ANALOG DEVICES

Blackfin Embedded Processor

Silicon Anomaly List ADSP-BF700/701/702/703/704/705/706/707

ABOUT ADSP-BF700/701/702/703/704/705/706/707 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF700/701/702/703/704/705/706/707 product(s) and the functionality specified in the ADSP-BF700/701/702/703/704/705/706/707 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 REVID bits <31:28> of the TAPC0_IDCODE register can be used to differentiate the revisions as shown below.

Silicon REVISION	TAPC0_IDCODE.REVID
1.1	0x2
1.0	0x1

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
10/06/2016	G	A	Added Anomaly: 19000058 Removed Anomaly: 19000053
06/16/2016	F	A	Added Silicon Revision 1.1 Removed Silicon Revision 0.0 Added Anomalies: 19000059
02/29/2016	E	A	Added Anomalies: 19000055, 19000057
05/17/2015	D	PrD	Added Silicon Revision 1.0 Added Anomalies: 19000051, 19000053, 19000054 Revised Anomalies: 19000047
11/20/2014	C	PrD	Added Anomalies: 19000040, 19000043, 19000045, 19000047
07/03/2014	B	PrC	Added Anomalies: 19000026, 19000032, 19000034, 19000038
04/03/2014	A	PrB	Initial Version

Blackfin and the Blackfin logo are registered trademarks of Analog Devices, Inc.

NR004345G

Document Feedback

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.

One Technology Way, P.O.Box 9106, Norwood, MA 02062-9106 U.S.A.
Tel: 781.329.4700
©2016 Analog Devices, Inc. All rights reserved.
Technical Support **www.analog.com**

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF700/701/702/703/704/705/706/707 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	Rev 1.0	Rev 1.1
1	19000002	Disabling Posted MMR Writes While Writes Are Outstanding May Cause Unpredictable Results	x	x
2	19000003	Invalid Opcode Does Not Cause Exception	x	x
3	19000004	SEQSTAT Parity Bits Cannot Be Cleared	x	x
4	19000005	Parity Error Status Registers May Capture Incorrect Error Address	x	x
5	19000007	Reads of SPU_SECCHK by Non-Secure Masters Result in an Erroneous Violation Interrupt	x	x
6	19000010	STI Directly Before CLI Does Not Enable Interrupts	x	x
7	19000017	Reading Certain PKTE Registers May Return Incorrect Data During Packet Processing	x	x
8	19000026	Secure SPI Master Boot Only Supported From Memory-Mapped SPI Devices on SPI2	x	x
9	19000032	Transactions to Certain SPU and SMPU MMR Regions Cause Erroneous Errors	x	x
10	19000034	BP_CFG.JUMPCCEN Bit Is Not Enabled by Default	x	x
11	19000038	Writes to the SPI_SLVSEL Register Do Not Take Effect	x	x
12	19000040	Deep Power Down Mode Exit for LPDDR Does Not Work as Expected	x	x
13	19000043	GP Timer Generates First Interrupt/Trigger One Edge Late in EXTCLK Mode	x	x
14	19000045	TESTSET Instruction May Cause Core Hang	x	x
15	19000047	Dynamic Branch Prediction for Self-Modifying Code is Not Functional	x	x
16	19000051	Internal DMC PHY DLL May Not Lock to the New DCLK Frequency	x	x
17	19000054	Dynamic Branch Prediction During Self-Nested ISRs Causes Unpredictable Results	x	.
18	19000055	Core Writes to CAN MMRs May Not Occur	x	.
19	19000057	Emulator Cannot Unlock Locked Processors if Other Devices are Present in JTAG Scan Chain	x	.
20	19000058	System MMR Reads Can Cause Core Hang	x	x
21	19000059	SMC Byte Enable Signals Tri-State During Read Operations	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-BF700/701/702/703/704/705/706/707 including a description, workaround, and identification of applicable silicon revisions.

1. 19000002 - Disabling Posted MMR Writes While Writes Are Outstanding May Cause Unpredictable Results:

DESCRIPTION:
Disabling posted MMR writes using the SYSCFG.MPWEN bit while MMR writes are outstanding may cause unpredictable results.

WORKAROUND:
Always place an SSYNC instruction after clearing the SYSCFG.MPWEN bit.

APPLIES TO REVISION(S):
1.0, 1.1

2. 19000003 - Invalid Opcode Does Not Cause Exception:

DESCRIPTION:
The opcode 0xEC81D373 is invalid and should cause an exception if executed, but it does not.

WORKAROUND:
None

APPLIES TO REVISION(S):
1.0, 1.1

3. 19000004 - SEQSTAT Parity Bits Cannot Be Cleared:

DESCRIPTION:
The parity bits in the Sequencer Status (SEQSTAT) register cannot be cleared.

WORKAROUND:
Use the L1IM_IPERR_STAT.BYTELOC and L1DM_DPERR_STAT.BYTELOC bits to clear parity errors. The SEQSTAT parity bits will not update to show that the parity error(s) have been cleared unless a core or system reset is applied.

APPLIES TO REVISION(S):
1.0, 1.1

4. 19000005 - Parity Error Status Registers May Capture Incorrect Error Address:

DESCRIPTION:
In the rare case of two consecutive parity errors, the L1IM_IPERR_STAT and L1DM_DPERR_STAT registers may capture the second error address location instead of the first. For this condition to occur, both of the following events must occur:

1. Two consecutive DAG0 reads have parity errors
2. The return of the first read data is delayed by the core

WORKAROUND:
None

APPLIES TO REVISION(S):
1.0, 1.1

5. 19000007 - Reads of SPU_SECCHK by Non-Secure Masters Result in an Erroneous Violation Interrupt:

DESCRIPTION:
Reads of SPU_SECCHK by non-secure masters result in an erroneous SPU violation interrupt. The erroneous interrupt will only be observed if the SPU violation interrupt is enabled.

WORKAROUND:
There are two possible workarounds:

1. Do not use the SPU_SECCHK register.
2. Ignore the erroneous SPU interrupt after SPU_SECCHK is read.

APPLIES TO REVISION(S):
1.0, 1.1

6. 19000010 - STI Directly Before CLI Does Not Enable Interrupts:

DESCRIPTION:
If a CLI instruction immediately follows an STI instruction (as in the following example), interrupts will not be enabled:

```
STI;
CLI R0;
```

When the above sequence is executed, even pending interrupts will not be accepted by the core.

WORKAROUND:
Make sure there are two or more instructions between the STI and CLI instructions. NOP instructions can be used.

APPLIES TO REVISION(S):
1.0, 1.1

7. 19000017 - Reading Certain PKTE Registers May Return Incorrect Data During Packet Processing:

DESCRIPTION:
Reading out the PKTE_BUF_THRESH, PKTE_INBUF_CNT, or PKTE_OUTBUF_CNT registers within one SCLK1 cycle of the Packet Engine starting to process a packet results in an incorrect value being read. This situation can occur when working in Direct Host Mode and starting to poll for the amount of data that can be transferred (input or output) shortly after starting packet processing by writing to the PKTE_SA_RDY register.

For Autonomous Ring Mode, there is no need to read the affected registers because the data will automatically be transferred out to the specified host memory buffers.

WORKAROUND:
In all modes, the anomaly can by avoided by not reading any of the affected registers after starting packet processing using the PKTE_SA_RDY register.

Additionally, since Direct Host Mode is a manual sequential operation, the Data Output Buffer can be emptied before configuring and starting a new job to process another packet. It can then be assumed that the Data Input Buffer is empty when starting with a new packet. By skipping the first polling operation, this anomaly can be avoided. Also, the maximum amount of data to transfer can be assumed to be equal to the Input Data Buffer size of 256 bytes, so there is no need to check the threshold register to gauge this. For the output side, it suffices not to start polling for the availability of data before input data has been transferred.

APPLIES TO REVISION(S):
1.0, 1.1

8. 19000026 - Secure SPI Master Boot Only Supported From Memory-Mapped SPI Devices on SPI2:

DESCRIPTION:
Secure SPI master boot can only be done in memory-mapped mode from SPI2. SPI0 and SPI1 do not support memory-mapped mode; therefore, they cannot support secure SPI master boot. The same restriction applies when calling the ROM API to boot.

WORKAROUND:
If secure SPI boot is needed, always configure dBootCommand to use SPI2 in XIP mode. When calling the ROM API, ensure that the lowest nibble (boot source device) of the boot command parameter is 0x7. The memory-mapped address where the boot needs to be started also needs to be passed as a start address parameter. For example, the below ROM API call boots from SPI flash mapped to 0x40000000 in XIP mode using SPI2:

```
adi_rom_Boot(0x40000000,0,0,0,0x207);
```

APPLIES TO REVISION(S):
1.0, 1.1

9. 19000032 - Transactions to Certain SPU and SMPU MMR Regions Cause Erroneous Errors:

DESCRIPTION:
Non-secure reads or writes to the upper half of each SPU instance's MMR space will be erroneously blocked and cause a bus error when the SPU is configured as a non-secure slave. The same is true for each SMPU instance. The affected MMR address range can be calculated for each instance of the SPU and SMPU, as follows:

Lower bound = Instance Address Offset + 0x800
Upper bound = Instance Address Offset + 0xFFF

WORKAROUND:
None

APPLIES TO REVISION(S):
1.0, 1.1

10. 19000034 - BP_CFG.JUMPCCEN Bit Is Not Enabled by Default:

DESCRIPTION:
The branch predictor does not come out of reset with optimal settings for most general-purpose programs. The BP_CFG.JUMPCCEN bit should be set.

WORKAROUND:
Set the BP_CFG.JUMPCCEN bit as early as possible in the application program.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
1.0, 1.1

11. 19000038 - Writes to the SPI_SLVSEL Register Do Not Take Effect:

DESCRIPTION:
A single write to the SPI_SLVSEL register should change the state of the register and cause the modified software-controlled SPI slave selects to assert or de-assert. Instead, a single write to SPI_SLVSEL has no effect.

WORKAROUND:
Any write to SPI_SLVSEL should be done twice (back-to-back) with the same value in order for the change to take effect.

APPLIES TO REVISION(S):
1.0, 1.1

12. 19000040 - Deep Power Down Mode Exit for LPDDR Does Not Work as Expected:

DESCRIPTION:
The following steps show the intended DMC programming model for LPDDR deep power down mode exit:

1. Clear the DMC0_CTL.DPDREQ bit to exit from deep power down mode.
2. Wait for DMC0_STAT.DPDACK to be cleared.
3. Wait for DMC0_STAT.INITDONE to be set to make sure that the automatic initialization sequence is completed.

The automatic initialization sequence in Step 3 may fail to correctly complete, though the DMC0_STAT.INITDONE bit will be set as if the initialization had completed correctly. When the automatic initialization sequence does not complete as expected, unpredictable results (e.g., data corruption) can occur during subsequent accesses to LPDDR memory.

WORKAROUND:
As shown below, a fresh memory initialization sequence must be explicitly initiated by user code after exiting deep power down mode:

1. Clear the DMC0_CTL.DPDREQ bit to exit deep power down mode.
2. Wait for DMC0_STAT.DPDACK to be cleared.
3. Wait for DMC0_STAT.INITDONE bit to be set (this does not guarantee that the automatic initialization sequence was successful, so step 4 must be performed as well).
4. Set the DMC0_CTL.INIT bit and wait again for the DMC0_STAT.INITDONE to be set.

The C source code to implement the recommended sequence above is:

```
#include <cdefBF707.h>

*pREG_DMC0_CTL&=~BITM_DMC_CTL_DPDREQ;                 // Step 1

while((*pREG_DMC0_STAT&BITM_DMC_STAT_DPDACK)==1);    // Step 2

while((*pREG_DMC0_STAT&BITM_DMC_STAT_INITDONE)==0); // Step 3

*pREG_DMC0_CTL|=BITM_DMC_CTL_INIT;                  // Step 4
while((*pREG_DMC0_STAT&BITM_DMC_STAT_INITDONE)==0);
```

APPLIES TO REVISION(S):
1.0, 1.1

13. 19000043 - GP Timer Generates First Interrupt/Trigger One Edge Late in EXTCLK Mode:

DESCRIPTION:
When any GP Timer is configured in external clock mode, the first interrupt/trigger should occur, along with the setting of the corresponding `TIMER_DATA_ILAT` bit, after the `TIMER_TMRn_CNT` register reaches the value programmed in the `TIMER_TMRn_PER` register. Instead, the interrupt/trigger and the setting of the `TIMER_DATA_ILAT` bit occur one signal edge later. At this point, the `TIMER_TMRn_CNT` register will have rolled over to 1. Subsequent interrupts/triggers occur after the correct number of edges.

For example, the `TIMER_TMRn_PER` register is configured to a value of 7. After the timer has started, the first interrupt/trigger will occur after the timer receives 8 external signal edges. Subsequent interrupts/triggers will correctly occur every 7 signal edges.

WORKAROUND:
For interrupts/triggers to occur every **n** edges of the external clock, the `TIMER_TMRn_PER` register should be configured to **n-1** for the initial event and to **n** for subsequent events, as shown in the following pseudocode:

```
TIMER_TMRn_PER = n-1;   // Configure PERIOD register with n-1
TIMER_RUN_SET = 1;      // Enable the timer
TIMER_TMRn_PER = n;     // Configure PERIOD register with n
```

The update which sets TIMER_TMRn_PER = n will not take effect until the 2nd period.

APPLIES TO REVISION(S):
1.0, 1.1

14. 19000045 - TESTSET Instruction May Cause Core Hang:

DESCRIPTION:
The TESTSET instruction can cause the core to hang.

WORKAROUND:
Do not use TESTSET.

APPLIES TO REVISION(S):
1.0, 1.1

15. 19000047 - Dynamic Branch Prediction for Self-Modifying Code is Not Functional:

DESCRIPTION:
Dynamic branch prediction for self-modifying code is not functional and may cause unpredictable results.

WORKAROUND:
If self-modifying code is used, reset the branch predictor before each code modification, as described in the following pseudo-code:

1. CSYNC instruction // Allows pending transactions to complete
2. Set BP_CFG.CLRBP // Begin clearing the prediction table
3. Clear SYSCFG.BPEN // Disable the branch predictor
4. Wait at least 150 CCLK cycles // Allows prediction table to clear
5. Set SYSCFG.BPEN // Re-enable the branch predictor

Even in the absence of this anomaly, it is good programming practice to reset the branch predictor when code modifications occur. This will clear invalid prediction tables and typically result in better performance.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
1.0, 1.1

16. 19000051 - Internal DMC PHY DLL May Not Lock to the New DCLK Frequency:

DESCRIPTION:

The DLL in the DMC PHY is responsible for generating the required phase between the strobe (DQS0/DQS1) and data (DQn) signals when driven out of the controller. Whenever the DCLK frequency is modified using the CGU, the DLL is expected to lock automatically to the new DCLK frequency. Because of this anomaly, the DLL may not lock to the new DCLK frequency; thus, the controller may not generate the required phase between the DQS0/DQS1 and DQn signals, which can result in data corruption when trying to access a DDR2 or LPDDR memory device.

The boot code stored in ROM does not work around this anomaly. Consequently, any boot or pre-boot features that rely on the boot code to program the DMC are not functional. This list of non-functional features includes:

1. DMC configuration through values stored in OTP.
2. Wake-up actions - These allow DMC or CGU settings to be restored after hibernate from the DPM restore registers.

Because these two features are not functional the dmcEN field in OTP memory and the WUA_EN field in DPM_RESTORE0 should not be set.

WORKAROUND:

To avoid the issue, the DLL must be held in reset during any CGU programming that results in a change to the DCLK frequency. Once a valid Init_cgu function is in place, perform the workaround using the following C code:

```
*pREG_DMC_PHY_CTL0|=0x800;  // Set bit 11 of the DMC_PHY_CTL0 register
Init_cgu();                 // Program the CGU to change the DCLK frequency
*pREG_DMC_PHY_CTL0&=~0x800; // Clear bit 11 of the DMC_PHY_CTL0 register
```

This workaround may be used in initcode to allow the DMC to be initialized before the full application has been loaded during a cold boot or after a wake from hibernate.

APPLIES TO REVISION(S):

1.0, 1.1

17. 19000054 - Dynamic Branch Prediction During Self-Nested ISRs Causes Unpredictable Results:

DESCRIPTION:

The System Event Controller (SEC) is a single source that raises interrupt requests to the core at level 11 (IVG11). In order to support the SEC's ability to forward higher-priority interrupt requests to the core while the core is servicing a lower-priority interrupt, the IVG11 ISR must be able to be interrupted by itself, which means self-nesting must be enabled.

Consider the case where interrupt source A is higher-priority than interrupt source B coming from the SEC. When the IVG11 ISR is servicing interrupt B, interrupt A occurs. Due to the self-nesting nature of the IVG11 ISR, the IVG11 ISR code is executed again to service interrupt A. If dynamic branch prediction is enabled during this execution, and the nested interrupt occurs exactly between the source and destination instructions of a branch (JUMPs, CALLs, or RTS) in the interrupt handler, the self-nesting state of the ISR can be lost, thus resulting in unpredictable code flow, which can lead to various run-time failures (including exceptions).

WORKAROUND:

The simplest workaround is to globally disable dynamic branch prediction by clearing the SYSCFG.BPEN bit. However, this may lead to reduced core performance.

An alternative is to disable dynamic prediction only in self-nested interrupt service routines. This can be accomplished by modifying the interrupt dispatcher code that is the wrapper for the IVG11 ISR. The code should use two global variables - one to keep track of the ISR nesting level and one to store the BP-enable bit in the outermost ISR. The workaround requires that code be added to both the prologue and the epilogue for each self-nesting ISR.

In the self-nesting ISR prologue:

```
// Keep a count of how deeply nested the application is running at
R0 = [_isr_nest_count];
CC = R0;
R0 += 1;
[_isr_nest_count] = R0;

// Save the BPEN bit for the outer ISR only, and disable BPEN.
IF CC JUMP _already_nested;
   R0 = SYSCFG;
   R1 = BITM_SYSCFG_BPEN;
   R1 = R0 & R1;
   [_prev_bp_enable] = R1;       // Save the BPEN bit
   BITCLR(R0, BITP_SYSCFG_BPEN);
   CSYNC;                        // Make sure pipeline is clear before disabling BP
   SYSCFG = R0;
_already_nested:                 // Normal dispatcher code goes here
```

Then, in the self-nesting ISR epilogue:

```
// Decrement nest count and restore BP enable bit at the outermost level
R0 = [_isr_nest_count];
R0 += -1;
CC = R0;
[_isr_nest_count] = R0;
IF CC JUMP _still_nested;
   R1 = [_prev_bp_enable];
   R0 = SYSCFG;
   R0 = R0 | R1;
   SYSCFG = R0;
_still_nested:                   // Rest of dispatcher epilogue goes here
```

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):

1.0

18. 19000055 - Core Writes to CAN MMRs May Not Occur:

DESCRIPTION:
When the CAN bus is active, all frames are sampled whether or not the CAN is configured to receive the message or respond to a remote frame request. An internal mailbox comparison run is initiated after the frame header is latched (during the r0 bit for standard frames, and during the r1 bit for extended frames) to determine whether or not the active message is a match to any enabled receive mailbox in the CAN block. During this sensitive window where the internal scan is taking place, core writes to the CAN MMR address space may not occur, and lost writes will go undetected.

Core reads of the CAN MMR space are not affected by this anomaly.

WORKAROUND:
The anomaly does not occur when the CAN module is electrically disconnected from the CAN bus.

When electrically connected to the CAN bus, it is impossible to determine when the internal comparison run is taking place. As a result, all writes to the CAN MMR space must be implemented as an atomic triple-write access in order to guarantee that the write occurs outside the sensitive window, per the following pseudo-code:

```
CAN_TRIPLE_WRITE(CAN_MMR_ADDRESS, Value):
   DISABLE_INTERRUPTS;
   CAN_MMR_ADDRESS = Value;
   CAN_MMR_ADDRESS = Value;
   CAN_MMR_ADDRESS = Value;
   SSYNC;
   ENABLE_INTERRUPTS;
```

While this workaround guarantees that all core writes will occur properly to the CAN MMR space, it introduces the potential for multiple writes to occur where only a single write should have been made. Though not an issue in most cases, this is problematic when considering writes to W1C registers like CAN_RMP and CAN_MBRIF when processing a received message. For these registers, if one of the first two writes occurs properly and the hardware then resets the same bit due to the arrival of the next message to the same mailbox, a subsequent write from the workaround code will now erroneously clear the status bit again immediately without actually processing the newly arrived data, and there will be no indication that the second message was not processed. For this case, software can further utilize the Temporary Mailbox Disable feature prior to writing the CAN_RMP or CAN_MBRIF registers (e.g., for clearing the RMP bit for RX mailbox 2):

```
CAN_TRIPLE_WRITE(REG_CAN0_MBTD, (BITM_CAN_MBTD_TDR|MB_NUM)); // MB_NUM = 0x2
while(!(*pREG_CAN0_MBTD & BITM_CAN_MBTD_TDA));               // Poll for ACK
CAN_TRIPLE_WRITE(REG_CAN0_RMP1, BITM_CAN_RMP1_MB02);         // Clear RMP2
```

With this additional code, a second message destined for the same receive mailbox as the first message will not result in an undetected lost message due to the required workaround code. If the 2nd message is in the process of being stored at the time that the temporary disable request is executed, it will be received completely (and the appropriate CAN_RML bit will be set). However, if the temporary disable request occurs before the second message becomes ongoing (at the start of the DLC field of the active message), then a second receive mailbox must also be enabled to receive the same message ID as the first message, otherwise the second message will not be received due to there being no match to an enabled RX mailbox in the comparison run that establishes which message center to store to.

APPLIES TO REVISION(S):
1.0

19. 19000057 - Emulator Cannot Unlock Locked Processors if Other Devices are Present in JTAG Scan Chain:

DESCRIPTION:
This issue occurs only if the processor has been locked by using the Lock API to write to OTP memory. Once the processor has been locked, it cannot be unlocked by the emulator if any other devices are present in the JTAG scan chain.

WORKAROUND:
None

APPLIES TO REVISION(S):
1.0

20. 19000058 - System MMR Reads Can Cause Core Hang:

DESCRIPTION:

A non-speculative system MMR read can hang the core if a very specific set of unpredictable system conditions manifests around a change in program flow. When the hang condition occurs, the read pends forever unless interrupted by a non-maskable event such as an NMI or emulation interrupt.

The root cause of the issue is related to the alignment of the System MMR read instruction in memory and the behavior of the pre-fetch unit governed by the state of the pipeline.

This hang condition is very rare but is easily reproducible if the system timing aligns precisely as required for it to occur during run-time (time-to-failure will be consistent and observed on any device). Designs subjected to longevity testing exceeding the expected run-time for the application that did not encounter this issue during test are highly unlikely to ever encounter it.

This anomaly does not apply to core MMR accesses and system MMR writes.

WORKAROUND:

The problem is avoided if the System MMR read is buffered on both sides by NOPs and aligned to an 8-byte boundary in memory, depending on the width of the instruction that actually performs the access to the MMR.

For the 64-bit immediate load instruction:

```
.align 8;
DO_SYSMMR_READ:
  NOP;
  SYSMMR_READ;   // 64-bit immediate address access instruction: Rx = [32-bit address];
  NOP; NOP; NOP;
```

For the 16-bit indirect load instruction:

```
.align 8;
DO_SYSMMR_READ:
  SYSMMR_READ;   // 16-bit indirect access instruction: Rx = [Px + offset];
  NOP; NOP; NOP;
```

For the 32-bit indirect load instruction:

```
.align 8;
DO_SYSMMR_READ:
  NOP;
  MNOP;
  SYSMMR_READ;   // 32-bit indirect access instruction: Rx = [Px + offset];
  NOP; NOP; NOP;
```

If the system MMR read is performed in the context of a hardware loop, ensure that the above sequences do not span the loop bottom.

This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and operating systems supported by Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):

1.0, 1.1

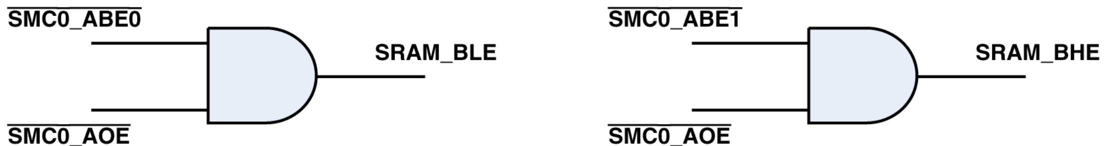
21. 19000059 - SMC Byte Enable Signals Tri-State During Read Operations:

DESCRIPTION:

During SMC read operations, the byte enable signals (SMC0_ABE0 and SMC0_ABE1) are tri-stated instead of being driven low. Therefore, when an 8-bit SMC write access is followed by a 16-bit or 32-bit read access, the read access may fail if the device requires active low byte enable signals during read operations.

WORKAROUND:

While interfacing with the external SRAM, the SRAM byte enable signals can be driven low during read operations using external logic as shown in the figure.



For SMC read operations, the SMC0_AOE signal is low. This drives the SRAM_BHE and SRAM_BLE signals low.
This external logic does not affect the SMC write operations, as the SMC0_AOE signal is high during write operations.

APPLIES TO REVISION(S):

1.0

©2016 Analog Devices, Inc. All rights reserved. Trademarks and registered trademarks are the property of their respective owners.

ANALOG DEVICES

www.analog.com