



Silicon Anomaly List

ABOUT ADSP-BF538/BF538F SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin ADSP-BF538/BF538F product(s) and the functionality specified in the ADSP-BF538/BF538F 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>
0.5	0x0005
0.4	0x0004
0.3	0x0003

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
09/18/2008	G	B	Added Anomalies: 05000425, 05000426, 05000436 Revised Anomalies: 05000283, 05000315
06/18/2008	F	B	Added Silicon Revision 0.5 Added Anomalies: 05000416
02/08/2008	E	0	Added Anomalies: 05000374, 05000402, 05000403
11/15/2007	D	0	Added Anomalies: 05000355, 05000357, 05000366, 05000371, 05000375
03/13/2007	C	PrD	Added Silicon Revision 0.4 Added Anomalies: 05000315, 05000317
12/07/2006	B	PrD	Added Anomalies: 05000245, 05000310, 05000312, 05000313
09/15/2006	A	PrD	Initial Revision

NR003277G

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF538/BF538F anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.3	0.4	0.5
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x	x
2	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x	x
3	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x	x
4	05000166	PPI Data Lengths between 8 and 16 Do Not Zero Out Upper Bits	x	x	x
5	05000179	PPI_COUNT Cannot Be Programmed to 0 in General Purpose TX or RX Modes	x	x	x
6	05000180	PPI_DELAY Not Functional in PPI Modes with 0 Frame Syncs	x	x	x
7	05000193	False I/O Pin Interrupts on Edge-Sensitive Inputs When Polarity Setting Is Changed	x	x	x
8	05000199	Current DMA Address Shows Wrong Value During Carry Fix	x	.	.
9	05000219	NMI Event at Boot Time Results in Unpredictable State	x	x	x
10	05000229	SPI Slave Boot Mode Modifies Registers from Reset Value	x	x	x
11	05000233	PPI_FS3 Is Not Driven in 2 or 3 Internal Frame Sync Transmit Modes	x	x	x
12	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x	x
13	05000253	Maximum External Clock Speed for Timers	x	x	x
14	05000270	High I/O Activity Causes Output Voltage of Internal Voltage Regulator (Vddint) to Decrease	x	.	.
15	05000272	Certain Data Cache Writethrough Modes Fail for Vddint <= 0.9V	x	x	x
16	05000273	Writes to Synchronous SDRAM Memory May Be Lost	x	.	.
17	05000277	Writes to an I/O Data Register One SCLK Cycle after an Edge Is Detected May Clear Interrupt	x	.	.
18	05000278	Disabling Peripherals with DMA Running May Cause DMA System Instability	x	.	.
19	05000281	False Hardware Error Exception when ISR Context Is Not Restored	x	.	.
20	05000282	Memory DMA Corruption with 32-Bit Data and Traffic Control	x	.	.
21	05000283	System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage	x	.	.
22	05000288	SPORTs May Receive Bad Data If FIFOs Fill Up	x	.	.
23	05000291	Reads from CAN Mailbox and Acceptance Mask Area Can Fail	x	.	.
24	05000293	Hibernate Leakage Current Is Higher Than Specified	x	.	.
25	05000294	Timer Pin Limitations for PPI TX Modes with External Frame Syncs	x	x	x
26	05000301	Memory-To-Memory DMA Source/Destination Descriptors Must Be in Same Memory Space	x	.	.
27	05000304	SSYNCS After Writes To CAN/DMA MMR Registers Are Not Always Handled Correctly	x	.	.
28	05000307	SCKELOW Bit Does Not Maintain State Through Hibernate	x	.	.
29	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x	x
30	05000312	Errors when SSYNC, CSYNC, or Loads to LT, LB and LC Registers Are Interrupted	x	x	.
31	05000313	PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes	x	.	.
32	05000315	Killed System MMR Write Completes Erroneously on Next System MMR Access	x	.	.
33	05000317	PFx Glitch on Write to PORTFIO or PORTFIO_TOGGLE	x	.	.
34	05000355	Regulator Programming Blocked when Hibernate Wakeup Source Remains Active	x	x	.
35	05000357	Serial Port (SPORT) Multichannel Transmit Failure when Channel 0 Is Disabled	x	x	.
36	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x	x
37	05000371	Possible RETS Register Corruption when Subroutine Is under 5 Cycles in Duration	x	x	.
38	05000374	Entering Hibernate State with Peripheral Wakeups Enabled Draws Excess Current	.	x	.
39	05000375	GPIO Pins PC1 and PC4 Can Function as Normal Outputs	x	.	.
40	05000402	SSYNC Stalls Processor when Executed from Non-Cacheable Memory	x	.	.
41	05000403	Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall	x	x	x
42	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x	x

No.	ID	Description	0.3	0.4	0.5
43	05000425	Multichannel SPORT Channel Misalignment Under Specific Configuration	x	x	x
44	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x	x
45	05000436	Specific GPIO Pins May Change State when Entering Hibernate	.	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-BF538/BF538F including a description, workaround, and identification of applicable silicon revisions.

1. 05000074 - Multi-Issue Instruction with dsp32shiftime in slot1 and P-reg Store in slot2 Not Supported:

DESCRIPTION:

A multi-issue instruction with dsp32shiftime 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 dsp32shiftime 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

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):

0.3, 0.4, 0.5

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] = r5.1;  
r4.1 = 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

4. 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

5. 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, 0.5

6. 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

7. 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, 0.4, 0.5

8. 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

9. 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

10. 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

11. 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

12. 05000245 - False Hardware Error from an Access in the Shadow of a Conditional Branch:**DESCRIPTION:**

If a load accesses reserved or illegal memory on the opposite control flow of a conditional jump to the taken path, a false hardware error will occur.

The following sequences demonstrate how this can happen:

Sequence #1:

For the "predicted not taken" branch, the pipeline will load the instructions that sequentially follow the branch instruction that was predicted not taken. By the pipeline design, these instructions can be speculatively executed before they are aborted due to the branch misprediction. The anomaly occurs if any of the three instruction slots following the branch contain loads which might cause a hardware error:

```
BRCC X [predicted not taken]
r0 = [p0];    // If any of these three loads accesses non-existent
r1 = [p1];    // memory, such as external SDRAM when the SDRAM
r2 = [p2];    // controller is off, then a hardware error will result.
```

Sequence #2:

For the "predicted taken" branch, the one instruction slot at the destination of the branch cannot contain an access which might cause a hardware error:

```
BRCC X (bp)
Y: ...
...
X: r0 = [p0]; // If this instruction accesses non-existent memory,
              // such as external SDRAM when the SDRAM controller
              // is off, then a hardware error will result.
```

WORKAROUND:

If you are programming in assembly, it is necessary to avoid the conditions described above.

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 from the stack and not illegal)
- not to a volatile qualified type address (in which case from a known legal address)
- from an address unknown to the compiler
- not duplicated in the branch targets (in which case it must be okay to execute speculatively)
- not executed previous to the jump (in which case it must be okay 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

13. 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, 0.5

14. 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

15. 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

16. 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

17. 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

18. 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 spurious interrupts.

APPLIES TO REVISION(S):

0.3

19. 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

20. 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

21. 05000283 - System MMR Write Is Stalled Indefinitely when Killed in a Particular Stage:

DESCRIPTION:

Consider the following sequence:

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

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

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

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

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

WORKAROUND:

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

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

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

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

APPLIES TO REVISION(S):

0.3

22. 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

23. 05000291 - Reads from CAN Mailbox and Acceptance Mask Area Can Fail:**DESCRIPTION:**

An arbitration failure can occur when the core and the CAN controller attempt to access the CAN mailbox RAM or CAN acceptance mask RAM simultaneously. When this happens, the core read returns data from the CAN mailbox or acceptance mask RAM location that the CAN controller is currently accessing instead of getting the data from the location the core is attempting to read. The CAN controller can access the RAM in three windows in time:

- 1) During the interframe spacing, pending transmit requests are interrogated to determine which enabled TX mailbox has priority. In order to construct the CAN TX message in the internal TX buffer to be placed on the CAN bus, the CAN controller must first read the CAN mailbox RAM (ID, Length, and Data) of the TX mailbox that has priority.
- 2) After the arbitration phase of every message detected on the CAN RX pin, the CAN controller interrogates all message centers to check for a matching message ID by scanning the ID registers and acceptance masks simultaneously. All message centers are checked regardless of whether a match is found or not.
- 3) If a message ID match was detected in 2), the CAN controller will access the CAN mailbox RAM at the conclusion of the message to store the ID, Length, and Data (and Timestamp, if enabled) to the targeted message center.

WORKAROUND:

Whenever a core read from the CAN mailbox RAM or acceptance mask RAM is desired, make three reads of the CAN RAM location of interest, spaced ~80 SCLK cycles apart, and take the majority decision to determine the good read. The 80 SCLK delay ensures that no two reads come from the same bad window where the failure can happen. Due to the specific timing required, this workaround must include disabling interrupts in order to prevent against the reads spanning multiple windows where the arbitration logic can fail. The following pseudo-code is an example:

```

r0 = CAN_MB00_DATA0;           // Base Address of CAN Mailbox RAM
r1 = MAILBOX_NUMBER << 5;     // << 5 Provides Offset into CAN RAM
p0 = r0 + r1;                 // Add offset to base and set p0 pointer
p1 = TEMP_AREA;              // 3 local copies of mailbox (DATA3/2/1/0, LENGTH, ID0/1)
p2 = CAN_STATUS;            // System MMR reads insert needed SCLK domain delays
cli r1;                       // Disable interrupts (begin critical region)
p3=3; lsetup(l10b,l10e) lc0=p3; // Read entire mailbox 3 times
l10b: r0 = w[p0+CAN_MB00_DATA3-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store DATA3
      r0 = w[p0+CAN_MB00_DATA2-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store DATA2
      r0 = w[p0+CAN_MB00_DATA1-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store DATA1
      r0 = w[p0+CAN_MB00_DATA0-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store DATA0
      r0 = w[p0+CAN_MB00_LENGTH-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store LENGTH
      r0 = w[p0+CAN_MB00_ID0-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store ID0
      r0 = w[p0+CAN_MB00_ID1-CAN_MB00_DATA0](z); w[p1++] = r0; // Read/Store ID1
p3=16; lsetup(l20b,l20e) lc1=p3;
l20b:
l20e: r0 = w[p2](z);           // 16 reads provides required delay in SCLK domain
l10e: nop;
sti r1;                       // Restore interrupts (end critical region)
p1 = TEMP_AREA;              // Restore P1 for comparison
p2 = GOOD_READ_DATA;        // Local copy of mailbox area to write good data to
p3=7; lsetup(l30b,l30e) lc0=p3; // Compare loop for all 7 registers in mailbox RAM
l30b: r0 = w[p1](z); r1 = w[p1+7*2](z); r2 = w[p1+7*4](z); // Get the values read
      CC = r1 == r2;          // Compare 2nd read to 3rd read
      if CC r0 = r2;         // If equal, this is the good value. Save to R0.
      p1+= 2;                // Check next register in mailbox RAM
l30e: w[p2++] = r0;          // If 2nd read != 3rd read, 1st read was good

```

Translating the above code to C/C++ is not recommended, as the compiler's generated code may not result in the timing required. If C/C++ code is used, this ASM code should be in-lined and, if desired, a macro can be created to utilize C/C++ data types. Please see the VisualDSP++ C/C++ Compiler and Libraries Manual for Blackfin® Processors for guidance.

APPLIES TO REVISION(S):

0.3

24. 05000293 - Hibernate Leakage Current Is Higher Than Specified:

DESCRIPTION:

Hibernate leakage current is specified to be 50uA in the processor data-sheet. Due to this anomaly, the actual measurement is ~200uA.

WORKAROUND:

None.

APPLIES TO REVISION(S):

0.3

25. 05000294 - 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:

In all transmit modes with external frame sync(s), the timer pin(s) associated with the frame sync(s) (TMR1 and/or TMR2) must be configured as inputs. The Timer(s) must be enabled in EXT_CLK mode (TMODE = 11) with the output pad disable feature activated (OUT_DIS = 1).

For PPI_FS1:

```
*pTIMER1_CONFIG = OUT_DIS | EXT_CLK;  
*pTIMER_ENABLE |= TIMEN1;
```

For PPI_FS2:

```
*pTIMER2_CONFIG = OUT_DIS | EXT_CLK;  
*pTIMER_ENABLE |= TIMEN2;
```

Note: If used as an external frame sync, the timer is not available for general-purpose use.

APPLIES TO REVISION(S):

0.3, 0.4, 0.5

26. 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

27. 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):

0.3

28. 05000307 - SCKELOW Bit Does Not Maintain State Through Hibernate:**DESCRIPTION:**

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

WORKAROUND:

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

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

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

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

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

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

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

```
Read PTR_TO_32BitDataLabel;
Compare to VR_CTL_HIBERNATE_VALUE;

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

APPLIES TO REVISION(S):

0.3

29. 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 occurring.

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

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

30. 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

31. 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

32. 05000315 - Killed System MMR Write Completes Erroneously on Next System MMR Access:**DESCRIPTION:**

Consider the following sequence:

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

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

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

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

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

WORKAROUND:

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

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

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

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

APPLIES TO REVISION(S):

0.3

33. 05000317 - PFX Glitch on Write to PORTFIO or PORTFIO_TOGGLE:

DESCRIPTION:

When PORTFIO or PORTFIO_TOGGLE is written, a glitch can occur on the PFX pins.

WORKAROUND:

Read the register to be written, then write the register.

APPLIES TO REVISION(S):

0.3

34. 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):

0.3, 0.4

35. 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

36. 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

37. 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

38. 05000374 - Entering Hibernate State with Peripheral Wakeups Enabled Draws Excess Current:

DESCRIPTION:

If the VR_CTL register is configured with any of the peripheral wakeup bits set to 1 prior to putting the processor into the Hibernate State, the processor will consume ~10 mA instead of the documented 50 uA.

This excessive current consumption is not present when none of the peripheral wakeups are enabled (i.e., only assertion of the /RESET pin can take the processor out of Hibernate).

WORKAROUND:

None.

APPLIES TO REVISION(S):

0.4

39. 05000375 - GPIO Pins PC1 and PC4 Can Function as Normal Outputs:

DESCRIPTION:

The PC1 and PC4 GPIO pins of the processor, though documented as 5V-tolerant input/open-drain output pins, can be configured as GPIO outputs and provide the ability to drive the output pin both high and low.

WORKAROUND:

For designs moving to silicon with this anomaly fixed, if PC1 and/or PC4 are intended to be used as full-featured GPIO outputs, an external pull-up resistor is required to "drive" logic high on the output pin. With this external pull-up in place, no changes would be required for software written for silicon with this anomaly.

APPLIES TO REVISION(S):

0.3

40. 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):

0.3

41. 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

42. 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

43. 05000425 - Multichannel SPORT Channel Misalignment Under Specific Configuration:

DESCRIPTION:

When using the Serial Port in Multi-Channel Mode, the transmit and receive channels can get misaligned if a very specific configuration for the SPORT is met, as follows:

- 1) Window Offset (WOFF) = 0.
- 2) Frame Delay (MFD) > 0.
- 3) Window Size is an odd multiple of 8 (i.e., WSIZE is an even number).
- 4) The time between RFS pulses is exactly equal to the window duration.

When this exact configuration is used, the multi-channel mode channel enable registers are mismatched after the first window concludes, which results in the TDV signal being driven according to incorrect channel assignments and receive data being sampled on the wrong channels. So, the first window will send and receive properly, but all windows after the first will be misaligned and data sent and received will be corrupted.

This error occurs for external and internal clocks and RFS.

WORKAROUND:

There are several workarounds possible:

- 1) Use a window offset other than 0.
- 2) Use a frame delay of 0.
- 3) Use a window size that is an even multiple of 8.
- 4) For internal RFS, make sure that SPORTx_RFSDIV is at least equal to the window size (# of enabled channels * SLEN).

APPLIES TO REVISION(S):

0.3, 0.4, 0.5

44. 05000426 - Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors:**DESCRIPTION:**

A false hardware error is generated if there is an indirect jump or call through a pointer which may point to reserved or illegal memory on the opposite control flow of a conditional jump to the taken path. This commonly occurs when using function pointers, which can be invalid (e.g., set to -1). For example:

```
CC = P2 == -0x1;
IF CC JUMP skip;
CALL (P2);
skip:
RETS;
```

Before the IF CC JUMP instruction can be committed, the pipeline speculatively issues the instruction fetch for the address at -1 (0xffffffff) and causes the false hardware error. It is a false hardware error because the offending instruction is never actually executed. This can occur if the pointer use occurs within two instructions of the conditional branch (predicted not taken), as follows:

```
BRCC X [predicted not taken]
Y: JUMP (P-reg); // If either of these two p-regs describe non-existent
  CALL (P-reg); // memory, such as external SDRAM when the SDRAM
X: RETS;        // controller is off, then a hardware error will result.
```

WORKAROUND:

If instruction cache is on or the ICPLBs are enabled, this anomaly does not apply.

If instruction cache is off and ICPLBs are disabled, the indirect pointer instructions must be 2 instructions away from the branch instruction, which can be implemented using NOPs:

```
BRCC X [predicted not taken]
Y: NOP;          // These two NOPs will properly pad the indirect pointer
  NOP;          // used in the next line.
  JUMP (P-reg);
  CALL (P-reg);
X: RETS;
```

APPLIES TO REVISION(S):

0.3, 0.4, 0.5

45. 05000436 - Specific GPIO Pins May Change State when Entering Hibernate:**DESCRIPTION:**

When reset, the GPIO ports C, D, and E pins select their peripheral function, and the PORTxIO_FER registers must be programmed by software in order to use them as GPIO pins. When hibernate state is entered, the GPIO ports are reset, thus configuring all Port C, D, and E pins to their peripheral function. Subsequently, the processor will drive any peripheral output pins to the peripheral pin's inactive state immediately before the processor enters hibernate, at which point the pins are tri-stated, as documented in the processor datasheet. For example, the CANTX pin (PC0) will be driven inactive HIGH just before it tri-states, thus resulting in potentially unexpected signaling if the pin is used for GPIO purposes during normal operation. If the pin was already driving high, there is no issue. However, if it was driving low, the unexpected high pulse may have negative system implications. The table below depicts the affected GPIO pins:

PIN NAME	CURRENT STATE: LOW	CURRENT STATE: HIGH	CURRENT STATE: HI-Z
CANTX/PC0	Low --> High --> Hi-Z	High --> Hi-Z	Hi-Z --> High --> Hi-Z
TX1/PD11	Low --> High --> Hi-Z	High --> Hi-Z	Hi-Z --> High --> Hi-Z
TX2/PD13	Low --> High --> Hi-Z	High --> Hi-Z	Hi-Z --> High --> Hi-Z

The proper way to read this table is "For PIN NAME, if the state of the pin at the time of executing the hibernate sequence is CURRENT STATE: X, then the pin will be driven to the states defined in the CURRENT STATE: X column". In the table, the normal behavior is to go from the current state directly to Hi-Z. The anomaly is the intermittent state shown between the current state and Hi-Z.

WORKAROUND:

The best workaround is to not use the pins in the table above if other devices in the system cannot tolerate the indicated pin transition before the pin goes to Hi-Z.

APPLIES TO REVISION(S):

0.4, 0.5