill Buffer Data lost:

DESCRIPTION:
With Data Cache enabled, data scheduled for writing in the fill buffer will not be written if this exact - and unlikely - sequence of events is encountered:
1) A dual-DAG instruction (two memory transfers in the same instruction) must access the same sub-bank in memory (collision) with no cache misses

2) The instruction immediately following must be a write that hits a cache line that is being filled AND this fill must be just being completed in that same cycle

3) A new fill is initiated that reuses the same fill buffer.

WORKAROUND:
Avoid this exact sequence of events.

Workarounds include, but are not limited to:

1) Avoid collisions by placing data in separate sub-banks
OR
2) Insert a nop or any other non-write instruction between the two instructions described in the offending sequence above
OR
3) Convert the single DAG write from the second instruction described above to a dual-DAG instruction with a dummy read from the same address (in this case, of course, make sure that the next instruction does not cause the same sequence)

APPLIES TO REVISION(S):
0.3

14. 05000175 - Overlapping Sequencer and Memory Stalls:

DESCRIPTION:
When Sequencer stalls (i.e., arithmetic stalls caused by register dependencies or certain MMR register accesses) coincide exactly in time with memory stalls (i.e., dual-DAG collisions into the same sub-bank of memory, cache misses, etc.), there is a small likelihood that subsequent transactions may be lost. The stalls must exactly coincide, meaning that the start cycle and duration of the stalls must be identical.

WORKAROUND:
The easiest workaround for this issue is to limit the stalls on the processor side by not programming compute stalls and by surrounding MMR accesses with CSYNC instructions.

A better approach is to analyze the stalls with the following method, using the on-chip performance monitor:

The following code can be used to count the number of overlapping core stalls and memory freeze cycles. This does not necessarily indicate that a problem will result because the durations may be different. For example, compute stalls generally last only a cycle or two, and a memory freeze resulting from a cache miss will last many cycles - so, this could not cause a problem, but it would still increment the counter.

```
// Set up performance counters to monitor overlapping stalls
r0.h=0x0091;
r0.l=0xd21b;
p0.h=PFCTL;
p0.l=PFCTL;
[p0]=r0;                // wake up first
[p0]=r0;                // then enable

p0.h=PFCNTR0;
p0.l=PFCNTR0;
r0=0;
[p0++]=r0;              // clear both counters
[p0]=r0;
```

After this, the secondary performance counter (PFCNTR1) counts the number of cycles of overlapping stalls.

One way of finding stalls in existing code is to initialize the second counter with 0xffffffff:

```
r0.h=0xffff;
r0.l=0xffff;
p0.h=PFCNTR1;
p0.l=PFCNTR1;
[p0]=r0;
```

When the counter overflows, this will trigger a hardware error condition, which is interrupt 5. Put a breakpoint in the interrupt 5 routine, which can consist of only the rti instruction. Step back to the interrupted routine, and the stall should be a few lines of code above the return point. Insert nop instructions to eliminate compute stalls, and insert CSYNCs to avoid MMR stalls. Repeat this procedure until the performance counter remains zero.

APPLIES TO REVISION(S):
0.3

15. 05000176 - Overflow Bit Asserted when Multiplication of -1 by -1 Followed by Accumulator Saturation:

DESCRIPTION:
A multiply of -1 by -1 in 1.15 format, followed by an accumulator saturation instruction on a 32-bit boundary, may cause the saturation instruction to incorrectly set the overflow bit. However, the numeric result of the accumulator(s) is still correct.

For example, the following instruction sequence will show this behavior:

```
R2 = 0;
A0 = 0;              // Clear accumulator
R0 = 0x8000 (Z);     // -1 in 1.15 format
R1 = R0;

A0 -= R0.L * R1.L; // Multiply -1 by -1 and subtract from accumulator
ASTAT = R2;        // Clear ASTAT
A0 = A0 (S);       // Will incorrectly set the overflow bit
```

WORKAROUND:
You can either ignore the overflow bit after such a instruction sequence OR add a dummy accumulate instruction after the multiplication of -1 by -1. For instance, assuming that R3 contains zero and the initial conditions are the same as in the example:

```
A0 -= R0.L * R1.L ; // Multiply -1 by -1 and subtract from accumulator
A0 += R3.L * R3.L;  // Dummy add zero to previous result
ASTAT = R2;         // Clear ASTAT
A0 = A0 (S);        // Will now correctly clear the overflow bit
```

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):
0.3

16. 05000179 - PPI_COUNT Cannot Be Programmed to 0 in General Purpose TX or RX Modes:

DESCRIPTION:
In General Purpose modes, the PPI must receive or transmit blocks of at least 2 words. Single word transfers (PPI_COUNT value of 0) are not functional.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3

17. 05000180 - PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs:

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

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

APPLIES TO REVISION(S):
0.3, 0.5

18. 05000181 - Disabling the PPI Resets the PPI Configuration Registers:

DESCRIPTION:
Upon disabling the PPI, all PPI MMRs return to their reset values.

WORKAROUND:
Re-program all relevant MMRs each time the PPI is disabled and then re-enabled.

APPLIES TO REVISION(S):
0.3

19. 05000182 - Internal Memory DMA Does Not Operate at Full Speed:

DESCRIPTION:
The Internal Memory DMA controller only operates up to a 500MHz core clock frequency. Clock frequencies above this limit may cause unexpected behavior.

WORKAROUND:
If the use of IMDMA is required, set the core clock frequency to a maximum of 500MHz.

APPLIES TO REVISION(S):
0.3, 0.5

20. 05000184 - Timer Pin Limitations for PPI TX Modes with External Frame Syncs:

DESCRIPTION:
In the transmit modes with external frame syncs, the timer pins associated with the Frame Sync(s) (Timer 8 and/or 9 for PPI0, Timer 10 and/or 11 for PPI1) must be configured as inputs. The Timer(s) must be enabled and are not avaialable for general use.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3

21. 05000185 - Early PPI Transmit when FS1 Asserts before FS2 in TX Mode with 2 External Frame Syncs:

DESCRIPTION:
If FS1 asserts before FS2 has asserted, the PPI does not wait for assertion of FS2 prior to transmitting data.

WORKAROUND:
Use either one external frame sync mode or two internally generated frame syncs, if possible.

APPLIES TO REVISION(S):
0.3

22. 05000186 - Upper PPI Pins Driven when PPI Packing Enabled and Data Length >8 Bits:

DESCRIPTION:
When packing of PPI data is enabled, but the data length (DLEN) of the PPI port is set to values of 10 bits or greater, the higher bits (and pins) in excess of 8 may be driven (to either zeroes or ones).

WORKAROUND:
Since packing is meaningful only on 8-bit data, this is easily avoided by setting the proper data length of 8 bits.

APPLIES TO REVISION(S):
0.3

23. 05000187 - IMDMA Corrupted Data after a Halt:

DESCRIPTION:
During an IMDMA transfer, if software halts the destination channel before the associated source channel, then a subsequent transfer may either get wrong data or the data from the previous transfer.

This problem does not occur when the order of halting is inverted or if the DMA channel is halted due to an error condition.

WORKAROUND:
When disabling the IMDMA transfer, disable the source channel first.

APPLIES TO REVISION(S):
0.3, 0.5

24. 05000188 - IMDMA Restrictions on Descriptor and Buffer Placement in Memory:

DESCRIPTION:
In descriptor mode, there are certain cases where IMDMA can behave unexpectedly:
1) Data buffer in fast (L1) memory, descriptor(s) in slow (external) memory
2) Data buffer and descriptors in L1 memory, but in different banks
3) Two IMDMA channels operating on data in different L1 banks

WORKAROUND:
The described cases are generally avoided by taking care of the placement:
1) Place the descriptor(s) in L1 (same bank as data buffer) or L2 memory
2) Place data buffer and descriptors in the same banks of L1
3) Place the data buffer and descriptors of all channels in the same bank of L1

APPLIES TO REVISION(S):
0.3

25. 05000189 - False Protection Exceptions when Speculative Fetch Is Cancelled:

DESCRIPTION:
A false Code or Data Protection Exception may be raised if it is caused by a speculative fetch that is canceled. For instance, if a JUMP or an RTS instruction located at the last word of a valid page branches to another valid page, but the speculative instruction fetch was from an invalid or non-existing memory location, an exception would still be raised. The same applies for interrupts, where executing an RTI instruction results in program flow continuing from the point preceding an instruction that was previously speculatively fetched (which generated the false protection exception). Similarly, if a post-incremented indirect data memory access performs a speculative access to a protected or non-existing memory location (e.g., R0 = [P0++]; where P0 points to the last word of a bank) an incorrect exception would occur.

WORKAROUND:
1) Do not place branch instructions or data at bank boundaries. Leave at least 76 bytes free before any boundary with a reserved memory space. This will prevent false exceptions from occurring.

2) Have the exception handler confirm whether the exception was valid or not before taking action. This can be done by verifying if the CODE_FAULT_ADDR (or the DATA_FAULT_ADDR) register contains an address that is within a valid page. In that case, no action is performed.

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):
0.3

26. 05000190 - PPI Not Functional at Core Voltage < 1Volt:

DESCRIPTION:
The PPI may malfunction if the core voltage is set to levels below 1 Volt.

WORKAROUND:
Set the core Voltage to levels above or equal to 1 Volt.

APPLIES TO REVISION(S):
0.3, 0.5

27. 05000193 - False I/O Pin Interrupts on Edge-Sensitive Inputs When Polarity Setting Is Changed:

DESCRIPTION:
Consider the following scenario:

1) Pins are configured as edge-sensitive inputs.

2) The interrupt occurs on the rising edge.

3) Input level is constant and 0.

4) Change the polarity setting to set the interrupt to occur on the falling edge instead.

In this case, an erroneous interrupt will occur, even though no edge was physically present at the input. This will also occur at any subsequent writes of a 1 to this bit of the polarity register. If the polarity register is reset to 0, no interrupt is generated, as expected.

In the opposite case, with the external pin level equal to 1, the erroneous interrupt is generated when the polarity bit is changed from 1 to 0 (and any subsequent writes of a 1 to this bit of the polarity register), and not when changed from 0 to 1.

In the case of multiple I/O pins configured as edge-sensitive interrupts, ANY change to the polarity register will affect all those I/O pins in the above manner. The workaround in this case needs to be applied to all of those pins.

Similar considerations apply to the input enable register. Changing this setting while edge-sensitive interrupts are enabled will also cause unwanted interrupts.

WORKAROUND:
Prior to changing the polarity (and/or input enable) register(s), disable the interrupts (i.e., by clearing the PFA or PFB IMASK bit), change the register setting, clear the interrupt request as you normally would (write to the data or clear registers), and then re-enable the interrupt again.

APPLIES TO REVISION(S):
0.3

28. 05000194 - Restarting SPORT in Specific Modes May Cause Data Corruption:

DESCRIPTION:
When using internal SPORT clocks or external SPORT clocks with falling edge selected (SPORTx_TCR1:TCKE, SPORTx_RCR1:RCKE), if the SPORT is disabled and then subsequently re-enabled, there is a possibility of corrupting the first word transmitted or received.

WORKAROUND:
For internal SPORT clocks:
After disabling the SPORT, write bit 1 of the SPORT configuration 1 register (SPORTx_TCR1:ITCLK, SPORTx_RCR1:IRCLK) to 0. This will switch the SPORT clock to external, allowing the SPORT to be fully reset.

For external SPORT clocks with falling edge selected:
After disabling the SPORT, write bit 14 of the SPORT configuration 1 register (SPORTx_TCR1:TCKE, SPORTx_RCR1:RCKE) to 0. This will allow the SPORT to be fully reset.

APPLIES TO REVISION(S):
0.3

29. 05000198 - Failing MMR Accesses when Preceding Memory Read Stalls:

DESCRIPTION:
If an MMR read immediately follows a stalled memory read, the MMR read will fail (i.e., the wrong data will be read). Likewise, if an MMR write immediately follows a stalled memory read, the write may not take place (will be lost).

Instructions that include a memory read are:

```
reg = [ Preg ], etc.
reg = [ Ireg ], etc.
stack pop, stack pop multiple
UNLINK
TESTSET
PREFETCH
```

and any of the above in parallel issue.

Instructions that include a memory write are:

```
[ Preg ] = reg, etc.
[ Ireg ] = reg, etc.
stack push, stack push multiple
LINK
```

and any of the above in parallel issue.

WORKAROUND:
Placing a NOP (or any non-memory access) in front of the MMR access will prevent this problem.

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):
0.3

30. 05000199 - Current DMA Address Shows Wrong Value During Carry Fix:

DESCRIPTION:
If the DMA Current Address register (DMAx_CURR_ADDR) of an active channel is read during the carry fix cycle, then the upper half of the register will be off by one. The LSBs will have been updated with the new value, while the MSBs will still have the previous value. A second read of the register will return the correct value.

The carry fix cycle occurs when the DMA address is being modified such that the address crosses a 64k boundary. If the DMA address cannot cross a 64k address boundary, the read will never be incorrect.

WORKAROUND:
1) Avoid DMA addresses that cross a 64K address boundary.
OR
2) Read the DMA Current Address register twice to verify value read.

APPLIES TO REVISION(S):
0.3

31. 05000200 - SPORT TFS and DT Are Incorrectly Driven During Inactive Channels in Certain Conditions:

DESCRIPTION:
In multichannel mode, the SPORT's MRCS registers are used to select which channels are active (transmitted or received) and which ones are to be ignored. For each ignored channel, the DT output will be three-stated. The problem is seen when "Multichannel DMA packing" is disabled. In this mode, for the inactive channels, the most significant BIT (MSB) of the data word will be driven on the DT line and, correspondingly, the TFS will be driven high for the duration of one bit to indicate valid data.

WORKAROUND:
A possible workaround is to enable "Multichannel DMA packing". In this mode, only active channels are DMA'ed from/into memory, and the inactive channels will effectively be blanked out. However, in this mode, it is not possible to dynamically change the active/inactive channels as in the non-packed DMA mode, so it may not be feasible for all applications.

APPLIES TO REVISION(S):
0.3

32. 05000202 - Possible Infinite Stall with Specific Dual-DAG Situation:

DESCRIPTION:
For this problem to occur, the processor must perform a data memory read of a non-cacheable or write-through cacheable L2 or external memory address that was recently written. The "recent write" must currently be held in the write buffer when it is read again. The read must be executed during the same clock cycle that this "recent write" drains from the write buffer to the destination memory location.

Immediately prior to the read of the "recent write", a dual-DAG access must be executed. Dual-DAG accesses must collide with each other, but may have any relationship with the "recent write" address.

Immediately following the read of the "recent write", there must not be a read, write, or prefetch (else this anomaly is avoided).

Note: A dual-DAG collision is if both DAGs access the same sub-bank in L1 memory or L1 cache.

WORKAROUND:
1) Avoid reads of recently written L2 addresses immediately after colliding dual-DAG accesses.
OR
2) Precede the offensive dual-DAG/L2 read instructions with a SSYNC to insure the previous write is no longer in the write buffer.
OR
3) Configure all L2 as write-back cacheable (avoids using the write-buffers).

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):
0.3

33. 05000204 - Incorrect Data Read with Writethrough "Allocate Cache Lines on Reads Only" Cache Mode:

DESCRIPTION:
When Writethrough cache is enabled and DCPLB_DATAx:CPLB_L1_AOW = 0 (Allocate Cache Lines on Reads Only), incorrect data may be read in the following scenario:
• Must write to an address which is writethrough cacheable in no-allocate-on-write mode, and cache miss must occur
• Must read above address while write is still in Store buffer (not yet in Write buffer or destination memory location)
• Must then read above address again after it drains from Store buffer. Data returned from cache will be incorrect but the destination memory location will have correct value.

WORKAROUND:
When configuring data cache as writethrough, set DCPLB_DATAx:CPLB_L1_AOW = 1 (allocate cache lines on reads and writes) to avoid the possibility of this anomaly.

APPLIES TO REVISION(S):
0.3

34. 05000205 - Specific Sequence that Can Cause DMA Error or DMA Stopping:

DESCRIPTION:
The problem can occur when 3 or more consecutive DMA reads from a single L1 memory bank (Bank A or Bank B) are stalled for an extended period of time (due to core or cache activity). It is irrelevant which DMA controller is requesting the reads, or in which order they occur. In this situation, when a subsequent L1 DMA read is initiated from a different bank, it is possible that the data from the last 2 reads will collide, thus corrupting the data.

WORKAROUND:
Locate all DMA data within the same L1 memory bank (or split DMA data amongst both cores, if possible).

APPLIES TO REVISION(S):
0.3

35. 05000207 - Recovery from "Brown-Out" Condition:

DESCRIPTION:
When a "brown-out" occurs, the internal Voltage regulator cannot be reset using the hardware reset pin. A "brown-out" is defined as a condition in which VDDext drops below the range specified in the data sheet, but does not drop all the way to 0 V, before it returns to the proper value.

WORKAROUND:
In order to recover from a "brown-out", the processor must be powered down completely and then powered back up.

APPLIES TO REVISION(S):
0.3

36. 05000208 - VSTAT Status Bit in PLL_STAT Register Is Not Functional:

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

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

APPLIES TO REVISION(S):
0.3, 0.5

37. 05000209 - Speed Path in Computational Unit Affects Certain Instructions:

DESCRIPTION:
The following instructions can sometimes operate incorrectly when the preceding instruction is creating the operand for the instruction.

The affected instructions are:

```
EXTRACT (x)
DEPOSIT (x)
SIGNBITS
EXPADJ
```

An example is shown for the SIGNBITS instruction:

```
R0 = ASHIFT R2 BY R3.L;
R1.L = SIGNBITS R0;
```

WORKAROUND:
There are two workarounds that will avoid the problem:

1) Precede SIGNBITS instructions with a NOP:

```
R0 = ASHIFT R2 BY R3.L;
NOP;
R1.L = SIGNBITS R0;
```

2) Make sure the operand register for the SIGNBITS is not dependent on the previous instruction:

```
R0 = ASHIFT R2 BY R3.L;
// ** another useful instruction that is not updating R0 **;
R1.L = SIGNBITS R0;
```

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):
0.3

38. 05000215 - UART TX Interrupt Masked Erroneously:

DESCRIPTION:
During a UART TX interrupt, if the IIR register is read and the THR register is not written to (to clear the TX interrupt), the UART TX interrupt is masked. This can happen in an ISR if the end of the string is reached. In this case, disabling and enabling the ETBEI bit has no effect on the state of the interrupt enable, but it should.

WORKAROUND:
Clear the ETBEI bit within the UART TX interrupt to end a string, and then execute an RTI. To re-enable the interrupt, simply set the ETBEI bit. This will take the processor directly into the UART TX interrupt service routine when the THR register is empty.

APPLIES TO REVISION(S):
0.3

39. 05000219 - NMI Event at Boot Time Results in Unpredictable State:

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

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

APPLIES TO REVISION(S):
0.3

40. 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 or an access by the other core. 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):
0.3

41. 05000225 - Incorrect Pulse-Width of UART Start Bit:

DESCRIPTION:
The duration of the start-bit in a word transmitted by the UART interface is incorrect.

For Clock Divisor values greater than 1 (as determined by the UART_DLL and UART_DLH registers), the pulse width can assume values of 14/16th or 15/16th of the nominal bit time.

Data, Parity and Stop bits have proper duration. The data will be correctly received.

See anomalies 05000230 and 05000231 regarding UART timing.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3

42. 05000227 - Scratchpad Memory Bank Reads May Return Incorrect Data:

DESCRIPTION:
Reads from the scratchpad memory may return incorrect data under some conditions. The problem occurs when reads of scratchpad memory are immediately followed by another read (of any location, including non-scratchpad locations), where the addresses being accessed do not have the same least significant address bit. This means that one of the transfers has to be a byte access. The other access has to be either a byte, 16-bit or 32-bit access on a different byte boundary than the first access. In addition, the instruction immediately before the Scratchpad memory read has to generate a memory stall due to either a dual DAG bank collision, a non-L1 memory data fetch, or a cache-line fill.

If an instruction does not perform a read immediately after the scratchpad read, no problems occur. Also, back to back non-byte reads function properly.

WORKAROUND:
The simplest workaround for assembly programmers is to place any non-read instruction after each scratchpad read. For C programmers, one solution is to avoid mapping any data to the scratchpad.

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):
0.3

43. 05000230 - UART Receiver is Less Robust Against Baudrate Differences in Certain Conditions:

DESCRIPTION:
In asynchronous communications, transmitter and receiver bit clocks can differ to a certain percentage of the nominal value. The exact amount is dependent upon the configuration of the word to be transmitted and other external factors such as signal quality.

For the Blackfin UART receiver, the tolerance is different when its bit-clock is slower or faster than the sender's clock. The Blackfin UART receiver will tolerate differences well when the transmitting side (sender) is operating at slightly lower bit rates. If, however, the sender is operating at slightly higher bit rates than the Blackfin receiver, the communication may fail for back-to-back transfers (i.e. no gaps between the words).

The reason for this is that the receiver, as per the standard implementation, samples the stop-bit, like any other bit, 16 times before it is ready to detect a new start bit condition. Since the decision of whether a stop-bit has been detected is done after the 9th sample, the receiver should immediately be ready for the next word if a stop-bit is detected. Instead, the Blackfin receiver will take the remaining 7 samples as well.

The effect of this is that the sampling error may accumulate for the kind of data transfers described above.

The anomaly has no effect on single transfers or transfers with gaps between the words, as the sampling error will not accumulate.

WORKAROUND:
The sender should operate at a lower (or identical) bit-rate.

If this cannot be guaranteed (and if possible), configure the sender to transmit more than one stop-bit, thus inserting the necessary gaps between words.

If both the sender and the receiver are Blackfin devices in a bidirectional communication channel, the above workarounds will not resolve this issue. This is due to anomalies 05000225 and 05000231.

APPLIES TO REVISION(S):
0.3

44. 05000231 - UART STB Bit Incorrectly Affects Receiver Setting:

DESCRIPTION:
The STB bit controls how many stop-bits are generated by the transmitter.

However, this setting also incorrectly affects how many stop-bits are sampled by the receiver. The correct behavior is for the receiver to always sample and test one stop-bit. However, the receiver will sample and test the number of stop-bits set by the STB bit. This incorrect behavior also affects framing error detection.

Note that this anomaly makes the workaround for anomaly 05000230 not applicable to the case of a bidirectional link composed of two Blackfin devices.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3

45. 05000232 - SPORT Data Transmit Lines Are Incorrectly Driven in Multichannel Mode:

DESCRIPTION:
In multichannel mode, the SPORT's MRCS registers are used to select which channels are active (transmitted or received) and which ones are to be ignored. For each ignored channel, the DT output should be tri-stated. This is not the case. The DT output as actively driven instead.

WORKAROUND:
This only affects applications where multiple SPORTs' transmitters are sharing the same line. External glue logic is required in that case.

APPLIES TO REVISION(S):
0.3

46. 05000242 - DF Bit in PLL_CTL Register Does Not Respond to Hardware Reset:

DESCRIPTION:
If the DF bit is set prior to a hardware reset, the PLL will continue to divide CLKIN by 2 after the hardware reset, but the DF bit itself will be cleared in the PLL_CTL register.

WORKAROUND:
Reprogram the PLL with DF cleared if the desire is to not divide CLKIN by 2 after reset.

APPLIES TO REVISION(S):
0.3

47. 05000244 - If I-Cache Is On, CSYNC/SSYNC/IDLE Around Change of Control Causes Failures:

DESCRIPTION:

When instruction cache is enabled, a CSYNC/SSYNC/IDLE around a Change of Control (including asynchronous exceptions/interrupts) can cause unpredictable results.

An example of the most common sequence that can cause this issue consists of a BRCC (NP) followed by CSYNC/SSYNC/IDLE anywhere in the next three instructions:

```
BRCC X [predicted not taken]
NOP
NOP
CSYNC/SSYNC/IDLE  // this instruction is bad in any of the 3 instructions following BRCC X
```

Another sequence that would encounter this problem would be if a BRCC (BP) which points to a CSYNC/SSYNC/IDLE is followed by a stalling instruction that allows the speculatively fetched CSYNC/SYNC/IDLE to "catch up" to the BRCC to within two cycles:

```
BRCC X (BP)
Y: ...
   ...
X: CSYNC/SSYNC/IDLE
```

This sequence is extremely difficult to reproduce with a failure. It requires an exact combination of stalls before the BRCC along with some very specific cache behavior.

WORKAROUND:

Turning the instruction cache off is one way to avoid the anomaly.

Assembly code must avoid the above scenarios. For all cases not related to asynchronous events, NOPs should be inserted to avoid the anomaly condition described:

```
IF CC JUMP ...;                          IF CC JUMP X (BP);
NOP;           // 3 NOP Pads             ...
NOP;                                     ...
NOP;                                     X: NOP; // NOP Pad at Jump Target
CSYNC/SSYNC/IDLE;                        CSYNC/SSYNC/IDLE;
```

For asynchronous interrupt events, the SSYNC/CSYNC/IDLE instruction can be protected by disabling interrupts and padding the SSYNC/CSYNC/IDLE with 2 leading NOPs:

```
CLI R0;
NOP; NOP;            // 2 Padding Instructions
CSYNC/SSYNC/IDLE
STI R0;
```

For exceptions, 3 padding NOPs should be implemented following any access to a cacheable region of memory.

Finally, as the workaround involves Supervisor Mode instructions to disable and enable interrupts, this does not apply to User Mode. In user space, do not use CSYNC or SSYNC instructions.

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):

0.3

48. 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):

0.3, 0.5

49. 05000248 - TESTSET Operation Forces Stall on the Other Core:

DESCRIPTION:
When one core performs a TESTSET operation to internal L2 memory, it locks out the other core if the latter is accessing L2 memory at the same time.

WORKAROUND:
Any TESTSET operation to internal L2 memory should be followed immediately by a (dummy) write to L2 memory. To ensure that the dummy write is performed immediately after the TESTSET operation, these two instructions should be performed in a critical region, as follows:

```
CLI R0;
NOP; NOP;
TESTSET(P0);
W[P1] = R0;   // P1 Points to L2 Memory
STI R0;
```

Note that the need for a critical code region implies that this workaround is not applicable in user space. No workaround exists in user space outside of avoiding the conditions mentioned.

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):
0.3

50. 05000250 - Incorrect Bit Shift of Data Word in Multichannel (TDM) Mode in Certain Conditions:

DESCRIPTION:
In multichannel mode, when the period of the frame sync is bigger than the actual data frame width by ONE bit (i.e. there is one "inactive bit"), the FIRST word of the transmitted frame is shifted to the left by one bit and the LSB is the MSB of the second word. All other words are transmitted correctly.

All data is transmitted correctly if the Frame Sync period is equal to the actual frame width or bigger by more than 1 bit.

For example, if there are 8 words of 16-bit data each, that would be 128 bits in the data frame.

If RFSDIV = 127 --> all data words are CORRECT
If RFSDIV = 128 --> first word is INCORRECT
If RFSDIV = 129 --> all data words are CORRECT
If RFSDIV = 130 --> all data words are CORRECT

WORKAROUND:
Set the RFSDIV register value to the number of data bits +/- 1 to avoid the case described above.

APPLIES TO REVISION(S):
0.3

51. 05000251 - Exception Not Generated for MMR Accesses in Reserved Region:

DESCRIPTION:
The region at addresses 0xFFC01900 - 0xFFC01AFF is reserved in the system MMR memory space. However, accesses to this area will not generate hardware error interrupts.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3

52. 05000253 - Maximum External Clock Speed for Timers:

DESCRIPTION:

The General-Purpose Timers can generate PWM output waveforms on the TMRx pin whose timing is quantified in either system clock (SCLK) periods or in periods of an externally supplied clock (TMRCLK or TACLK). For proper operation, SCLK must be faster than the source that is utilized, TMRCLK or TACLK.

The specification in the data sheet and hardware reference manual allows for TMRCLK and TACLK speeds of up to 1/2 SCLK.

However, the maximum rate is less than this limit. The minimum SCLK/TMRCLK or SCLK/TACLK ratio is somewhere in the range of 2.5 to 2.7. The exact value is not yet characterized.

WORKAROUND:

A minimum SCLK/TMRCLK or SCLK/TACLK ratio of 3 is safe to use.

APPLIES TO REVISION(S):

0.3

53. 05000254 - Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock:

DESCRIPTION:

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

WORKAROUND:

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

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

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

APPLIES TO REVISION(S):

0.5

54. 05000257 - Interrupt/Exception During Short Hardware Loop May Cause Bad Instruction Fetches:

DESCRIPTION:
Unpredictable behavior can result when hardware loops shorter than 4 instructions in length are interrupted at the end of the loop due to an interrupt or exception. In this situation, the processor's loop buffers, which are used to reduce the instruction fetch latency, operate incorrectly, resulting in the wrong instructions being fetched as the loop exits.

WORKAROUND:
There are a few possible workarounds for this anomaly. The first is to clear the loop buffers by writing to the Loop Counter registers (LC0 and LC1) inside all interrupt/exception handlers:

```
R0 = LC0;
LC0 = R0;
R0 = LC1;
LC1 = R0;
```

A second idea would be to include the loop counters in the context switch code:

```
[--SP] = LC0;
[--SP] = LC1;

<interrupt code>

LC1 = [SP++];
LC0 = [SP++];
```

Finally, another workaround would be to pad the loop with NOPs to increase the loop length to greater than or equal to 4 instructions.

Alternatively, if the event handlers use hardware loops, the above steps are not required, since every time an LCx register is written to, its corresponding loop buffer is cleared.

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):
0.3

55. 05000258 - Instruction Cache Is Corrupted When Bits 9 and 12 of the ICPLB Data Registers Differ:

DESCRIPTION:
When bit 9 and bit 12 of the ICPLB Data MMR differ, the cache may not update properly. For example, for a particular cache line, the cache tag may be valid while the contents of that cache line are not present in the cache.

WORKAROUND:
Set bit 9 to the state of bit 12 in each ICPLB entry.

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):
0.3

56. 05000260 - ICPLB_STATUS MMR Register May Be Corrupted:

DESCRIPTION:
The ICPLB Status register cannot be relied upon to determine which CPLB caused an exception. This register is corrupted if:

1) There is a jump to anywhere within the last 64 bits of a page (as defined by an ICPLB), AND
2) An instruction located within these last 64 bits generates an instruction exception, AND
3) Speculative instruction fetches increment into the next page and encounter another instruction exception cause.

When all of these criteria are met, ICPLB_STATUS will reflect the speculative instruction fetch rather than the initial exception cause.

WORKAROUND:
Handle instruction protection violations and ICPLB multiple hits without using this register.

Use the ICPLB_FAULT_ADDR register to see the address that caused the exception:
1) For CPLB misses, exceptions simply swap in a CPLB entry that covers the address in question.

2) For the case of multiple CPLB hits, use the ICPLB_FAULT_ADDR register to find out which address caused the exception and then iterate through all the CPLB entries to see which of the CPLBs cover the fault address.

3) For a protection violation exception, the handling is user-specific.

APPLIES TO REVISION(S):
0.3

57. 05000261 - DCPLB_FAULT_ADDR MMR Register May Be Corrupted:

DESCRIPTION:
The DCPLB_FAULT_ADDR MMR register may be corrupted. For this to happen, an aborted data memory access must generate BOTH a protection exception and a stall (due to either a dual-DAG collision, addressing of cacheable memory and missing, or simply fetching from L2).

WORKAROUND:
1) Immediately return from the data exception handler upon an initial entry into the handler (without any servicing yet), and then trust the DCPLB_FAULT_ADDR upon a second pass through the same data CPLB exception handler. Unless the cause is an exception that is serviced, the exception will be regenerated and cause a second pass. In the second pass, however, the DCPLB_FAULT_ADDR register will be correct because it is never generated incorrectly immediately after returning from an exception handler. To ensure that the same exception is being responded to in the second pass (rather than a higher priority exception), a copy of the RETX register should be acquired in the first pass and compared against in the second pass.

OR

2) Be tolerant of the artifacts generated by misprocessing the exception. For the three types of data memory exceptions - protection violation, CPLB miss, and CPLB multiple hit - the recommended software workaround is as follows:

a) For data protection exceptions, use the DCPLB_STATUS register rather than the DCPLB_FAULT_ADDR register in the handler. This will provide the page of the protection violation rather than the full address of the exception. Although not ideal, this likely provides sufficient information.

b) For data CPLB miss exceptions, use the DCPLB_FAULT_ADDR register, but be warned that the address reported in this register might be that of a previously canceled speculative exception rather than the true current exception. It might therefore:
i) point to a page which already has a loaded descriptor.
OR
ii) point to an address which will never actually be fetched.

For case i), although a CPLB miss handler might create a redundant CPLB entry (unless further page checking is done), this may be tolerated if a multiple CPLB hit handler exists to remove this rarely generated redundant page descriptor.

For case ii), since the DCPLB_FAULT_ADDR register will never be incorrect immediately after returning from the exception handler, nonsensical addressees can be ignored by the CPLB miss handler without generating an infinite exception handler loop due to repetitively faulty DCPLB_FAULT_ADDR register contents.

c) For data CPLB multiple hit exceptions, have such a handler thanks to the issue described above. Don't count on multiple CPLB exceptions never 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, 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):
0.3

58. 05000262 - Stores To Data Cache May Be Lost:

DESCRIPTION:
A committed pending write into the sub-bank targeted by the first of two consecutive dual-DAG operations will be lost when:

1) Data cache is enabled, AND
2) For the first dual-DAG access, DAG0 is a cache miss, DAG1 is a read, and both accesses alias to the same non-L1 sub-bank, AND
3) The second dual-DAG is the next instruction, and DAG1 is an access (read or write) of L1 SRAM, AND
4) There's an unpredicted change of flow within three clock cycles after the first dual-DAG access. The user has no control over the change of flow.

WORKAROUND:
1) Don't use data cache, OR
2) Avoid consecutive dual-DAG memory accesses where the first dual-DAG access:
a) has both DAGs targeting L2, AND
b) has both DAGs aliasing to the same sub-bank, AND
c) includes a read by DAG1, which is then immediately followed by the second dual-DAG access where DAG1 is an L1 access.

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):
0.3

59. 05000263 - Hardware Loop Corrupted When Taking an ICPLB Exception:

DESCRIPTION:
There is an error in the hardware loop logic which can cause incorrect instructions to get executed when the processor is running loops and instruction ICPLB exceptions occur.

WORKAROUND:
Either:
1) Avoid using hardware loops, OR
2) Make sure hardware loops are located only in L1 memory, OR
3) Make sure ICPLB exceptions do not occur while executing a hardware loop located outside L1 memory.

If a hardware loop is contained within L1 memory, the loop must not generate an ICPLB exception, for example, by crossing a CPLB page boundary into a page with no valid CPLB definitions. In addition, do not allow branching out to non-L1 memory from within the loop when an ICPLB exception might be generated at the target address. Also, if the loop might be interrupted and the interrupt service routines (ISR) reside in non-L1 memory, the ISRs should not generate ICPLB exceptions.

APPLIES TO REVISION(S):
0.3

60. 05000264 - CSYNC/SSYNC/IDLE Causes Infinite Stall in Penultimate Instruction in Hardware Loop:

DESCRIPTION:
If a SSYNC, CSYNC, or IDLE is placed in the second to last instruction of a hardware loop, there is a possibility that the processor will enter an infinite stall when trying to execute the sync.

WORKAROUND:
Do not put a SSYNC, CSYNC, or IDLE instruction in the second to last instruction of a hardware loop.

Because an interrupt or an exception will bring the processor out of the stall, this problem may not be obvious if you're running DMA or interrupts.

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):
0.3

61. 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):

0.3, 0.5

62. 05000266 - IMDMA Destination IRQ Status Must Be Read Prior to Using IMDMA:

DESCRIPTION:
IMDMA transfers do not work properly if there is not a read made from the the IMDMA Destination IRQ status register before the IMDMA control registers are accessed.

WORKAROUND:
Read the IRQ status register for the IMDMA destination channel 0 before writing any IMDMA control registers. This only needs to happen once after reset, not before each write.

APPLIES TO REVISION(S):
0.5

63. 05000267 - IMDMA May Corrupt Data under Certain Conditions:

DESCRIPTION:
When transferring data from L1 memory or on-chip L2 memory TO on-chip L2 memory, data can become corrupted when a "bank conflict" exists in L2 memory or when a core access (instruction or data fetch) is made at the same time as an IMDMA access. A "bank conflict" arises when the IMDMA and a System DMA try to access the same L2 sub-bank in the same cycle. The L2 memory consists of 16 8KB banks.

Transferring data FROM on-chip L2 memory to L1 memory can produce corrupted data, even when bank conflicts are avoided.

This anomaly will not apply to transfers from L1 to L1 memory.

WORKAROUND:
To avoid this errata in transfers from L1 memory or on-chip L2 memory TO on-chip L2 memory, the following conditions must be met:
1) When enabling the IMDMA, first enable the source channel and then issue an SSYNC instruction. Then enable the destination channel and then issue another SSYNC instruction.

2) Ensure that System DMA channels are not accessing the same L2 sub-bank as the IMDMA channels.

3) Avoid any core access (either instruction fetch or data fetch) from occurring to on-chip L2 memory space while an IMDMA is transferring data from L2 memory to L1 memory.

Transfers FROM on-chip L2 memory to L1 memory should be done via System MEMDMA (DMA controllers 1 or 2)

APPLIES TO REVISION(S):
0.3, 0.5

64. 05000269 - High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Increase:

DESCRIPTION:

The internal voltage regulator is susceptible to supply and ground noise transients induced by high I/O activity, particularly in cases of high VDDext. This can result in a higher VDDint than the value that was programmed. In some cases, the value increases to a number outside the upper spec of the range in the data sheet. VDDint returns to the programmed value when the I/O activity diminishes or stops. Devices in BGA packages are more susceptible than devices in LQFP packages.

To date, increased voltages that have exceeded the upper end of the spec value have only been observed while running tests with artificially high I/O activity (e.g., when all bits of the address and data lines toggle every clock). Out-of-spec behavior has not been observed in any customer application running application code.

WORKAROUND:

This problem does not occur if an external voltage regulator is used. To determine if the problem exists in your application, you should monitor the VDDint waveform under the following conditions/setup:

• Apply the maximum VDDext based on the tolerance of VDDext supply.

• Run the application in a steady state (non-startup) condition.

• Connect an oscilloscope with minimum ground and signal loops to VDDint.

• Set the oscilloscope to trigger on a VDDint value that is between a number that is 10% higher than your programmed value and the absolute maximum voltage (see data-sheet for maximum Vddint specification), in order to avoid normal transients.

Not all parts are equally susceptible to this issue. Repeat the above monitoring on a minimum of 10 devices.

If the issue does occur, the value of VDDint will increase during periods of high I/O activity. If the max value of VDDint remains at or below the maximum VDDint, there will be no long term reliability issues, but power consumption will be higher.

Since the problem is a function of VDDext, I/O activity, and the programmed value of VDDint, the following techniques may mitigate/improve this issue:

• The problem is more likely to occur with VDDext values above 3.3V, so it is best to use a 3.3V (or less) regulator with a tolerance of +/- 2% or better.

• Ensure adequate bypassing on VDDext.

• Reduce the I/O activity if possible (for example, operate at a lower SCLK frequency rate).

• Follow the requirements in application note EE-228. In addition, use a PMOS FET with the lowest gate charge ratings consistent with your application's current rating needs.

APPLIES TO REVISION(S):

0.3

65. 05000270 - High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Decrease:

DESCRIPTION:
Heavy I/O activity can cause VDDint to decrease. The reference voltage, which is used to create the set point for the loop, is decreased by the supply noise. The voltage may drop to a level that is lower than the minimum required to meet your application's frequency of operations. The VDDint value returns to the programmed value once high I/O activity is halted.

WORKAROUND:
This issue does not occur when an external regulator is used.   To determine if the problem exists in your application, you should monitor the VDDint waveform under the following conditions/setup:

• Apply the maximum VDDext based on the tolerance of VDDext supply.

• Run the application in a steady state (non-startup) condition.

• Connect an oscilloscope with minimum ground and signal loops to VDDint.

• Set the oscilloscope to trigger on a VDDint value that is 5% lower than the programmed value.

The following items can mitigate this issue:

• Lower the I/O activity by reducing SCLK frequency, if possible.

• Increase the programmed value of the voltage regulator by an amount (in multiples of 50mV) closest to the observed decrease.

• Ensure adequate bypassing on VDDext.

APPLIES TO REVISION(S):
0.3

66. 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):
0.3, 0.5

67. 05000274 - Data Cache Write Back to External Synchronous Memory May Be Lost:

DESCRIPTION:
When the Core Clock is not at least twice as fast as the the System Clock, writes to external synchronous memory may be lost during cache victimization. Note that cache victims are effectively 256 bit wide writes and this issue only manifests itself when performing greater than 32 bit wide write accesses to external synchronous memory.

WORKAROUND:
Either make sure that the Core Clock (CCLK) is at least twice as fast as the System Clock (SCLK) or, if using data cache, use the Write Through policy, as there is no cache victimization in this mode.

APPLIES TO REVISION(S):
0.3, 0.5

68. 05000275 - PPI Timing and Sampling Information Updates:

DESCRIPTION:
The PPI timing information and the corresponding diagrams in the ADSP-BF561: Blackfin® Embedded Symmetric Multi-Processor Data Sheet (Rev. 0, 2/2005) are incorrect.

WORKAROUND:
Refer to the latest data sheet for the correct information.

APPLIES TO REVISION(S):
0.3, 0.5

69. 05000276 - Timing Requirements Change for External Frame Sync PPI Modes with Non-Zero PPI_DELAY:

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

70. 05000277 - Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt:

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

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

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

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

APPLIES TO REVISION(S):
0.3

71. 05000278 - Disabling Peripherals with DMA Running May Cause DMA System Instability:

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

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

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

APPLIES TO REVISION(S):
0.3

72. 05000281 - False Hardware Error when ISR Context Is Not Restored:

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

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

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

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

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

or any similar instruction.

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

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

APPLIES TO REVISION(S):
0.3

73. 05000283 - System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage:

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

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

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

74. 05000287 - Reads Will Receive Incorrect Data under Certain Conditions:

DESCRIPTION:
Under certain specific conditions, a read will receive incorrect data.

This problem can occur when two writes are made through port B surrounding a read that is made through port A. For the problem to occur, the following conditions must be met:

1) Data cache is enabled.
2) A non-cacheable read is made in between the two cacheable writes.

The following piece of code illustrates the problem:

```
p4.h = 0xfeb0; //P4 and P5 point to 2 cacheable locations
p4.l = 0x2000;
p5.h = 0xfeb0;
p5.l = 0x2002;

w[p5] = r0;    // uses byte enable for the upper half of the 32-bit word
w[p4] = r0;    // uses byte enable for the lower half of the 32-bit word
nop;
nop;
i1.l = 0x0102; // i1 points to non-cacheable location
i1.h = 0x2400;
mnop || i0 += 2 || r2.l = w[i1]; // problem occurs HERE
```

In this case, the read will return data from the lower half of the register. Instead, it gets the upper half.

The problem also occurs after a link instruction (which does multiple 32-bit writes) or two 32-bit writes.

Note: The Blackfin processor gives priorities to reads over writes unless an SSYNC or CSYNC is used after the write and before a read.

WORKAROUND:
There are three workarounds for this problem:

1) Disable data cache.
OR
2) Change the DMEM_CONTROL registers so that the port preferences for DAG0 and DAG1 are set to port B.
OR
3) Add a CSYNC after the writes.

APPLIES TO REVISION(S):
0.3

75. 05000288 - SPORTs May Receive Bad Data If FIFOs Fill Up:

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

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

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

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

WORKAROUND:
Avoid the conditions described in the problem description.

Operating so closely to a FIFO overflow should be avoided.

APPLIES TO REVISION(S):
0.3

76. 05000301 - Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space:

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

77. 05000302 - SSYNCs after Writes to DMA MMR Registers May Not Be Handled Correctly:

DESCRIPTION:

When a DMA channel has been granted permission to fetch descriptors from memory, writes to System MMRs associated with the same DMA controller will be held off until the descriptor fetch completes, regardless of the presence of an SSYNC instruction.

One unwanted effect from this behavior would be in the case of DMA interrupts, where the ISR code performs the correct sequence to clear the interrupt request:

```
p0.h = hi(DMA3_IRQ_STATUS);
p0.l = lo(DMA3_IRQ_STATUS);
r0.l = 0x0001;
w[p0] = r0.l;                // Write-1-to-Clear Interrupt Request
ssync;                       // Allow write to complete
rti;
```

If another DMA channel from the same DMA controller is currently fetching descriptors at the time of the write, this write will be delayed and, if the delay exceeds the duration of the subsequent SSYNC instruction, the ISR code will execute the RTI instruction and vector to the ISR again because the DMAx_IRQ_STATUS bit hasn't yet been cleared. This behavior is true for writes to all system MMRs associated with the DMA Controller that is busy doing the descriptor fetch.

WORKAROUND:

If a dummy read from the MMR register is inserted before the SSYNC, this will guarantee that the previous write completes before the read is able to execute. For example, using the above example, read back the IRQ Status register after it is written:

```
p0.h = hi(DMA3_IRQ_STATUS);
p0.l = lo(DMA3_IRQ_STATUS);
r0.l = 0x0001;
w[p0] = r0.l;                // Write-1-to-Clear Interrupt Request
r0.l = w[p0];                // Insert dummy read before ssync
ssync;                       // Allow write to complete
rti;
```

APPLIES TO REVISION(S):

0.3, 0.5

78. 05000305 - SPORT_HYS Bit in PLL_CTL Register Is Not Functional:

DESCRIPTION:

The SPORT Hysteresis bit (SPORT_HYS, bit 15) in the PLL_CTL register is not functional. This bit always reads as 0, and writing a 1 has no effect.

WORKAROUND:

None.

APPLIES TO REVISION(S):

0.3

79. 05000307 - SCKELOW Bit Does Not Maintain State Through Hibernate:

DESCRIPTION:

The SCKELOW bit (bit 15) of VR_CTL does not maintain status through the hibernate reset sequence, therefore it cannot be read during the boot sequence to determine whether the processor is cold-starting or coming from the hibernate state.

WORKAROUND:

If the Hibernate feature is being used, application code can copy VR_CTL to a specific location in external memory before going to Hibernate, which can then be interrogated in an initialization block to determine whether a context save was performed previously or not. For example, create a 32-bit data element in your source code and use the LDF to resolve it to a specific location. In the LDF:

```
RESOLVE(32BitDataLabel, 32BIT_ADDR_IN_EXTERNAL_SDRAM_DATA_SEGMENT);
```

Then, when you prepare to enter Hibernate in your source code, this pseudo-code can be used:

```
#define VR_CTL_HIBERNATE_VALUE   0x000080DC      // Your value for VR_CTL goes here

PTR_TO_32BitDataLabel = VR_CTL_HIBERNATE_VALUE;  // 32-Bit Access
VR_CTL = VR_CTL_HIBERNATE_VALUE;                 // 16-Bit Access

CLI/IDLE PLL Programming Sequence;               // Latch write to VR_CTL
```

Then, in an initialization block, this pseudo-code can be used to check for this exact 32-bit value to determine whether that location was previously written or not:

```
Read PTR_TO_32BitDataLabel;
Compare to VR_CTL_HIBERNATE_VALUE;

if TRUE
   Execute post-hibernate boot sequence;
else
   Perform full boot;
```

APPLIES TO REVISION(S):

0.3

80. 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.

APPLIES TO REVISION(S):

0.3, 0.5

81. 05000312 - Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted:

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

82. 05000313 - PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes:

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

83. 05000315 - Killed System MMR Write Completes Erroneously on Next System MMR Access:

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

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

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

84. 05000320 - PF2 Output Remains Asserted after SPI Master Boot:

DESCRIPTION:
For the Master SPI boot mode, the boot ROM writes the FIO0_DIR register to enable PF2 as an output and uses the FIO0_FLAG_S and FIO0_FLAG_C registers to control the SPI chip-select driven on PF2 to control the SPI slave. When the boot completes, PF2 is enabled as an output and driven high.

WORKAROUND:
When using this boot mode, be sure to reset the FIO0 registers appropriately, if needed. If PF2 is to be utilized for another purpose after booting completes, FIO0_DIR and FIO0_FLAG_D should both be reset to 0x0000.

APPLIES TO REVISION(S):
0.5

85. 05000323 - Erroneous GPIO Flag Pin Operations under Specific Sequences:

DESCRIPTION:
When an access to a Flag I/O System MMR (any register with a FIOx_ prefix) is followed by another System MMR access that is not in the Flag I/O block, the flag's input driver can become active for a moment. As a result, the output values held in the flag latch may clear erroneously, causing unwanted transitions in pin state on the output pins.

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

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

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

```
P5.H = HI(FIO0_FLAG_D);  P5.L = LO(FIO0_FLAG_D);
P4.H = HI(SICA_SYSCR);   P4.L = LO(SICA_SYSCR);

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

/* any FIOx_ sequence */
R6 = W[P5](Z);
R5 = W[P5+FIO0_MASKA_D-FIO0_FLAG_D](Z);  /* FIO0_MASKA_D read */
R6 = R5 & R6;

W[P5+FIO0_FLAG_C-FIO0_FLAG_D] = R6;   /* last FIOx_ access to FIO0_FLAG_C (0xFFC00704) */
R5 = W[P4](Z);                       /* dummy read from SICA_SYSCR (0xFFC00104) */
STI R7;           /* restore interrupts */
```

Note that address bits 4, 5 and 6 of SICA_SYSCR and FIO0_FLAG_C are b#000. Therefore, a SICA_SYSCR read safely resolves the critical situation introduced by the FIO0_FLAG_C access.

The following is a list of safe registers for each FIOx register.

If the last FIOx access was to...	Then a "safe register" is...
FIOx_FLAG_D/C/S/T	SICA_SYSCR
FIOx_MASKA_D/C/S/T	UART_SCR
FIOx_MASKB_D/C/S/T	UART_GCTL
FIOx_DIR/POLAR/EDGE/BOTH	SPORT0_STAT
FIOx_INEN	DMA1_1_CONFIG

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices (VisualDSP++, VDK, 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):
0.3, 0.5

86. 05000326 - SPORT Secondary Receive Channel Not Functional when Word Length >16 Bits:

DESCRIPTION:
When the SPORT is configured in secondary mode with a word length greater than 16 (in the SPORTx_RCR2 register, the RXSE bit is set and the SLEN field is greater than 0x0F), the SPORT receiver gets stuck latching the upper 16 bits of the first word received on the DRxSEC pin into the receive FIFO. The rest of the available space in the FIFO is subsequently filled with this data, rather than with the serial data of the second primary and secondary words, and an overflow error is generated.

WORKAROUND:
Do not use serial word lengths greater than 16 bits when the receive secondary side of the SPORT is enabled.

APPLIES TO REVISION(S):
0.5

87. 05000331 - 24-Bit SPI Boot Mode Is Not Functional:

DESCRIPTION:
The 24-bit SPI boot mode is not functional.

WORKAROUND:
The 16-bit SPI boot mode works properly, so a 16-bit addressable SPI device is an option.

APPLIES TO REVISION(S):
0.3

88. 05000332 - Slave SPI Boot Mode Is Not Functional:

DESCRIPTION:
The Slave SPI Boot Mode is not functional.

WORKAROUND:
Do not use the BMODE[1:0] configuration b#10. Use one of the other boot modes instead.

APPLIES TO REVISION(S):
0.3

89. 05000333 - Flag Data Register Writes One SCLK Cycle after Edge Is Detected May Clear Interrupt Status:

DESCRIPTION:
If a write to any flag data register occurs one system clock cycle after an edge is detected on an edge-triggered interrupt, then the flag bit may be cleared one system clock cycle after it has been set.

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

Writes to the flag data register include Data, Clear, Set and Toggle.

WORKAROUND:
If possible, prevent mixture of different GPIO functions on the same port. The input edge-sensitive pins can be dedicated to one port. A different port can be used to toggle output pins and will prevent against a detected edge stored in the edge-sensitive SIC_ISRx or flag-registers being overwritten/cleared. This can be applied if multiple edge-sensitive input pins need to be assigned to one interrupt.

If only one edge-sensitive source is assigned to one interrupt, it can be assumed to be the source of the interrupt and a read instruction of SIC_ISRx and FIOx_FLAG registers is not required. Note that all interrupts are properly captured, when interrupts are enabled.

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

APPLIES TO REVISION(S):
0.3

90. 05000339 - ALT_TIMING Bit in PLL_CTL Register Is Not Functional:

DESCRIPTION:
The ALT_TIMING bit (bit 4) of the PLL_CTL register, which allows software to configure which edge the PPI data is sampled on with respect to the PPI frame sync, is not functional. The bit always reads as 0, and writing a 1 has no effect.

WORKAROUND:
None.

APPLIES TO REVISION(S):
0.3

91. 05000343 - Memory DMA FIFO Causes Throughput Degradation on Writes to External Memory:

DESCRIPTION:
Due to a design change implemented to address a previous anomaly, Memory DMA write operations to external memory have a 3-deep FIFO rather than a 4-deep FIFO. As a result, Memory DMA throughput can be reduced by up to 25% if the external memory is able to keep up with the data coming from the internal memory buffer. The outbound memory DMA FIFO depth was corrected back to 4 in revision 0.5 silicon.

Note that this is only a problem for memory DMAs and does not apply at all to peripheral DMA channels.

WORKAROUND:
None.

APPLIES TO REVISION(S):
0.3

92. 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):
0.3, 0.5

93. 05000362 - Conflicting Column Address Widths Causes SDRAM Errors:

DESCRIPTION:
The column address width settings for banks 2 and 3 are misconnected in the SDRAM controller. Accesses to bank 2 will result in an error if the Column Address Width for bank 3 (EB3CAW) is not set to be the same as that of bank 2.

WORKAROUND:
If using bank 2, make sure that banks 2 and 3 have the same column address width settings in the EBIU_SDBCTL register. This must be the case regardless of whether or not bank 3 is enabled.

APPLIES TO REVISION(S):
0.3, 0.5

94. 05000363 - UART Break Signal Issues:

DESCRIPTION:

When a Break signal is received, the UART controller should issue a single error interrupt. However, the controller issues a number of error interrupts instead, as it generates an error interrupt for every bit time that the break signal is active. For example, if a break signal holds the line low for ~250ms at a baud rate of 57600. This results in ~1400 error interrupts being generated, independent of the fact that there is no start or stop bits in the stream. Internally, the data is received as 0s in the UART_RBR register, as the data sampled is all 0s during the break signal.

Another problem with the above is the timing of the break signal itself. Depending on when the next valid character is transmitted, the result of the Break signal may split the next valid character into two invalid characters, as the first bits received may be appended to the previous bad data, with the rest of the valid character sampled as the NEXT character. This behavior can propogate through a stream of subsequent characters received over the UART if data is continuously streamed after a break.

WORKAROUND:

For the flood of interrupts generated by the single break signal, there is no workaround other than to have software condense the numerous interrupts into one and then service it. Every error interrupt that is generated must be serviced individually. For example, software could use a flag to control this. If the UART error interrupt handler sets a flag and then skips subsequent interrupt requests until that flag is cleared, the multiple breaks can be treated as one. The same flag would be cleared in the UART RX interrupt to indicate that the break has completed and new valid data has been received over the UART.

For the data being split after the break concludes, that data will be unrecoverable. If the host holds off on sending the next data after the break for one full character's worth of bit times, the Blackfin UART will have recovered from the break signal and will be ready to resume receiving valid data. For example, for 8-bit data with a start bit, a parity bit, and two stop bits, the host should wait at least 12 UART bit times after a break signal before issuing the next valid data.

APPLIES TO REVISION(S):

0.3

95. 05000366 - PPI Underflow Error Goes Undetected in ITU-R 656 Mode:

DESCRIPTION:

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

WORKAROUND:

None.

APPLIES TO REVISION(S):

0.3, 0.5

96. 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):
0.3, 0.5

97. 05000403 - Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall:

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

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

WORKAROUND:
There are two ways to avoid this anomaly:

1) Use edge-sensitivity for the pin(s) being used to generate the wakeup event.

2) Ensure that the edge on the wakeup signal is clean and held at the trigger level for at least 3 system clock (SCLK) cycles.

APPLIES TO REVISION(S):
0.3, 0.5

98. 05000412 - TESTSET Instruction Causes Data Corruption with Writeback Data Cache Enabled:

DESCRIPTION:

If the TESTSET instruction is used to operate on L2 memory and there is data in external memory that is cached using writeback mode, the data in external memory and/or L2 memory can be corrupted. This issue applies equally to both cores.

In order to see this issue, a single core must initiate both the TESTSET and the cache access. The problem is not encountered if one core issues a TESTSET and the other core issues the cache line request. Similarly, the problem does not manifest if the TESTSET and the cache access are to the same memory space (L2/L2 or external/external).

The corrupted data will be seen in L2 and/or external memory when the problem occurs.

WORKAROUND:

Either do not use writeback cache or precede the TESTSET instruction with an SSYNC instruction. If preceding the TESTSET instruction by an SSYNC instruction, do the following:

```
    CLI R0;
    R1 = [P0];  /* If the address that P0 points to is not covered by an installed CPLB,
perform a dummy read to make sure CPLB is installed */
    NOP;
    NOP;
    SSYNC;
    TESTSET (P0);
    STI R0;
```

The problem is also avoided if the TESTSET instruction is used to operate on a non-cacheable location that is in the same space as the writeback data. For example, the TESTSET instruction operates on non-cacheable external memory, and the writeback cacheable location is also in external memory.

APPLIES TO REVISION(S):

0.3, 0.5

99. 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):

0.3, 0.5

100. 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):
0.3, 0.5

101. 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):

0.3, 0.5

102. 05000428 - Lost/Corrupted L2/L3 Memory Write after Speculative L2 Memory Read by Core B:

DESCRIPTION:

When a write to internal L2 memory or external L3 memory follows a speculative read from internal L2 memory, the write may be lost or corrupted. The corrupted/lost write to L2/L3 memory does not have to be a write to the same L2 address as the one speculatively loaded for the problem to occur. For this anomaly to occur, the speculative read must be caused by a read in the shadow of a branch. The accesses do not have to be consecutive accesses, so the problem can occur even if there are multiple instructions between the speculative read and the write. For the "branch predicted not taken" case:

```
BRCC X [predicted not taken]
R0 = [P0];     // If any of these three loads accesses L2 memory from Core
R1 = [P1];     // B, speculative execution in the pipeline causes the
R2 = [P2];     // anomaly trigger condition.
X: ...         // Any number of instructions...
[P3] = R0;     // This write can be corrupted or lost, if write is to L2/L3
```

For the "branch predicted taken" case:

```
BRCC X (bp) [predicted taken]
...            // Predicted Jumped Instructions
X: R0 = [P0];  // This read can cause the anomaly if P0 references L2
...            // Any number of instructions...
[P1] = R0;     // This write can be corrupted or lost, if write is to L2/L3
```

It should be noted that the TESTSET instruction is also impacted by this anomaly, and special instructions such as UNLINK would also have to be protected if the Stack Pointer resides in L2 memory.

The issue does not occur if the speculative read access is caused by an interrupt or exception, and this issue occurs only when the accesses are performed by core B.

WORKAROUND:

Avoid making L2 reads in the shadow of branches. If possible, move the read to before the jump. If this is not possible, the read must be delayed beyond the shadow of the branch.

For the "branch predicted not taken" case, add any 3 instructions between the branch instruction and the read instruction to ensure that the read is not speculatively executed:

```
BRCC X [predicted not taken]
NOP; NOP; NOP; // Pad the read with 3 instructions
R0 = [P0];     // Even if these reads access L2 memory from Core B, they
R1 = [P1];     // will not be speculatively executed, therefore avoiding
R2 = [P2];     // the anomaly trigger condition.
X: ...         // Any number of instructions...
[P3] = R0;     // This L2/L3 write is now safe.
```

For the "branch predicted taken" case, the speculative read cannot be at the target of the branch:

```
BRCC X (bp) [predicted taken]
...            // Predicted Jumped Instructions
X: NOP;        // Added NOP at JUMP target takes the read out of the shadow
R0 = [P0];     // of the branch, thereby avoiding the anomaly trigger condition.
...            // Any number of instructions...
[P1] = R0;     // This L2/L3 write is now safe.
```

APPLIES TO REVISION(S):

0.5

103. 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):
0.3, 0.5

104. 05000458 - SCKELOW Feature Is Not Functional:

DESCRIPTION:
The SCKELOW feature is not functional. Even with the SCKELOW bit set in the VR_CTL register, the SCKE pin will be driven high while $\overline{\text{RESET}}$ is asserted, which will cause any connected SDRAM to exit self-refresh mode.

WORKAROUND:
None.

APPLIES TO REVISION(S):
0.3, 0.5

105. 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):

0.3, 0.5

106. 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):

0.3, 0.5

107. 05000471 - Boot Failure When SDRAM Control Signals Toggle Coming Out Of Reset:

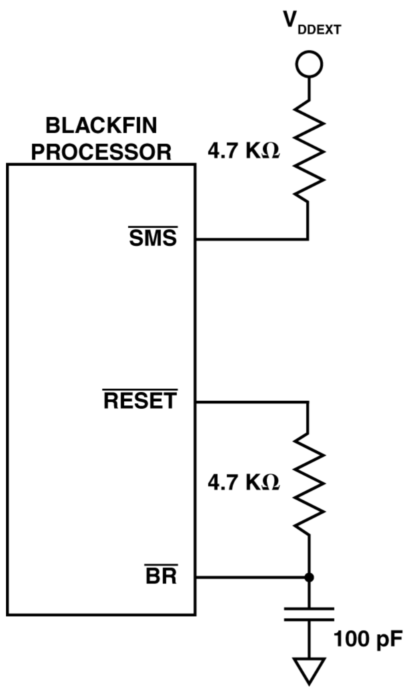
DESCRIPTION:
When $\overline{\text{RESET}}$ is de-asserted at the conclusion of the power-on reset sequence (not applicable to warm resets), there is a very small chance that the SDRAM control signals ($\overline{\text{SRAS}}$, $\overline{\text{SCAS}}$, $\overline{\text{SWE}}$, $\overline{\text{SMS}}$,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

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

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

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

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



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

APPLIES TO REVISION(S):
0.3, 0.5

108. 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):
0.3, 0.5

109. 05000475 - Possible Lockup Condition when Modifying PLL from External Memory:

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

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

APPLIES TO REVISION(S):
0.3, 0.5

110. 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.<BR><BR>The same applies to a semaphore that is shared between multiple cores within the same device.

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

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

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

APPLIES TO REVISION(S):
0.3, 0.5

111. 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):
0.3, 0.5

112. 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):
0.3, 0.5

113. 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):
0.3, 0.5

114. 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):

0.3, 0.5

115. 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):
0.3, 0.5