ANALOG DEVICES

Blackfin®
Embedded Processor

Silicon Anomaly List

ADSP-BF534/BF536/BF537

ABOUT ADSP-BF534/BF536/BF537 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin ADSP-BF534/BF536/BF537 product(s) and the functionality specified in the ADSP-BF534/BF536/BF537 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>
0.3	0x0003
0.2	0x0002

APPLICABILITY

Each anomaly applies to specific silicon revisions. See Summary or Detailed List for affected revisions. Unless otherwise specified in the Detailed List, anomalies apply to all models.

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
02/08/2008	C	D	Added Anomalies - 05000402, 05000403 Removed Anomaly - 05000359
12/10/2007	B	D	Removed Silicon Revision 0.1 Added Anomalies - 05000350, 05000355, 05000366, 05000371 Removed Anomaly - 05000167
09/04/2007	A	D	Initial Consolidated Revision - Replaces anomaly lists for ADSP-BF534 (Rev M), ADSP-BF536 (Rev L) and ADSP-BF537 (Rev M) Added Anomalies - 05000167, 05000341, 05000357, 05000359

NR003531C

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-BF534/BF536/BF537 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.2	0.3
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x
2	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x
3	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x
4	05000180	PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs	x	x
5	05000244	If I-Cache Is On, CSYNC/SSYNC/IDLE Around Change of Control Causes Failures	x	.
6	05000245	Spurious Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
7	05000250	Incorrect Bit Shift of Data Word in Multichannel (TDM) Mode in Certain Conditions	x	.
8	05000252	EMAC TX DMA Error After an Early Frame Abort	x	.
9	05000253	Maximum External Clock Speed for Timers	x	.
10	05000254	Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock	.	x
11	05000255	Entering Hibernate State with RTC Seconds Interrupt Not Functional	x	.
12	05000256	EMAC MDIO Input Latched on Wrong MDC Edge	x	.
13	05000257	Interrupt/Exception During Short Hardware Loop May Cause Bad Instruction Fetches	x	.
14	05000258	Instruction Cache Is Corrupted When Bits 9 and 12 of the ICPLB Data Registers Differ	x	.
15	05000260	ICPLB_STATUS MMR Register May Be Corrupted	x	.
16	05000261	DCPLB_FAULT_ADDR MMR Register May Be Corrupted	x	.
17	05000262	Stores To Data Cache May Be Lost	x	.
18	05000263	Hardware Loop Corrupted When Taking an ICPLB Exception	x	.
19	05000264	CSYNC/SSYNC/IDLE Causes Infinite Stall in Penultimate Instruction in Hardware Loop	x	.
20	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x
21	05000268	Memory DMA Error when Peripheral DMA Is Running with Non-Zero DEB_TRAFFIC_PERIOD	x	.
22	05000270	High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Decrease	x	.
23	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x
24	05000273	Writes to Synchronous SDRAM Memory May Be Lost	x	.
25	05000277	Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt	x	.
26	05000278	Disabling Peripherals with DMA Running May Cause DMA System Instability	x	.
27	05000280	SPI Master Boot Mode Does Not Work Well with Atmel Data Flash Devices	x	x
28	05000281	False Hardware Error Exception when ISR Context Is Not Restored	x	.
29	05000282	Memory DMA Corruption with 32-Bit Data and Traffic Control	x	.
30	05000283	System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage	x	.
31	05000285	New Feature: EMAC TX DMA Word Alignment (Not Available on Older Silicon)	x	.
32	05000288	SPORTs May Receive Bad Data If FIFOs Fill Up	x	.
33	05000301	Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space	x	x
34	05000304	SSYNCs After Writes To CAN/DMA MMR Registers Are Not Always Handled Correctly	x	.
35	05000305	New Feature: Additional Hysteresis on SPORT Input Pins (Not Available on Older Silicon)	x	.
36	05000307	SCKELOW Bit Does Not Maintain State Through Hibernate	x	.
37	05000309	Writing UART_THR While UART Clock Is Disabled Sends Erroneous Start Bit	x	.
38	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
39	05000312	Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted	x	x
40	05000313	PPI Is Level-Sensitive on First Transfer	x	x
41	05000315	Killed System MMR Write Completes Erroneously on Next System MMR Access	x	.
42	05000316	EMAC RMII Mode: Collisions Occur in Full Duplex Mode	x	.

No.	ID	Description	0.2	0.3
43	05000321	EMAC RMII Mode: TX Frames in Half Duplex Fail with Status "No Carrier"	x	.
44	05000322	EMAC RMII Mode at 10-Base-T Speed: RX Frames Not Received Properly	x	x
45	05000341	Ethernet MAC MDIO Reads Do Not Meet IEEE Specification	.	x
46	05000350	New Feature: UART Remains Enabled after UART Boot (Not Available on Older Silicon)	x	.
47	05000355	Regulator Programming Blocked when Hibernate Wakeup Source Remains Active	x	x
48	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	x
49	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x
50	05000371	Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration	x	x
51	05000402	SSYNC Stalls Processor when Executed from Non-Cacheable Memory	x	.
52	05000403	Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall	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-BF534/BF536/BF537 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):
0.2, 0.3

2. 05000119 - DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops:

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

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

APPLIES TO REVISION(S):
0.2, 0.3

3. 05000122 - Rx.H Cannot Be Used to Access 16-bit System MMR Registers:

DESCRIPTION:
When accessing 16-bit system MMR registers, the high half of the data registers may not be used. If a high half register is used, incorrect data will be written to the system MMR register, but no exception will be generated. For example (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):
0.2, 0.3

4. 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):
0.2, 0.3

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

APPLIES TO REVISION(S):

0.2

6. 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 errata 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):

0.2, 0.3

7. 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):

0.2

8. 05000252 - EMAC TX DMA Error After an Early Frame Abort:

DESCRIPTION:
A DMA error can occur when the EMAC performs a TX early abort operation when the data and descriptors are in different memory spaces (i.e., data in external memory and descriptors in L1).

A TX Early Abort is rare. It occurs when both:
1) Early in a frame, the EMAC TX FIFO is still doing its initial FIFO fill (i.e., it has not yet reached the 90th byte or the end of frame, whichever is less), AND
2) Any of the following four frame abort cases occurs:
a) DMA Underrun
b) Excessive Collisions
c) Excessive Deferral, only if Deferral Check is enabled (OPMODE:DC = 1)
d) Late Collision,only if Late Collision Retry is disabled (OPMODE:LCTRE = 0)

In addition to the four causes of early abort, it may be possible to see this errata later in the frame (after 90 bytes have been delivered by DMA) in the event of either a:
a) DMA Underrun
b) Late Collision, only if Late Collision Retry is disabled (OPMODE:LCTRE = 0)<BR>

WORKAROUND:
The workaround is to configure the memory space like this:

	Descriptor	Data Buffer
Data 1	in A	in B
Data 2	in C	in D
Status	in D	in E

Each letter (A-E) represents a memory space. Different letters may represent the same or different memory space, but each group of items where the same letter appears must be in the same memory space.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):
0.2

9. 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):
0.2

10. 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):
0.3

11. 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):
0.2

12. 05000256 - EMAC MDIO Input Latched on Wrong MDC Edge:

DESCRIPTION:

The MDIO input is latched on the falling edge of MDC plus one SCLK cycle. This may result in communication errors with certain PHY devices when the MDC clock is programmed for very short cycle times. MDC/MDIO are used to communicate with the internal control registers of a PHY device.

Note: the MDC clock is driven by the processor and is derived from SCLK by a programmable clock divider in the EMAC peripheral. If the divisor register MDCDIV is set to N, the period of the MDC will be 2(N+1) times SCLK.

PHYs normally drive MDIO on the rising edge of MDC, while the processor samples MDIO on MDC falling plus one SCLK, as stated above. This means that the total delay of the PHY's clock-to-output delay plus propagation delay (call it tdMDIO) must not exceed one-half of the MDC period plus one SCLK cycle. If the processor's MDC divisor register is set to N, then the propagation delay must not exceed [2(N+1)]/2 + 1 = N+2 SCLK cycles.

WORKAROUND:

If a PHY encounters this issue, then determine tdMDIO for the PHY and set MDCDIV > (tdMDIO/tSCLK) + 2.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):

0.2

13. 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):

0.2

14. 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):
0.2

15. 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):
0.2

16. 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):

0.2

17. 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):
0.2

18. 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):
0.2

19. 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):
0.2

20. 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):

0.2, 0.3

21. 05000268 - Memory DMA Error when Peripheral DMA Is Running with Non-Zero DEB_TRAFFIC_PERIOD:

DESCRIPTION:
When a Memory DMA channel is active at the same time as any peripheral DMA channel and the DEB_TRAFFIC_PERIOD bits in the DMA_TC_PER register are non-zero, data in the Memory DMA transfer can be corrupted.

WORKAROUND:
1. Do not use a Memory DMA channel at the same time as a Peripheral DMA channel in your application
or
2. If you do use Memory DMA at the same time as peripheral DMA, set the DEB_TRAFFIC_PERIOD bits in the DMA_TC_PER register to b#0000.

APPLIES TO REVISION(S):
0.2

22. 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):
0.2

23. 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):
0.2, 0.3

24. 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):
0.2

25. 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):
0.2

26. 05000278 - Disabling Peripherals with DMA Running May Cause DMA System Instability:

DESCRIPTION:
If a peripheral (PPI, SPORT, SPI, etc.) is disabled 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:
Disable the peripheral's associated DMA channel before disabling the peripheral itself.

APPLIES TO REVISION(S):
0.2

27. 05000280 - SPI Master Boot Mode Does Not Work Well with Atmel Data Flash Devices:

DESCRIPTION:
The boot ROM code causes booting from Atmel serial dataflash devices to fail if the boot stream is larger than the SPI device's page size. This only happens if the dataflashes are operating in Standard Addressing Mode.

WORKAROUND:
Use the series D devices (i.e., AT45DB642D) in the Power-Of-2 Addressing Mode.

APPLIES TO REVISION(S):
0.2, 0.3

28. 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):
0.2

29. 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):
0.2

30. 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):
0.2

31. 05000285 - New Feature: EMAC TX DMA Word Alignment (Not Available on Older Silicon):

DESCRIPTION:
Added in revision 0.3, the EMAC TX DMA Word Alignment (TXDWA) mode allows software to program whether TX frame data will begin on an odd- or even-aligned word address. This improves efficiency in some applications by allowing software to avoid a full-frame memory-to-memory copy that was otherwise necessary in order to align the TX data to our current fixed word alignment. Bit 4 (TXDWA) of the EMAC_SYSCTL register is used to enable this mode. Refer to the latest Hardware Reference Manual for a more detailed description of this mode.

On revisions prior to revision 0.3, bit 4 (TXDWA) of the EMAC_SYSCTL register is reserved. Reads of this bit will always return 0, and writes to this bit are ignored.

WORKAROUND:
None.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):
0.2

32. 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):
0.2

33. 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):
0.2, 0.3

34. 05000304 - SSYNCs After Writes To CAN/DMA MMR Registers Are Not Always Handled Correctly:

DESCRIPTION:

The DMA Controller and the CAN peripheral can each hold off Peripheral Access Bus accesses to its MMR space when it is currently accessing the same space itself. This delay may exceed the duration of a subsequent SSYNC instruction in the application code following the write, which could lead to undesired results.

For DMA controllers, when a DMA channel has been granted permission to fetch descriptors from memory, other accesses to System MMRs associated with the same DMA controller will be held off until the descriptor fetch completes, which could take several SCLKs depending on the size of the descriptor being fetched. A side-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 another vector to the ISR will be executed because the DMAx_IRQ_STATUS bit hasn't yet been cleared. This behavior is true for all System MMRs associated with the DMA Controller busy doing the descriptor fetch.

For the CAN controller, accesses to the RAM area could have similar results when the CAN controller is preparing to transmit or is receiving a message. The CAN registers that comprise the RAM area are the Acceptance Mask registers (CAN_AMxxL and CAN_AMxxH) and the Mailbox RAM registers (CAN_MBxx_DATA0, CAN_MBxx_DATA1, CAN_MBxx_DATA2, CAN_MBxx_DATA3, CAN_MBxx_LENGTH, CAN_MBxx_TIMESTAMP, CAN_MBxx_ID0, and CAN_MBxx_ID1).

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 DMA Controller 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):

0.2

35. 05000305 - New Feature: Additional Hysteresis on SPORT Input Pins (Not Available on Older Silicon):

DESCRIPTION:
An enhancement was made that allows an additional 250 mV of hysteresis to be added to the SPORT input pins by setting bit 15 of the PLL_CTL register. This bit is cleared by default and has to be enabled by a write, followed by the appropriate PLL programming sequence.

This additional hysteresis is available to improve immunity to noise on the input pins of the SPORT, as described in anomly 05000265, and the feature is not available on older revisions of silicon.

WORKAROUND:
This feature is not available on older revisions of silicon. If this anomaly applies, this feature is not available for that revision of silicon.

APPLIES TO REVISION(S):
0.2

36. 05000307 - SCKELOW Bit Does Not Maintain State Through Hibernate:

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

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

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

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

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

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

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

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

```
Read PTR_TO_32BitDataLabel;
Compare to VR_CTL_HIBERNATE_VALUE;

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

APPLIES TO REVISION(S):
0.2

37. 05000309 - Writing UART_THR While UART Clock Is Disabled Sends Erroneous Start Bit:

DESCRIPTION:
If the UART Transmit Hold Register (UART_THR) is written while the internal UART clocks are off (UCEN=0 in UART_GCTL), the UART TX pin is driven low until the UART clocks are subsequently enabled (UCEN=1 in UART_GCTL). At that point, the UART logic will then drive the actual UART packet to the TX pin, including start bit, data, parity, and stop bits.

The net effect is that the original low-going edge on the TX pin (at the time the write to UART_THR takes place) will be interpreted as a start bit by a connected receiver. Since the UART logic will properly send the message when the UART clocks do get enabled, this is likely to cause frame errors because the UART will be driving data during the time that the receiver is expecting parity/stop information.

WORKAROUND:
Do not write to the UART_THR register when UCEN=0.

APPLIES TO REVISION(S):
0.2

38. 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):
0.2, 0.3

39. 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++];
```

APPLIES TO REVISION(S):
0.2, 0.3

40. 05000313 - PPI Is Level-Sensitive on First Transfer:

DESCRIPTION:
When the PPI is configured to trigger on an 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 start to transfer if the frame sync is at the active state.

This can cause the PPI to start prematurely.

WORKAROUND:
When using an 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):
0.2, 0.3

41. 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):
0.2

42. 05000316 - EMAC RMII Mode: Collisions Occur in Full Duplex Mode:

DESCRIPTION:
In the full duplex operation of RMII mode, the assertion of both TX_EN and CRS_DV results in a collision detection, which means the outbound data and inbound data are available at the same time. It forces the transmitter to back off for some time and try again. This problem may compromise the transmission bandwidth.

WORKAROUND:
None.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):
0.2

43. 05000321 - EMAC RMII Mode: TX Frames in Half Duplex Fail with Status "No Carrier":

DESCRIPTION:
In the half duplex operation of RMII mode, the No Carrier bit (CSR) of the EMAC_TX_STAT register is always set (no carrier). This prevents the TX frames from being reported with status TX_OK. As a result, the software may not be aware that its TX frames have been sent. This problem comes from the dilemma that the PHY must deassert CRS_DV to avoid collision while the same signal is used for TX_CRS during transmission. This same reason may also lead to a spurious report of loss of carrier in the TX status word (TX_LOSS bit is set).

WORKAROUND:
None.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):
0.2

44. 05000322 - EMAC RMII Mode at 10-Base-T Speed: RX Frames Not Received Properly:

DESCRIPTION:
The Ethernet MAC programmed to operate in RMII mode at 10-Base-T speed is unable to decode RX Frames properly.

WORKAROUND:
Use high speed (100-Base-T) and/or a MII connection for networking.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):
0.2, 0.3

45. 05000341 - Ethernet MAC MDIO Reads Do Not Meet IEEE Specification:

DESCRIPTION:
When reading data from PHY registers via the Ethernet MAC (EMAC) MII interface, the EMAC latches the MDIO signal sourced by the PHY at the rising edge of the System Clock following the falling edge of MDC. The IEEE 802.3 standard stipulates that the PHY drive the MDIO at the rising edge of MDC with a 0 - 300ns output delay. In certain combinations of SCLK/MDC clocks and circuit routing, the EMAC may not be able to latch the correct MDIO data value.

WORKAROUND:
The following condition must be met to ensure the EMAC latches the correct MDIO data:

2 * Tprop + 300ns (or the actual PHY delay) < 0.5 * MDC period + tck where:

• Tprop is the propagation time between the PHY and the Blackfin MII.
• tck is the delay between MDC falling edge and SCLK rising edge (-0.5 * SCLK period + 4ns).

To achieve this requirement, SCLK and MDC can be adjusted by programming the the PLL_DIV and EMAC_SYSCTL registers, respectively.

APPLIES TO MODELS:

ADSP-BF536 and ADSP-BF537 only.

APPLIES TO REVISION(S):
0.3

46. 05000350 - New Feature: UART Remains Enabled after UART Boot (Not Available on Older Silicon):

DESCRIPTION:
On versions of silicon before revision 0.3, the boot kernel disabled the UART port after a successful boot sequence over the UART completed. Consequently, the UART0_DLL and UART0_DLH registers were reset and, therefore, the application did not have access to the bit rate information generated by the autobaud sequence at boot time. This behavior prohibits the processor from sending an acknowledgement to the host that the UART boot has completed successfully, which is common practice.

Starting with silicon revision 0.3, the boot ROM leaves the UART module enabled after the boot sequence completes; therefore, the booted application can continue communication with the host device at the obtained UART bit rate.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.2

47. 05000355 - Regulator Programming Blocked when Hibernate Wakeup Source Remains Active:

DESCRIPTION:
After the processor is placed into the hibernate state, a subsequent peripheral wakeup event can take the part out of hibernate. If that wakeup source remains asserted throughout the initiated reset sequence, writes to the VR_CTL register will not get latched into the regulator when the idle sequence is executed. Latching into the regulator circuit will be blocked until the wakeup source de-asserts.

The write will effectively be lost, however, the written value will go to the VR_CTL register's physical memory-mapped address. If no further writes to VR_CTL are performed, the "lost" write will be latched into the regulator hardware when the next idle instruction is executed.

WORKAROUND:
Ensure that the peripheral hibernate wakeup source de-asserts before attempting to program the regulator via the idle sequence required after the write to VR_CTL.

APPLIES TO REVISION(S):
0.2, 0.3

48. 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):
0.2, 0.3

49. 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):
0.2, 0.3

50. 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):
0.2, 0.3

51. 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):
0.2

52. 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):
0.2, 0.3

ANALOG DEVICES

www.analog.com