ANALOG DEVICES

Blackfin®
Embedded Processor

Silicon Anomaly List

ADSP-BF531/BF532/BF533

ABOUT ADSP-BF531/BF532/BF533 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin ADSP-BF531/BF532/BF533 product(s) and the functionality specified in the ADSP-BF531/BF532/BF533 data sheet(s) and the Hardware Reference book(s).

SILICON REVISIONS

A silicon revision number with the form "-x.x" is branded on all parts. The implementation field bits <15:0> of the DSPID core MMR register can be used to differentiate the revisions as shown below.

Silicon REVISION	DSPID<15:0>
0.6	0x0006
0.5	0x0005
0.4	0x0003*
0.3	0x0003

* - See anomaly 05000234

ANOMALY LIST REVISION HISTORY

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

Date	Anomaly List Revision	Data Sheet Revision	Additions and Changes
06/18/2008	D	F	Added Silicon Revision 0.6 Added Anomalies - 05000416
02/08/2008	C	E	Added Anomalies - 05000363, 05000400, 05000402, 05000403
12/10/2007	B	E	Added Anomalies - 05000366, 05000371
09/04/2007	A	E	Initial Consolidated Revision - Replaces anomaly lists for ADSP-BF531 (Rev W), ADSP-BF532 (Rev AB) and ADSP-BF533 (Rev X) Added Anomalies - 05000357 Revised Anomalies - 05000311

NR003532D

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

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF531/BF532/BF533 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.3	0.4	0.5	0.6
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x	x	x
2	05000099	UART Line Status Register (UART_LSR) Bits Are Not Updated at the Same Time	x	x	.	.
3	05000105	Watchpoint Status Register (WPSTAT) Bits Are Set on Every Corresponding Match	x	x	x	x
4	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x	x	x
5	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x	x	x
6	05000158	Instruction DMA Can Cause Data Cache Fills to Fail (Boot Implications)	x	x	.	.
7	05000166	PPI Data Lengths between 8 and 16 Do Not Zero Out Upper Bits	x	x	x	x
8	05000167	Turning SPORTs on while External Frame Sync Is Active May Corrupt Data	x	x	x	x
9	05000179	PPI_COUNT Cannot Be Programmed to 0 in General Purpose TX or RX Modes	x	x	.	.
10	05000180	PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs	x	x	x	x
11	05000183	Timer Pin Limitations for PPI TX Modes with External Frame Syncs	x	.	.	.
12	05000189	False Protection Exceptions when Speculative Fetch Is Cancelled	x	.	.	.
13	05000193	False I/O Pin Interrupts on Edge-Sensitive Inputs When Polarity Setting Is Changed	x	.	.	.
14	05000194	Restarting SPORT in Specific Modes May Cause Data Corruption	x	.	.	.
15	05000198	Failing MMR Accesses when Preceding Memory Read Stalls	x	x	.	.
16	05000199	Current DMA Address Shows Wrong Value During Carry Fix	x	.	.	.
17	05000200	SPORT TFS and DT Are Incorrectly Driven During Inactive Channels in Certain Conditions	x	x	.	.
18	05000201	Receive Frame Sync Not Ignored During Active Frames in SPORT Multi-Channel Mode	x	.	.	.
19	05000202	Possible Infinite Stall with Specific Dual-DAG Situation	x	x	.	.
20	05000203	Specific Sequence That Can Cause DMA Error or DMA Stopping	x	.	.	.
21	05000204	Incorrect Data Read with Writethrough "Allocate Cache Lines on Reads Only" Cache Mode	x	.	.	.
22	05000207	Recovery from "Brown-Out" Condition	x	.	.	.
23	05000208	VSTAT Status Bit in PLL_STAT Register Is Not Functional	x	x	x	x
24	05000209	Speed Path in Computational Unit Affects Certain Instructions	x	.	.	.
25	05000215	UART TX Interrupt Masked Erroneously	x	x	.	.
26	05000219	NMI Event at Boot Time Results in Unpredictable State	x	x	x	x
27	05000225	Incorrect Pulse-Width of UART Start Bit	x	x	.	.
28	05000227	Scratchpad Memory Bank Reads May Return Incorrect Data	x	x	.	.
29	05000229	SPI Slave Boot Mode Modifies Registers from Reset Value	x	x	x	x
30	05000230	UART Receiver is Less Robust Against Baudrate Differences in Certain Conditions	x	x	.	.
31	05000231	UART STB Bit Incorrectly Affects Receiver Setting	x	x	.	.
32	05000233	PPI_FS3 Is Not Driven in 2 or 3 Internal Frame Sync Transmit Modes	x	x	x	.
33	05000234	Incorrect Revision Number in DSPID Register	.	x	.	.
34	05000242	DF Bit in PLL_CTL Register Does Not Respond to Hardware Reset	x	x	.	.
35	05000244	If I-Cache Is On, CSYNC/SSYNC/IDLE Around Change of Control Causes Failures	x	x	.	.
36	05000245	Spurious Hardware Error from an Access in the Shadow of a Conditional Branch	x	x	x	x
37	05000246	Data CPLBs Should Prevent Spurious Hardware Errors	x	x	.	.
38	05000250	Incorrect Bit Shift of Data Word in Multichannel (TDM) Mode in Certain Conditions	.	x	.	.
39	05000253	Maximum External Clock Speed for Timers	x	x	.	.
40	05000254	Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock	.	.	x	x
41	05000255	Entering Hibernate State with RTC Seconds Interrupt Not Functional	x	x	.	.
42	05000257	Interrupt/Exception During Short Hardware Loop May Cause Bad Instruction Fetches	x	x	.	.

No.	ID	Description	0.3	0.4	0.5	0.6
43	05000258	Instruction Cache Is Corrupted When Bits 9 and 12 of the ICPLB Data Registers Differ	x	x	.	.
44	05000260	ICPLB_STATUS MMR Register May Be Corrupted	x	x	.	.
45	05000261	DCPLB_FAULT_ADDR MMR Register May Be Corrupted	x	x	.	.
46	05000262	Stores To Data Cache May Be Lost	x	x	.	.
47	05000263	Hardware Loop Corrupted When Taking an ICPLB Exception	x	x	.	.
48	05000264	CSYNC/SSYNC/IDLE Causes Infinite Stall in Penultimate Instruction in Hardware Loop	x	x	.	.
49	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x	x	x
50	05000269	High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Increase	x	x	.	.
51	05000270	High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Decrease	x	x	.	.
52	05000271	Spontaneous Reset of Internal Voltage Regulator	x	.	.	.
53	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x	x	x
54	05000273	Writes to Synchronous SDRAM Memory May Be Lost	x	x	x	.
55	05000276	Timing Requirements Change for External Frame Sync PPI Modes with Non-Zero PPI_DELAY	x	x	x	x
56	05000277	Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt	x	x	x	.
57	05000278	Disabling Peripherals with DMA Running May Cause DMA System Instability	x	x	x	.
58	05000281	False Hardware Error Exception when ISR Context Is Not Restored	x	x	x	.
59	05000282	Memory DMA Corruption with 32-Bit Data and Traffic Control	x	x	x	.
60	05000283	System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage	x	x	x	.
61	05000288	SPORTs May Receive Bad Data If FIFOs Fill Up	x	x	x	.
62	05000301	Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space	x	x	x	.
63	05000302	SSYNCs after Writes to DMA MMR Registers May Not Be Handled Correctly	x	x	.	.
64	05000305	SPORT_HYS Bit in PLL_CTL Register Is Not Functional	x	x	.	.
65	05000306	ALT_TIMING Bit in PPI_CONTROL Register Is Not Functional	x	x	.	.
66	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x	x	x
67	05000311	Erroneous Flag (GPIO) Pin Operations under Specific Sequences	x	x	x	.
68	05000312	Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted	x	x	x	.
69	05000313	PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes	x	x	x	.
70	05000315	Killed System MMR Write Completes Erroneously on Next System MMR Access	x	x	x	.
71	05000319	Internal Voltage Regulator Values of 1.05V, 1.10V and 1.15V Not Allowed for LQFP Packages	x	x	x	.
72	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	x	x	.
73	05000363	UART Break Signal Issues	x	x	.	.
74	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x	x	x
75	05000371	Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration	x	x	x	.
76	05000400	PPI Does Not Start Properly In Specific Mode	.	.	x	.
77	05000402	SSYNC Stalls Processor when Executed from Non-Cacheable Memory	.	.	x	.
78	05000403	Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall	x	x	x	x
79	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x	x	x

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

DETAILED LIST OF SILICON ANOMALIES

The following list details all known silicon anomalies for the ADSP-BF531/BF532/BF533 including a description, workaround, and identification of applicable silicon revisions.

1. 05000074 - Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported:

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

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

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

Examples of supported instructions:

```
R0 = R0 << 0x1 || [ P0 ] = R1 || NOP;
R0 = R0 << 0x1 || R1 = [ P0 ] || NOP;
R0 = R0 << 0x1 || P3 = [ P0 ] || NOP;
```

WORKAROUND:
In assembly programs, separate the multi-issue instruction into 2 separate instructions. The VisualDSP++ runtime libraries do not use the unsupported instructions. Additionally, the VisualDSP++ Blackfin compiler does not generate the unsupported instructions when targeting the parts and silicon revisions affected by this anomaly

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

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.

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 coding 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, 0.4

3. 05000105 - Watchpoint Status Register (WPSTAT) Bits Are Set on Every Corresponding Match:

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

4. 05000119 - DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops:

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

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

5. 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 (where P0 points to a 16-bit system MMR), this access would fail:

```
w[P0] = R5.H;
```

WORKAROUND:
Use other forms of 16-bit transfers when accessing 16-bit system MMR registers. For example (where p0 points to a 16-bit system MMR):

```
w[p0] = r5.l;
r4.l = w[p0];
r3 = w[p0](z);
w[p0] = r3;
```

The VisualDSP++ Blackfin compiler will not normally emit a problem instruction when generating code. It will insert a pack instruction to swap register halves in the cases where the MMR load occurs with a constant address, e.g. *MMR_Reg = value; It cannot, however, identify pointers unknown at compile time (such as parameters to functions) as pointers to MMRs. The VisualDSP++ runtime libraries also avoid this anomaly.

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

6. 05000158 - Instruction DMA Can Cause Data Cache Fills to Fail (Boot Implications):

DESCRIPTION:
After a DMA or core MMR DTEST register access has occurred to the L1 instruction memory, a data cache fill to the corresponding port may get corrupted data. This situation can only occur if stalls are introduced because the core is accessing the same data memory bank as the data cache fill.

This data cache fill can occur many cycles after the L1 Instruction memory DMA or MMR access. One example of the failure is when a program that enables data cache is booted. The Instruction DMA that occurred during the boot may prime the processor to be susceptible to data corruption in a data cache fill that occurs sometime during the main program's execution.

For Port A, data locations 0xFF80xxxx and instruction locations 0xFFA00000 -0xFFA07FFF (where applicable) are subject to the failure. For Port B, data locations 0xFF90xxxx and instruction locations 0xFFA08000 - 0xFFA0FFFF (where applicable) are subject to the failure.

The problem will also occur if the last DMA transaction was to the L1 instruction memory while a low priority data cache fill is ongoing.

WORKAROUND:
The best workaround for this issue is to set bit 9 of any DCPLB Data register that you use. This bit is shown as a reserved bit in the Hardware Reference Manual, but the documentation is being updated to describe the functionality of this bit.

A second workaround would be to perform a software core reset at the beginning of the program:

```
P0.H = HI(SYSCR);          // Check System Reset Configuration Register
P0.L = LO(SYSCR);
R0.L = W[P0];
CC = BITTST(R0,4);         // Check NO BOOT ON SOFTWARE RESET Bit
IF CC JUMP _No_Boot_Set;
BITSET(R0,4);              // Set NO BOOT ON SOFTWARE RESET Bit
W[P0] = R0;
SSYNC;                     // Ensure write completes before executing RAISE
RAISE 1;

_No_Boot_Set:
BITCLR(R0,4);              // Reset SYSCR to original value
W[P0] = R0;
```

If the program will be performing both data caching and L1 Instruction Memory DMA, perform a Data DMA to the data SRAM which shares a data port with the Instruction SRAM after performing an Instruction DMA. Do not perform any access that would cause data caching or victimization from the time the Instruction DMA begins until after the Data DMA has completed. Also, an MMR DTEST access to the corresponding data bank could also clear out the problem if the original problem was caused by the Instruction Memory DMA.

If the program will access the L1 Instruction Memory through the core MMR DTEST register, follow this access with a core MMR DTEST register access to the corresponding Data Bank.

If the processor is either not DMAing to instruction memory or not using data cache, you will not encounter this problem.

The VisualDSP++ runtime libraries cache support functions contain a workaround for this anomaly where necessary.

APPLIES TO REVISION(S):
0.3, 0.4

7. 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, 0.4, 0.5, 0.6

8. 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, 0.4, 0.5, 0.6

9. 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, 0.4

10. 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, 0.4, 0.5, 0.6

11. 05000183 - Timer Pin Limitations for PPI TX Modes with External Frame Syncs:

DESCRIPTION:
For certain PPI configurations, the general-purpose timers can be utilized as frame sync signals. When the PPI is set up for transmit modes that utilize one or more external frame syncs, the general-purpose timers will have limited functionality.

WORKAROUND:
The anomaly only applies to the "PPI transmit with 2 external frame syncs" mode, and only the TMR2 pin is affected. Timer 2 must be enabled and is not available for general use.

APPLIES TO REVISION(S):
0.3

12. 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 cancelled. 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. Similarly, if a post-incremented indirect data memory access performs a speculative access to a protected or non-existing memory location (for instance R0 = [P0++]; where P0 points to the last word of a page) an incorrect exception would occur.

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

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.

The default VisualDSP++ LDFs include a workaround for this hardware anomaly. The workaround will be automatically enabled for the appropriate silicon revisions, or the workaround can be enabled manually by defining the macro __WORKAROUND_AVOID_LDF_BLOCK_BOUNDARIES when linking.

When enabled, the LDFs will reserve 76 bytes at the boundaries of valid memory blocks.

APPLIES TO REVISION(S):
0.3

13. 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

14. 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

15. 05000198 - Failing MMR Accesses when Preceding Memory Read Stalls:

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

The VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or the workaround can be enabled manually by specifying the compiler flag '-workaround sdram-mmr-read'.

When enabled, the compiler will insert a NOP instruction between a load and an MMR load. It will insert the NOP instruction in cases where the MMR load occurs with a constant address, e.g. *MMR_ADDR = value;. It cannot, however identify pointers unknown at compile time (such as parameters to functions) as pointers to MMRs.

The macro __WORKAROUND_SDRAM_MMR_READ will be defined at compile, assemble and link stages when the workaround is enabled.

APPLIES TO REVISION(S):
0.3, 0.4

16. 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

17. 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, 0.4

18. 05000201 - Receive Frame Sync Not Ignored During Active Frames in SPORT Multi-Channel Mode:

DESCRIPTION:
An external RFS is not ignored during an active frame. This can result in the transfer of data stopping.
The same problem can occur with internal frame syncs if RFSDIV is programmed to a value less than the window size.

WORKAROUND:
Avoid external frame syncs while the current frame is active. Do not program the RFSDIV to a value less than the window size.

APPLIES TO REVISION(S):
0.3

19. 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).

The VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and parts, or the workaround can be enabled manually by specifying the compiler flag '-workaround infinite-stall-202'. When enabled, the compiler will insert a PREFETCH[SP] instruction to avoid the anomaly conditions.

The macro __WORKAROUND_INFINITE_STALL_202 will be defined at compile, assemble, and link stages when the workaround is enabled.

APPLIES TO REVISION(S):
0.3, 0.4

20. 05000203 - 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). 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, corrupting the data.

WORKAROUND:
There are two workarounds for this issue:
1) Locate all DMA data within the same L1 memory bank (for example, bank A or bank B).
2) Ensure that only 1 DMA Channel reads from a given L1 memory bank at any given time.

APPLIES TO REVISION(S):
0.3

21. 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

22. 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

23. 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, 0.4, 0.5, 0.6

24. 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;
```

The VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or the workaround can be enabled manually by specifying the compiler flag '-workaround dreg-comp-latency'. The VisualDSP++ run-time libraries also avoid the anomaly conditions when necessary. When enabled the compiler should insert a NOP instruction between two instructions where the first instruction assigns a value to a DREG, and the second instruction uses the DREG as a parameter to a SIGNBITS, EXTRACT, DEPOSIT or EXPADJ instruction.

The compiler also defines the macros __WORKAROUND_DREG_COMP_LATENCY and __WORKAROUNDS_ENABLED at the source, assembly and link build stages when this workaround is enabled.

APPLIES TO REVISION(S):
0.3

25. 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, 0.4

26. 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, 0.4, 0.5, 0.6

27. 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, 0.4

28. 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 not to map any data to the scratchpad.

Alternatively the VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or the workaround can be enabled manually by specifying the compiler flag -workaround scratchpad-read' With the workaround enabled, when a sequence of three load instructions occurs (or occur as parts of a multi-issue instruction), where at least one of (2) and (3) is a byte load, a nop will be inserted between (1) and (2) or (2) and (3):

A load instruction (1);
A load instruction (2);
A load instruction (3);

The macro __WORKAROUND_SCRATCHPAD_READ will be defined at compile, assemble and link stages when the workaround is enabled.

The VisualDSP++ run-time libraries also avoid this anomaly for appropriate silicon revisions and part numbers.

APPLIES TO REVISION(S):
0.3, 0.4

29. 05000229 - SPI Slave Boot Mode Modifies Registers from Reset Value:

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

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

30. 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, 0.4

31. 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, 0.4

32. 05000233 - PPI_FS3 Is Not Driven in 2 or 3 Internal Frame Sync Transmit Modes:

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

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

33. 05000234 - Incorrect Revision Number in DSPID Register:

DESCRIPTION:
The DSPID register does not contain the correct silicon revision information.

WORKAROUND:
The upper 4 bits of the REVID register (at address 0xFFC0 0014) can be read to obtain silicon revision information. The remaining bits at this location are reserved.

APPLIES TO REVISION(S):
0.4

34. 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, 0.4

35. 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. An example is:

```
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.

If you are programming in assembly, avoid the scenario described above. The Blackfin assembler will warn users who write assembly code that can potentially trigger the anomaly. Warning ea5507 will be issued by the assembler when a CSYNC or SSYNC could possibly be affected by the hardware anomaly.

The VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly for all cases not related to asynchronous events. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or you can enable the workaround manually by specifying the compiler flag -workaround speculative-syncs. With the workaround enabled, the compiler will insert nops to avoid the anomaly condition. The following forms of the anomaly are avoided by the compiler:

```
IF CC JUMP ...;
CSYNC/SSYNC/IDLE
CSYNC/SSYNC/IDLE
CSYNC/SSYNC/IDLE
```

```
IF CC JUMP X (BP);
...
...
X: CSYNC/SSYNC/IDLE
```

```
LSETUP(LT, LB)
   LT: CSYNC/SSYNC/IDLE
       CSYNC/SSYNC/IDLE
       IF CC JUMP X:
   LB: NOP;
X: ...
```

The macro __WORKAROUND_SPECULATIVE_SYNCS will be defined at compile, assemble, and link build phases when the workaround is enabled. The VisualDSP++ run-time libraries also avoid this anomaly for appropriate silicon revisions and part numbers.

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;
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.

APPLIES TO REVISION(S):
0.3, 0.4

36. 05000245 - Spurious Hardware Error from an Access in the Shadow of a Conditional Branch:

DESCRIPTION:
The anomaly is only an issue if there is a load which may access reserved or illegal memory on the opposite control flow of a conditional jump to the taken path. The following sequences demonstrate how this anomaly can appear:

Sequence #1:
For the "predicted not taken" branch, the three instruction slots following the branch should not contain accesses 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 should not 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.

The VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or you can enable the workaround manually by specifying the compiler flag '-workaround speculative-loads'.

With the workaround enabled, the compiler will insert nops to avoid the anomaly condition.

The macro __WORKAROUND_SPECULATIVE_LOADS will be defined at compile, assemble and link build phases when the workaround is enabled.

There are various checks in the compiler which avoid over-applying this workaround. For example, before the workaround is applied, it ensures that the load is:
• not through SP or FP (in which case to stack and not illegal)
• not to a volatile qualified type address (in which case to a known legal address)
• to an address unknown to the compiler
• not duplicated in the branch targets (in which case must be ok to execute speculatively)
• not executed previous to the jump (in which case must be ok to execute speculatively)

The VisualDSP++ run-time libraries also avoid this anomaly for appropriate silicon revisions and part numbers.

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

37. 05000246 - Data CPLBs Should Prevent Spurious Hardware Errors:

DESCRIPTION:
When Data CPLBs are enabled, hardware errors generated as a result of speculative accesses to reserved or undefined memory should not occur, but they do.

WORKAROUND:
None. Avoid accessing reserved memory as part of a speculative fetch.

The VisualDSP++ Blackfin compiler will ignore uses of its '-cplbs' switch when building for silicon revisions that are impacted by this anomaly or when the switch '-workaround no-cplb-spec-protect-246' is used. Uses of -cplbs would otherwise allow the compiler to ignore specualtive fetch issues.

APPLIES TO REVISION(S):
0.3, 0.4

38. 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):
0.4

39. 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, 0.4

40. 05000254 - Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock:

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

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

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

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

APPLIES TO REVISION(S):
0.5, 0.6

41. 05000255 - Entering Hibernate State with RTC Seconds Interrupt Not Functional:

DESCRIPTION:
Entering the low-power Hibernate state is achieved by disabling the internal Voltage regulator (Vddint = 0 Volts). The Real-Time Clock (RTC) is programmed to wake up the voltage regulator at a specific event.

If the RTC wake-up event is the Seconds event (RTC_ICTL = 0x0004), the wake-up signal is erroneously active for the entire second. In this case, the Voltage Regulator cannot be disabled because it will always be woken up immediately.

This applies only to Hibernate state. Deep-Sleep and other low power modes work correctly with the Seconds event.

Note that, for RTC events of greater period, the minimum time before the processor can re-enter Hibernate state after a wake-up event from the RTC is one second. For instance, in the case of the Minute event, the processor cannot re-enter Hibernate mode during the first second.

WORKAROUND:
A possible workaround is to do the following steps:

1) Disable the prescaler (RTC_PREN = 0); thus, the RTC will generate 32768 ticks every second.
2) Use the Stopwatch event instead of the Seconds event. The stopwatch register must be set to 32768 (or, more generally, to a value corresponding to the desired frequency) at every wake-up event.
3) In this case, the wake-up signal will be only approximately 15usec long. This is the minimum time the application has to wait (or do useful things) before re-entering Hibernate mode after a wake-up.

Note that this workaround implies that the RTC is not used for keeping track of the actual time, since the counters are incremented at 32768Hz instead of 1Hz.

APPLIES TO REVISION(S):
0.3, 0.4

42. 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.

The VisualDSP++ Blackfin Compiler includes a workaround for this anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or you can enable the workaround manually by specifying the compiler flag '-workaround short-loop-exceptions-257'. With the workaround enabled, the compiler will include a save and restore of the LC0 and LC1 registers in interrupt and exception handlers when this is not already performed. The compiler would normally only save these registers if they were used within the handler routine.

The macro __WORKAROUND_SHORT_LOOP_EXCEPTIONS will be defined at compile, assemble and link build phases when the workaround is enabled.

APPLIES TO REVISION(S):
0.3, 0.4

43. 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.

The VisualDSP++ Blackfin Runtime Libraries include a workaround for this anomaly.

The _cplb_mgr and _cplb_init routines (which are part of the default cache support in the runtime libraries) set bit 9 (reserved) to the same state as bit 12 (CPLB_L1_CHBL).

APPLIES TO REVISION(S):
0.3, 0.4

44. 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, 0.4

45. 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.

The VisualDSP++ Blackfin Runtime Libraries include a workaround for this anomaly. The workaround ignores DCPLB miss exceptions the first time they are raised from a particular PC. The fault address is guaranteed to be correct the second time.

APPLIES TO REVISION(S):
0.3, 0.4

46. 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.

The VisualDSP++ Blackfin Compiler includes a workaround for this anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or you can enable the workaround manually by specifying the compiler flag '-workaround stores-to-data-cache-262'.

With the workaround enabled the compiler will ensure that dual dag instructions which may trigger the anomaly are not issued. The compiler will attempt to use any bank information available to determine cases where the workaround is not required.

The macro __WORKAROUND_LOST_STORES_TO_DATA_CACHE_262 will be defined at compile, assemble and link build phases when the workaround is enabled.

The VisualDSP++ runtime libraries have been built and modified, where appropriate, to avoid the anomaly.

APPLIES TO REVISION(S):
0.3, 0.4

47. 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, 0.4

48. 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.

The VisualDSP++ Blackfin Compiler includes a workaround for this anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or you can enable the workaround manually by specifying the compiler flag '-workaround pre-loop-end-sync-stall-264'.

With the workaround enabled, the compiler will ensure that the second to last instruction of a hardware loop is not a CSYNC, SSYNC or IDLE instruction, which has the potential to trigger the anomaly.

The macro __WORKAROUND_PRE_LOOP_END_SYNC_STALL_264 will be defined at compile, assemble, and link build phases when the workaround is enabled.

APPLIES TO REVISION(S):
0.3, 0.4

49. 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. Spurious 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), spurious 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).

To improve immunity to noise, newer silicon revisions implement optional hysteresis that can be enabled for input pins by setting bit 15 of the PLL_CTL register, followed by the appropriate PLL programming sequence.

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

50. 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, 0.4

51. 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, 0.4

52. 05000271 - Spontaneous Reset of Internal Voltage Regulator:

DESCRIPTION:
If the internal voltage regulator has been programmed to a voltage level that is different from the default, noise on the voltage regulator circuit can cause it to spontaneously reset to the default value of 1.2V.

Note that the VLEV bit-field in the VR_CTL register is not affected, so this condition cannot be detected by reading the value of the register.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3

53. 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, 0.4, 0.5, 0.6

54. 05000273 - Writes to Synchronous SDRAM Memory May Be Lost:

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

55. 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, 0.4, 0.5, 0.6

56. 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, 0.4, 0.5

57. 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 spurios interrupts.

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

58. 05000281 - False Hardware Error Exception 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:
This is usually not an issue because the context will be restored before returning from an interrupt in most cases.

In cases like the one described, it is sufficient to 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, 0.4, 0.5

59. 05000282 - Memory DMA Corruption with 32-Bit Data and Traffic Control:

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

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

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

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

60. 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 occurs while the System MMR write is stalled (thus killing the write).
3) Interrupt Service Routine performs an "ssync;" instruction.

In order for this anomaly to happen, the interrupt 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 ISR 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 ISR, 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 ISR
```

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 breakpoint after the above code snippet if you need to halt the processor in the ISR.

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

61. 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, 0.4, 0.5

62. 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, 0.4, 0.5

63. 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, 0.4

64. 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, 0.4

65. 05000306 - ALT_TIMING Bit in PPI_CONTROL Register Is Not Functional:

DESCRIPTION:
The ALT_TIMING bit (bit 8) of the PPI_CONTROL 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, 0.4

66. 05000310 - False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory:

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

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

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.

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5, 0.6

67. 05000311 - Erroneous Flag (GPIO) Pin Operations under Specific Sequences:

DESCRIPTION:

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

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

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

WORKAROUND:

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

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

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

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

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

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

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

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

SPECIAL TOOLS IMPACT:

In VisualDSP++ 4.5 (February 2007 Update), the compiler inserted a dummy read of CHIPID. However, since the dummy read address varies (and since interrupts must be disabled), the workaround was replaced in VisualDSP++ 4.5 (June 2007 Update) by header 05000311.h, which defines macros for doing this under user control instead of compiler control. See the header file for more details.

APPLIES TO REVISION(S):

0.3, 0.4, 0.5

68. 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, 0.4, 0.5

69. 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, 0.4, 0.5

70. 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 occurs while the System MMR write is stalled (thus killing the write).
3) Interrupt Service Routine accesses (either read or write) any system MMR.

In order for this anomaly to happen, the interrupt 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 ISR will cause the previously stalled write to complete erroneously.

Similarly, if the System MMR write is killed by an instruction itself, such 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 ISR 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 ISR, 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 ISR
```

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 ISR.

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

71. 05000319 - Internal Voltage Regulator Values of 1.05V, 1.10V and 1.15V Not Allowed for LQFP Packages:

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

This issue only occurs on parts in LQFP packages.

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

APPLIES TO REVISION(S):
0.3, 0.4, 0.5

72. 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, 0.4, 0.5

73. 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, 0.4

74. 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, 0.4, 0.5, 0.6

75. 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.

Beginning with VisualDSP++ 4.5 Update 6 and VisualDSP++ 5.0 Update 2, the tools include workarounds for this anomaly. The C/C++ compiler workaround avoids generating stub function code by inserting NOP instructions or an unconditional JUMP instruction before the RTS. The JUMP workaround variant is used when optimizing for code-size (-Os) when more than two NOPs would otherwise be required. The assembler has been modified to detect and issue a warning (ea5516) for code that could cause the anomaly to occur. The runtime libraries and VDK support libraries have also been modified to avoid the anomaly.

These workarounds are enabled automatically in VisualDSP++ when building for affected processors. The compiler workaround can be enabled manually using the -workaround avoid-quick-rts-371 switch. The assembler warning is controlled using the -anomaly-detect 05000371 switch. When the workarounds are enabled, the macro __WORKAROUND_AVOID_QUICK_RTS_371 is defined at compile, assemble and link stages.

APPLIES TO REVISION(S):

0.3, 0.4, 0.5

76. 05000400 - PPI Does Not Start Properly In Specific Mode:

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

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

APPLIES TO REVISION(S):
0.5

77. 05000402 - SSYNC Stalls Processor when Executed from Non-Cacheable Memory:

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

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

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

1) The SSYNC is in L1 memory or in cacheable L2 memory.

2) The SSYNC is not at a loop bottom where the loop top is located in non-cacheable L2 memory.

3) If the SSYNC is located in a cacheable L2 page, it is at least eight 64-bit words away from the bottom of the page (as specified by a CPLB) if the following (address sequential) page is either L1 or non-cacheable L2 memory.

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

APPLIES TO REVISION(S):
0.5

78. 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, 0.4, 0.5, 0.6

79. 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, 0.4, 0.5, 0.6