# Arria® V SX and ST SoC Errata

2015.07.14

ES-1041 Subscribe Send Feedback

This document lists the errata for the Arria® V SX and ST SoC devices. Included in this document are the following errata:

- Altera®-specific Arria V SX and ST SoC errata, which includes the Hard Processor System (HPS) and the FPGA.
- Arm* Cortex*-A9 MPCore*
- Arm L2 cache

**Note:** To obtain third-party IP errata that applies to the HPS and is under NDA, please contact Intel® or your local field representative.

## Altera-Specific SoC Errata for Arria V SX and ST Devices

This section lists the Altera-specific SoC Errata that apply to the Hard Processor System (HPS) and the FPGA. Each listed erratum has an associated status which identifies any planned fixes.

**Table 1: Arria V SX and ST Altera-Specific HPS Errata Summary**

| Issue | Affected Devices | Planned Fix |
|---|---|---|
| | **Hard Processor System (HPS)** | |
| EMAC RMII PHY Interface is Only Supported Through the FPGA Fabric on page 3 | All Arria V SX and ST Devices | None |
| Hard Processor System Level 2 Cache Error Correction Code on page 4 | All Arria V SX and ST Devices | Rev C Silicon |
| Hard Processor System PLL Lock Issue After Power-on Reset or Cold Reset on page 5 | All Arria V SX and ST Devices | Rev D silicon: March, 2015 |
| HPS TAP Controller Is Reset By Cold Reset on page 7 | All Arria V SX and ST Devices | None |
| SPI Slave Output Signals Cannot Be Isolated When Routed to the HPS Pins on page 8 | All Arria V SX and ST Devices | None |
| | **FPGA** | |

| Issue | Affected Devices | Planned Fix |
| --- | --- | --- |
| **False Configuration Failure in Active Serial Multi-Device Configurations** on page 9 | All Arria V SX and ST Devices | None |

# Hard Processor System (HPS)

## EMAC RMII PHY Interface is Only Supported Through the FPGA Fabric

### Description

The default setting of the `physel_x` field in the System Manager EMAC Control Group's `ctrl` register cannot be used to configure an HPS I/O RMII PHY interface. Because the HPS I/O timings do not support RMII protocol, encodings `0x0` and `0x1` are the only valid values in the `physel_x` field. Selecting the `0x0` encoding routes the GMII/MII signals to the FPGA fabric only, and selecting the `0x1` encoding routes the RGMII interface to the HPS I/O only. If the `physel_x` encoding is left as `0x2`, the HPS PHY interface does not function properly.

### Workaround

If an RMII PHY interface is required, the `physel_x` field should be set to `0x0` so that the GMII/MII signals are routed to the FPGA. You can design an RMII soft adaptor in the FPGA configuration file that converts these MII signals to an RMII PHY interface that is mapped to the FPGA I/O pins. Refer to the "Programming Model" section of the EMAC chapter in the *Volume 3: Hard Processor System Technical Reference Manual* for more information about how to initialize the EMAC Controller and interface.

### Status

Affects: All Arria V SX and ST devices

Status: No planned fix

**Related Information**

**Arria V Device Handbook, Volume 3: Hard Processor System Technical Reference Manual**
For more information about how to initialize the EMAC Controller and interface, refer to the "Programming Model" section of the Ethernet Media Access Controller Chapter.

## Hard Processor System Level 2 Cache Error Correction Code

### Description

After enabling the L2 cache ECC feature, false ECC errors may occur.

### Workaround

For affected devices, L2 cache ECC can be used and this issue avoided by setting the `mpu_base_clk` to a maximum frequency as follows:

- Fast speed grade (-4) — 500 MHz
- Mid speed grade (-5) — 400 MHz
- Slow speed grade (-6) — 300 MHz

**Note:** If you are not using the L2 ECC feature, refer to the *Arria V Device Datasheet* for the maximum frequency of the `mpu_base_clk`.

### Status

Affects: All Arria V SX and ST devices

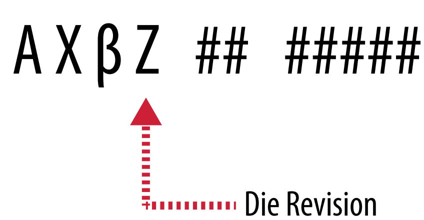
Status: Fixed in Rev C silicon

**Table 2: Device and Revision Fixed**

This table identifies the fixed silicon by die revision for each device.

| Device | Revision without Fix | Revision with Fix |
|---|---|---|
| 5ASXB3 | Rev A & B | Rev C |
| 5ASXB5 | Rev A & B | Rev C |
| 5ASTD3 | Rev A & B | Rev C |
| 5ASTD5 | Rev A & B | Rev C |

**Figure 1: Altera Date Code Marking Format**




**Related Information**

**[Arria V Device Datasheet]**

For more information about the `mpu_base_clk` frequency, refer to the *Arria V Device Datasheet*.

## Hard Processor System PLL Lock Issue After Power-on Reset or Cold Reset

### Description

One or more of the PLLs in the hard processor system (HPS) of Arria V SoC devices can take a long time to lock after power-on reset or cold reset. This occurs when the clock select (CSEL) pins are set to 01, 10, or 11. Some peripherals clocked by the HPS PLLs may fail to operate properly. While this failure is rare, typical symptoms include:

- The HPS hangs during the Boot ROM stage and is unable to proceed to the Preloader stage.
- An intermittent SDRAM calibration error in the Preloader.

### Workaround

These issues can be resolved by implementing the following changes:

1. Connect the CSEL pins `[1:0]` to pull down resistors (4.7 kΩ to 10 kΩ) on the board to force the CSEL input to be 0.
2. Download and install the appropriate SoCEDS patch for software version 13.1 or 14.0. Then, recompile the Preloader code.

To download and install the required SoCEDS patch for SoCEDS version 13.1 and 14.0, refer to **KDB solution rd06202014**. Software in the patch is integrated into SoCEDS version 14.0.1.

Selecting CSEL=00 causes the Boot ROM to bypass the PLLs on cold reset, instead using the external clock input (`osc_1_clk`) for the peripheral interfaces. Because the external clock is already stable, using it ensures the Preloader code loads properly from external Flash.

The software patch adds code to the Preloader, which locks the PLLs quickly, which in turn resolves the intermittent SDRAM calibration issue. It also loads a piece of code into the on-chip RAM to handle a warm reset. Clocks are handled differently depending on whether the reset is warm or cold.

Bypassing the PLLs after cold reset increases the boot time slightly because the external clock is slower than the Flash interface clock generated from the HPS peripheral PLL. Altera reserves the upper 4 KB of on-chip RAM for the warm boot code.

**Note:** Do not overwrite the content of the upper 4 KB of the address range in the on-chip RAM. If you need a smaller memory footprint for the code, file a service request using **mySupport**.

The on-chip memory restriction can be eliminated entirely if the SoC warm and cold reset pins are tied together, or if the HPS boots from FPGA memory.

### Status

Affects: All Arria V SX and ST devices

Status: Fixed in Rev D silicon

**Table 3: Device and Revision Fixed**

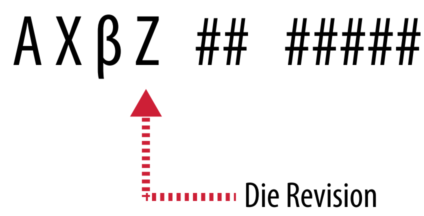
This table identifies the fixed silicon by die revision for each device.

| Device | Revision without Fix | Revision with Fix |
|---|---|---|
| 5ASXB3 | Rev C | Rev D |
| 5ASXB5 | Rev C | Rev D |
| 5ASTD3 | Rev C | Rev D |

| Device | Revision without Fix | Revision with Fix |
|---|---|---|
| 5ASTD5 | Rev C | Rev D |

**Figure 2: Altera Date Code Marking Format**

This figure shows the silicon revision as identified by the fourth letter of the lot ID.



## HPS TAP Controller Is Reset By Cold Reset

### Description

The HPS TAP controller should be only be reset by a JTAG reset and should remain active during all other resets. Because of this erratum, the HPS TAP controller is erroneously reset by a cold reset. The HPS JTAG controller will not be visible while HPS is in a cold reset. FPGA JTAG is unaffected.

### Workaround

This issue typically will not cause a problem for debugging applications executing in the HPS because you usually do not debug code through a reset. To have access to the HPS JTAG chain, release HPS from cold reset.

### Status

Affects: All Arria V SX and ST devices

Status: No planned fix

## SPI Slave Output Signals Cannot Be Isolated When Routed to the HPS Pins

### Description

The SPI output enable is not connected to the SPI HPS pins. Because of this error, the peripheral cannot isolate itself from the SPI bus when routed to the HPS pins. As a result, the HPS `SPIS_TXD` pin cannot be tri-stated by setting the `slv_oe` bit (bit 10) in the `ctrlr0` register to 1.

### Workaround

Route the SPI slave signals to the FPGA fabric. Because the output enable signal is exposed through the FPGA fabric, you can connect it to an FPGA tri-state pin and enable tri-stating when necessary.

### Status

Affects: All Arria V SX and ST devices

Status: No planned fix

# FPGA

## False Configuration Failure in Active Serial Multi-Device Configurations

### Description

In Active Serial (AS) multi-device configuration mode, the error checking for `CONF_DONE` release may not operate correctly. As a result, you may experience false configuration errors. The failure is indicated by the `CONF_DONE` going high, followed by the `nSTATUS` going low and reconfiguration repeatedly initiated.

### Workaround

To resolve this issue, perform both of the following:

1. Disable the `CONF_DONE` error checking in AS multi-device configuration mode:
   a. If you are using Quartus II software version 12.0 or older, check the "Disable AS mode CONF_DONE error check" option. This option can be found in the "Advanced" button, under the Convert Programming File window.
   b. If you are using Quartus II version 12.0 SP1 or later, the error checking is disabled automatically for AS multi-device configuration POF file generation.
2. Enable the `INIT_DONE` pin option:
   a. To ensure a successful configuration, Altera recommends that you enable the `INIT_DONE` optional pin for devices in the configuration chain. On the board, route out the `INIT_DONE` pin separately for both the master and slave devices. Monitor the `INIT_DONE` status for each of the devices to ensure a successful transition into User-mode.

**Note:** Other configuration modes (JTAG, Fast Passive Parallel (FPP), and Passive Serial (PS) (single and multi device configurations, and AS single-device configurations) are not affected.

### Status

Affects: All Arria V devices

Status: No fix planned

# Arm Cortex-A9 MPCore and L2 Cache Errata

This section lists the Arm Cortex-A9 MPCore and L2 Cache errata. Each listed erratum has an associated an associated category number which identifies the degree of the behavior.

The categories are as follows:

- Category 1: Behavior has no workaround and severely restricts the use of the product in all, or the majority of applications, rendering the device unusable.
- Category 2: Behavior contravenes the specified behavior and might limit or severely impair the intended use of the specified features, but does not render the product unusable in all or the majority of applications.
- Category 3: Behavior that was not the originally intended behavior but should not cause any problems in applications.

**Note:** There are no Category 1 Errata listed in this document.

**Table 4: Arm Cortex-A9 MPCore Errata**

| Errata Listing | Category Number |
|---|---|
| **Arm Cortex-A9 MPU** | |
| 761319: Ordering of Read Accesses to the Same Memory Location Might Be Uncertain on page 13 | Category 2 |
| 775420: Particular Data Cache Maintenance Operation Which Aborts Might Lead to Deadlock on page 14 | Category 2 |
| 782772: Speculative Execution of LDREX or STREX Instruction After a Write to Strongly Ordered Memory Might Lead to Deadlock on page 15 | Category 2 |
| 761320: Full Cache Line Writes to the Same Memory Region From Both Processors Might Cause Deadlock on page 16 | Category 2 |
| 845369: Under Very Rare Timing Circumstances Transition into Streaming Mode Might Create Data Corruption on page 17 | Category 2 |
| 740657: Global Timer Can Send Two Interrupts for the Same Event on page 18 | Category 3 |
| 751476: Missed Watchpoint on the Second Part of an Unaligned Access Crossing a Page Boundary on page 19 | Category 3 |
| 754322: Faulty MMU Translations Following ASID Switch on page 20 | Category 3 |
| 764369: Data or Unified Cache Line Maintenance by MVA Fails on Inner-Shareable Memory on page 22 | Category 3 |
| 782773: Updating a Translation Entry to Move a Page Mapping Might Erroneously Cause an Unexpected Translation Fault on page 24 | Category 3 |
| 794072: A Short Loop Including DMB Instruction Might Cause a Denial of Service When the Other Processor Executes a CP15 Broadcast Operation on page 25 | Category 3 |
| 794073: Speculative Instruction Fetches with MMU Disabled Might Not Comply with Architectural Requirements on page 27 | Category 3 |

| Errata Listing | Category Number |
|---|---|
| 794074: A Write Request to an Uncacheable, Shareable Normal Memory Region Might be Executed Twice, Possibly Causing a Software Synchronization Issue on page 28 | Category 3 |
| 725631: ISB is Counted in Performance Monitor Events 0x0C and 0x0D on page 30 | Category 3 |
| 729817: MainID Register Alias Addresses Are Not Mapped on Debug APB Interface on page 31 | Category 3 |
| 729818: In Debug State, the Next Instruction is Stalled When the SDABORT Flag is Set Instead of Being Discarded on page 32 | Category 3 |
| 751471: DBGPCSR Format Is Incorrect on page 33 | Category 3 |
| 752519: An Imprecise Abort Might Be Reported Twice on Non-Cacheable Reads on page 34 | Category 3 |
| 754323: Repeated Store in the Same Cache Line Might Delay the Visibility of the Store on page 35 | Category 3 |
| 756421: Sticky Pipeline Advance Bit Cannot be Cleared from Debug APB Accesses on page 36 | Category 3 |
| 757119: Some Unallocated Memory Hint Instructions Generate an UNDEFINED Exception Instead of Being Treated as a NOP on page 37 | Category 3 |
| 761321: MRC and MCR Are Not Counted in Event 0x68 on page 38 | Category 3 |
| 764319: Read Accesses to DBGPRSR and DBGPRCR May Generate an Unexpected UNDEF on page 39 | Category 3 |
| 771221: PLD Instructions Might Allocate Data in the Data Cache Regardless of the Cache Enable Bit Value on page 40 | Category 3 |
| 771224: Visibility of Debug Enable Access Rights to Enable/Disable Tracking is Not Ensured by an ISB on page 41 | Category 3 |
| 771225: Speculative Cacheable Reads to Aborting Memory Regions Clear the Internal Exclusive Monitor and May Lead to Livelock on page 42 | Category 3 |
| 775419: PMU Event 0x0A Might Count Twice the LDM PC ^ Instruction with Base Address Register Write-Back on page 44 | Category 3 |
| 782774: A Spurious Event 0x63 Can be Reported on an LDREX That is preceded by a Write to Strongly Ordered Memory Region on page 45 | Category 3 |
| **Arm L2 Cache Controller** | |
| 754670: A Continuous Write Flow Can Stall a Read Targeting the Same Memory Area on page 46 | Category 3 |
| 765569: Prefetcher Can Cross 4 KB Boundary if Offset is Programmed with Value 23 on page 47 | Category 3 |
| 729815: The High Priority for SO and Dev Reads Feature Can Cause Quality of Service Issues to Cacheable Read Transactions on page 48 | Category 3 |

| Errata Listing | Category Number |
| --- | --- |
| **Arm CoreSight Program Trace Macrocell (PTM)** | |
| **720107: Periodic Synchronization Can Be Delayed and Cause Overflow** on page 49 | Category 3 |
| **711668: Configuration Extension Register Has Wrong Value Status** on page 51 | Category 3 |

# Arm Cortex-A9 MPU

## 761319: Ordering of Read Accesses to the Same Memory Location Might Be Uncertain

### Description

The Arm architecture and the general rules of coherency require reads to the same memory location to be observed in sequential order. Because of some internal replay path mechanisms, the Cortex-A9 can see one read access bypassed by a following read access to the same memory location, thus not observing the values in program order.

### Impact

This erratum:

- Applies only to devices with a dual Cortex-A9 MPCore configuration.
- Can occur only on a process working in SMP mode on memory regions marked as normal memory write-back shared.
- Can cause data coherency failure.

### Workaround

The majority of multi-processing code examples follow styles that do not expose this erratum. Therefore, this erratum occurs rarely and is likely to affect only very specific areas of code that rely on a read-ordering behavior. There are two possible workarounds for this erratum:

- Use `LDREX` instead of standard `LDR` in volatile memory places that require a strict read-ordering.
- The alternative workaround is the recommended workaround for tool chain integration. This method requires insertion of a `DMB` between the affected `LDR` that requires this strict ordering rule.

For more information about integrating the workaround inside tool chains, please refer to the Programmer Advance Notice related to this erratum, *ARM UAN 0004A*.

### Category

Category 2

# 775420: Particular Data Cache Maintenance Operation Which Aborts Might Lead to Deadlock

## Description

Under certain micro-architectural circumstances, a data cache maintenance operation which aborts, followed by an `ISB` with no `DSB` occurring between these events, might light lead to processor deadlock.

This erratum requires the following conditions:

1. Some of the write operations are being handled by the processor and take a long time to complete. The typical situation is when the write operation, such as `STR`, `STM`, has missed in the L1 data cache.
2. No memory barrier (`DMB` or `DSB`) is inserted between the write operation and the data cache maintenance operation mentioned in condition 3.
3. A data cache maintenance operation is performed, which aborts because of its MMU settings.
4. No memory barrier (`DMB` or `DSB`) is inserted between the data cache maintenance operation in condition 3 and the `ISB` in condition 5. Any other kind of code can be executed here, starting with the abort exception handler following the aborted cache maintenance operation.
5. An `DMBISB` instruction is being executed by the processor.
6. No memory barrier ( or `DSB`) is inserted between the ISB in condition 5 and the read or write operation in condition 7.
7. A read or write operation is executed.

With the above conditions, an internal "Data Side drain request" signal might remain sticky causing the `ISB` to wait for the data side to empty, which never happens because the last read or write operation waits for the `ISB` to complete.

## Impact

This erratum can lead to processor deadlock.

## Workaround

A simple workaround for this erratum is to add a `DSB` at the beginning of the abort exception handler.

## Category

Category 2

# 782772: Speculative Execution of LDREX or STREX Instruction After a Write to Strongly Ordered Memory Might Lead to Deadlock

## Description

Under certain timing circumstances, a processor might deadlock when the execution of a write to a strongly-ordered memory region is followed by the speculative execution of a Load-Exclusive (`LDREX`) or a Store-Exclusive (`STREX`) instruction that is not speculated correctly. This incorrect speculation can be due to either the `LDREX` or `STREX` instruction being conditional and failing its condition code check or to the `LDREX` or `STREX` instruction being speculatively executed in the shadow of a mispredicted branch.

This erratum requires the following conditions:

- The processor executes a write instruction to a strongly-ordered memory region.
- The process speculatively executes a Load-Exclusive or Store-Exclusive instruction that is either:
  - A conditional instruction
  - An instruction in the shadow of a conditional branch.
- The Load-Exclusive or Store-Exclusive instruction is canceled because the speculation was incorrect, due to one of the following conditions:
  - The conditional Load-Exclusive or Store-Exclusive instruction failed its condition-code check.
  - The conditional branch was mispredicted so that all subsequent instructions speculatively executed must be flushed, including the Load-Exclusive or Store-Exclusive.

This erratum also requires additional timing conditions to be met. These are specific to each platform, and are not controllable by software. One timing condition that can cause this erratum is when the response to the strongly-ordered write from the external memory system must be received at the same time as the mispeculation is identified in the processor.

## Impact

This erratum can cause a processor deadlock.

## Workaround

The recommended workaround is to place a `DMB` instruction before each Load-Exclusive/Store-Exclusive loop sequence, to ensure that no pending write request can interfere with the executing of the `LDREX` or `STREX` instructions. The implementation of this workaround can be restricted to code regions which have access to strongly-ordered memory.

## Category

Category 2

# 761320: Full Cache Line Writes to the Same Memory Region From Both Processors Might Cause Deadlock

## Description

Under very rare circumstances, full cache line writes from the two processors on cache lines in hazard with other request may cause arbitration issues in the SCU, leading to processor deadlock.

This erratum only affects dual-core devices. To trigger this erratum, the two processors must be performing full cache line writes to cache lines which are in coherent memory regions and are in hazard with other access requests in the Snoop Control Unit (SCU). The hazard in the SCU happens when the Accelerator Coherency Port (ACP) is performing a read or a write of the same cache line.

Under certain rare timing circumstances, the requests might create a loop of dependencies causing a processor deadlock.

## Impact

This erratum can cause system deadlock. It is important to note that any scenario leading to this deadlock situation is uncommon. It requires both processors writing full cache lines to a coherent memory region, without taking any semaphore, with the ACP assessing the same lines at the same time, meaning that these latter accesses are not deterministic. This condition, combined with the extremely rare microarchitectural timing conditions under which the defect can happen, explains why this erratum is not expected to cause any significant malfunction in real systems.

## Workaround

This erratum can be worked around by setting bit[21] of the undocumented `Diagnostic Control Register` to 1. This register is encoded as CP15 c15 0 c0 1.

This bit can be written in secure state only, with the following read/modify/write code sequence:

```
MCR p15, 0, rt, c15, c0, 1
ORR rt, rt #0x200000
MCR p15, 0, rt, c15, c0, 1
```

When this bit is set, the "direct eviction" optimization in the bus interface unit is disabled, which means this erratum cannot occur.

Setting this bit might prevent the Cortex-A9 from utilizing the full bandwidth when performing intensive full cache line writes and therefore a slight performance drop might be visible.

In addition, this erratum cannot occur if at least one of the following bits in the diagnostic control register is set to 1:

- bit[23]: Disable read-allocate mode
- bit[22]: Disable write-allocate wait mode

## Category

Category 2

## 845369: Under Very Rare Timing Circumstances Transition into Streaming Mode Might Create Data Corruption

### Description

Under very rare timing circumstances, data corruption might occur on a dirty cache line that is evicted from the L1 data cache due to another cache line being entirely written.

The erratum requires the following conditions:

- The CPU contains a dirty line in its data cache.
- The CPU performs at least four full cache line writes, one of which is causing the eviction of the dirty line.
- The other CPU, or the ACP, is performing a read or write operation on the dirty line.

The issue requires very rare timing conditions to reach the point of failure. These timing conditions depend on the CPU micro-architecture, and are not controllable in software:

- The CPU must be in a transitional mode that might be triggered by the detection of the first two full cache line writes.
- The evicted line must remain stalled in the eviction buffer, which is likely to be caused by congested write traffic.
- The other coherent agent, either the other CPU or the ACP, must perform its coherency request on the evicted line while it is in the eviction buffer.

### Impact

This erratum might lead to data corruption.

### Workaround

A workaround for this erratum is provided by setting bit[22] of the undocumented `Diagnostic Control Register` to 1. This register is encoded as `CP15 c15 0 c0 1`. The bit can be written in secure state only, with the following read-modify-write code sequence:

```
MRC p15,0,rt,c15,c0,1
ORR rt,rt,#0x00400000
MCR p15,0,rt,c15,c0,1
```

When this bit is set, the processor is unable to switch into read-allocate (streaming) mode, which means this erratum cannot occur.

Setting this bit could possibly result in a visible drop in performance for routines that perform intensive memory accesses, such as `memset()` or `memcpy()`. However, the workaround is not expected to create any significant performance degradation in most standard applications.

### Category

Category 2

## 740657: Global Timer Can Send Two Interrupts for the Same Event

### Description

The global timer can be programmed to generate an interrupt request to the processor when it reaches a given programmed value. The timer may generate two interrupt requests instead of one, if you program the global timer not to use the auto-increment feature.

The `Global Timer Control` register is programmed with the following settings:

- When Bit[3]=0, the global timer is programmed in "single-shot" mode.
- When Bit[2]=1, the global timer IRQ generation is enabled.
- When bit[1]= 1, the global timer value comparison with the Comparator registers is enabled.
- When bit[0]= 1, the global timer count is enabled.

With these settings, an IRQ is generated to the processor when the global timer value reaches the value programmed in the Comparator registers. The interrupt handler then performs the following sequence:

1. Read the `ICCIAR` (Interrupt Acknowledge) register.
2. Clear the global timer flag.
3. Modify the comparator value, to set it to a higher value.
4. Write the `ICCEOIR` (End of Interrupt) register.

Under these conditions, because of this erratum, the global timer might generate a second (spurious) interrupt request to the processor at the end of this interrupt handler sequence.

### Impact

This erratum creates spurious interrupt requests in the system.

### Workaround

Because this erratum happens only when the Global Timer is programmed in "single-shot" mode, that is, when it does not use the auto-increment feature, a first possible workaround is to program the Global Timer to use the auto-increment feature.

If this first solution is not possible, a second workaround is to modify the interrupt handler to avoid the offending sequence. You can achieve this by clearing the global timer flag after incrementing the Comparator register value. The correct code sequence for the interrupt handler should then look like the following sequence:

1. Read the `ICCIAR` (Interrupt Acknowledge) register.
2. Modify the comparator value, to set it to a higher value.
3. Clear the global timer flag.
4. Clear the pending status information for interrupt 27 (Global Timer interrupt) in the distributor of the interrupt controller.
5. Write the `ICCEOIR` (End of Interrupt) register.

### Category

Category 3

## 751476: Missed Watchpoint on the Second Part of an Unaligned Access Crossing a Page Boundary

### Description

Under rare conditions, a watchpoint might be undetected if it occurs on the second part of an unaligned access that crosses a 4K page boundary and misses the µTLB for the second part of its request.

This erratum requires a previous conditional instruction which accesses the second 4 KB memory region (where the watchpoint is set), which misses in the µTLB and causes a condition fail. This erratum also requires that no other µTLB miss occurs between this conditional failed instruction and the unaligned access, which implies that the unaligned access must hit in the µTLB for the first part of its access.

### Impact

A watchpoint does not trigger when it should.

### Workaround

Because this erratum might occur in the case when a watchpoint is set on any of the first 3 bytes of a 4 KB memory region and unaligned accesses are not being faulted, the workaround is to set a guard watchpoint on the last byte of the previous page and to deal with any false positive matches if they occur.

### Category

Category 3

## 754322: Faulty MMU Translations Following ASID Switch

### Description

A µTLB entry might be corrupted following an ASID switch, possibly corrupting subsequent MMU translations.

This erratum requires execution of an explicit memory access that might be speculative. This type of memory access misses in the TLB and causes a translation walk. This erratum occurs when the translation table walk starts before the ASID switch code sequence, but completes after the ASID switch code sequence.

In this case, a new entry is allocated in the µTLB for the TLB entry of this translation table walk, but corresponds to the old ASID. Because the µTLB does not record the ASID value, the new MMU translation that should happen with the new ASID following the ASID switch, might hit this stale µTLB entry and become corrupted.

There is no security risk because the security state of the access is held in the µTLB and cannot be corrupted.

### Impact

This erratum might cause MMU translation corruption.

### Workaround

The workaround for this erratum is to add a `DSB` in the ASID switch code sequence. The Arm architecture only mandates an `ISB` before and after the ASID switch. Adding a `DSB` before the ASID switch ensures that the translation table walk completes before the ASID change, so that no stale entry can be allocated in the µTLB.

Modify the examples in the *Arm Architecture Reference Manual* for synchronizing the change in the ASID and TTBR as follows:

1. The sequence:

```
Change ASID to 0
ISB
Change Translation Table Base Register
ISB
Change ASID to new value
```

Becomes:

```
DSB
Change ASID to 0
ISB
Change Translation Table Base Register
ISB
DSB
Change ASID to new value
```

2. This sequence:

```
Change Translation Table Base Register to the global-only mappings
ISB
Change ASID to new value
ISB
Change Translation Table Base Register to new value
```

Becomes:

```
Change Translation Table Base Register to the global-only mappings
ISB
DSB
Change ASID to new value
ISB
Change Translation Table Base Register to new value
```

3. This sequence:

```
Set TTBCR.PD0 = 1
ISB
Change ASID to new value
Change Translation Table Base Register to new value
ISB
Set TTBCR.PD0 = 0
```

Becomes:

```
Set TTBCR.PD0 = 1
ISB
DSB
Change ASID to new value
Change Translation Table Base Register to new value
ISB
Set TTBCR.PD0 = 0
```

## Category

Category 3

# 764369: Data or Unified Cache Line Maintenance by MVA Fails on Inner-Shareable Memory

## Description

Under certain timing circumstances, a data or unified cache line maintenance operation by Modified Virtual Addresses (MVA) that targets an inner-shareable memory region might fail to propagate to either the Point of Coherency (PoC) or to the Point of Unification (PoU) of the system.

As a consequence, the visibility of the updated data might not be guaranteed to either the instruction side, in the case of self-modifying code, or to an external non-coherent agent, such as a DMA engine.

This erratum requires a dual Cortex-A9 MPCore device working in Symmetric Multi-Processing (SMP) mode with the broadcasting of CP15 maintenance operations enabled.

The following scenario shows how this erratum can occur:

1. One CPU performs a data or unified cache line maintenance operation by MVA targeting a memory region that is locally dirty.
2. The second CPU issues a memory request targeting this same memory location within the same time frame.

A race condition can occur, resulting in the cache operation not being performed to the specified Point of Unification or Point of Coherence.

This erratum affects the following maintenance operations:

- DCIMVAC: Invalidate data or unified cache line by MVA to PoC.
- DCCMVAC: Clean data or unified cache line by MVA to PoC.
- DCCMVAU: Clean data or unified cache line by MVA to PoU.
- DCCIMVAC: Clean and invalidate data or unified cache line by MVA to PoC.

This erratum can occur when the second CPU performs any of the following operations:

- A read request resulting from a Load instruction; the Load might be a speculative one.
- A write request resulting from any Store instruction.
- A data prefetch resulting from a PLD instruction; the PLD might be a speculative one.

## Impact

Because it is uncertain whether execution of the cache maintenance operation propagates to either the Point of Unification or the Point of Coherence, stale data might remain in the data cache and not become visible to other agents that should have gained visibility to it.

Note that the data remains coherent on the L1 data side. Any data read from the other processor in the Cortex-A9 MPCore cluster, or from the Accelerator Coherency Port (ACP), would see the correct data. In the same way, any write to the same cache line from the other processor in the Cortex-A9 MPCore cluster, or from the ACP, does not cause a data corruption resulting from a loss of either data.

Consequently, the failure can only impact non-coherent agents in the system. These agents can be either the instruction cache of the processor, in the case of self-modifying code, or any non-coherent external agent in the system such as a DMA.

## Workaround

Two workarounds are available for this erratum.

The first workaround requires the three following elements to be applied together:

- Set bit[0] in the undocumented `SCU Diagnostic Control` register located at offset 0x30 from the PERIPHBASE address. Setting this bit disables the "migratory line" feature and forces a dirty cache line to be evicted to the lower memory subsystem, which is both the Point of Coherency and the Point of Unification, when it is being read by another processor. Note that this bit can be written, but is always read as zero.
- Insert a `DSB` instruction before the cache maintenance operation. Note that if the cache maintenance operation executes within a loop that performs no other memory operations, ARM recommends only adding a `DSB` before entering the loop.
- Ensure there is no false sharing (on a cache line size alignment) for self-modifying code or for data produced for external non-coherent agent such as a DMA engine. For systems that cannot prevent false sharing in these regions, this third step can be replaced by performing the sequence of `DSB` followed by a cache maintenance operation twice.

Note that even when all three components of the workaround are in place, this erratum might still occur. However, this occurrence would require some extremely rare and complex timing conditions, so that the probability of reaching the point of failure is extremely low. This low probability, along with the fact that this erratum requires an uncommon software scenario, explains why this workaround is likely to be a reliable practical solution for most systems.

To ARM's knowledge, no failure has been observed in any system when all three components of this workaround have been implemented.

For critical systems that cannot cope with the extremely low failure risks associated with the above workaround, a second workaround is possible which involves changing the mapping of the data being accessed so that it is in a non-cacheable area. This ensures that the written data remains uncached, which means it is always visible to non-coherent agents in the system, or to the instruction side in the case of self-modifying code, without any need for cache maintenance operation.

## Category

Category 3

# 782773: Updating a Translation Entry to Move a Page Mapping Might Erroneously Cause an Unexpected Translation Fault

## Description

Under certain conditions specific to the Cortex-A9 microarchitecture, a write operation that updates a cacheable translation table entry might cause both the old and the new translation entry to be temporarily invisible to translation table walks, thus erroneously causing a translation fault.

This erratum requires the following conditions to happen:

1. The processor has its data cache and MMU enabled.
2. The TTB registers are set to work on memory regions with cacheable descriptors.
3. The processor is updating an existing cacheable translation table entry, and this write operation hits in the L1 data cache.
4. A hardware translation table walk is attempted. The hardware translation table walk can be due to either an instruction fetch, or to any other instruction execution that requires an address translation, including any load or store operation. This hardware translation walk must attempt to access the entry being updated in condition 2, and that access must hit in the L1 data cache.

In practice, this scenario can happen when an OS is changing the mapping of a physical page. The OS might have an existing mapping to a physical page (the old mapping), but wants to move the mapping to a new page (the new mapping). To do this, the OS might:

- Write a new translation entry, without cancelling the old one. At this point, the physical page is accessible using either the old mapping or the new mapping.
- Execute a `DSB` instruction followed by an `ISB` instruction pair, to ensure that the new translation entry is fully visible.
- Remove the old entry.

Because of this erratum, this sequence might fail because it can happen that neither the new mapping, nor the old mapping, is visible after the new entry is written, causing a translation fault.

## Impact

This erratum causes a translation fault.

## Workaround

The recommended workaround is to perform a clean and invalidate operation on the cache line that contains the translation entry before updating the entry, to ensure that the write operation misses in the data cache. This workaround prevents the microarchitectural conditions for this erratum from happening. Interrupts must be temporarily disabled so that no interrupt can be taken between the maintenance operation and the translation entry update to avoid the possibility of the interrupt service routine bringing the cache line back in the cache.

Another possible workaround is to place the translation table entries in non-cacheable memory areas, but this workaround is likely to have a noticeable performance penalty. Note that inserting a `DSB` instruction immediately after writing the new translation table entry significantly reduces the probability of encountering this erratum, but is not a complete workaround.

## Category

Category 3

# 794072: A Short Loop Including DMB Instruction Might Cause a Denial of Service When the Other Processor Executes a CP15 Broadcast Operation

## Description

A processor that continuously executes a short loop containing a `DMB` instruction might prevent a CP15 operation broadcast by the other processor making further progress, causing a denial of service.

This erratum only affects dual Cortex-A9 MPCore devices.

This erratum requires the following conditions:

- A dual core device with the processors working in SMP mode (`ACTLR.SMP`=1).
- One of the processors continuously executes a short loop containing at least one `DMB` instruction.
- The other processor executes a CP15 maintenance operation that is broadcast, meaning that this processor has enabled the broadcasting of CP15 operations (`ACTLR.FW`=1).

For this erratum to occur, the short loop containing the `DMB` instruction must meet both of the following additional conditions:

- No more than 10 instructions other than the `DMB` are executed between each `DMB`.
- No non-conditional Load or Store, or conditional Load or Store that pass the condition code check, are executed between each `DMB`.

When all the conditions for this erratum are met, the short loop creates a continuous stream of `DMB` instructions that may cause a denial of service by preventing the processor executing the short loop from executing the received broadcast CP15 operation. As a result, the processor that originally executed the broadcast CP15 operation is stalled until the execution of the loop is interrupted.

Note that because the process issuing the CP15 broadcast operation cannot complete operation, it cannot enter any debug mode and cannot take any interrupt. If the processor executing the short loop also cannot be interrupted, for example if it has disabled its interrupts, or if no interrupts are routed to this processor, this erratum might cause a system livelock.

## Impact

This erratum might create performance issues, or in the worst case it might cause a system livelock if the processor executing the `DMB` is in an infinite loop that cannot be interrupted.

## Workaround

This erratum can be worked around by setting bit[4] of the undocumented `Diagnostic Control` Register to 1. This register is encoded as CP15 c15 0 c0 1.

This bit can be written in the secure state only, with the following read-modify-write code sequence:

```
MRC p15,0,rt,c15,c0,1
ORR rt,rt,#0x10
MCR p15,0,rt,c15,c0,1
```

When it is set, this bit causes the `DMB` instruction to be decoded and executed like a `DSB`. Using this software workaround is not expected to have any impact on the overall performance of the processor on a typical code base.

Other workarounds are also available for this erratum, to either prevent or interrupt the continuous stream of `DMB` instructions that causes the deadlock. Examples include:

- Inserting a non-conditional Load or Store instruction in the loop between each `DMB`.
- Inserting additional instructions in the loop, such as NOPs, to avoid the processor seeing back to back `DMB` instructions.
- Making the processor that is executing the short loop take regular interrupts.

## Category

Category 3

## 794073: Speculative Instruction Fetches with MMU Disabled Might Not Comply with Architectural Requirements

### Description

When the MMU is disabled, an ARMv7 processor must follow some architectural rules regarding speculative fetches and the addresses to which these fetches can be initiated. These rules avoid potential read accesses to read-sensitive areas. For more information about these rules, see the description of "Behavior of Instruction Fetches When All Associated MMUs Are Disabled" in the *ARM Architecture Reference Manual*, ARMv7-A and ARMv7-R edition.

A Cortex-A9 processor usually operates with both the MMU and branch prediction enabled. If the processor operates in this condition for any significant amount of time, the BTAC (branch target address cache) will contain branch predictions. If the MMU is then disabled, but branch prediction remains enabled, these stale BTAC entries can cause the processor to violate the rules for speculative fetches.

This erratum can occur only if the following sequence of conditions is met:

1. The MMU and branch prediction are enabled.
2. Branches are executed.
3. The MMU is disabled, and branch prediction remains enabled.

### Impact

If the above conditions occur, it is possible that after the MMU is disabled, speculative instruction fetches might occur to read-sensitive locations.

### Workaround

The recommended workaround is to invalidate all entries in the BTAC by executing a `BPIALL` (invalidate entire branch prediction array) operation, followed by a `DSB`, before disabling the MMU. Another possible workaround is to disable branch prediction when disabling the MMU, and keep branch prediction disabled until the MMU is re-enabled.

### Category

Category 3

# 794074: A Write Request to an Uncacheable, Shareable Normal Memory Region Might be Executed Twice, Possibly Causing a Software Synchronization Issue

## Description

Under certain timing circumstances specific to the Cortex-A9 microarchitecture, a write request to an uncacheable, shareable normal memory region might be executed twice, causing the write request to be sent twice on the AXI bus. This condition might happen when the write request is followed by another write into the same naturally doubleword-aligned memory region, without a `DMB` between the two writes.

The repetition of the write usually has no impact on the overall behavior of the system, unless the repeated write is used for synchronization purposes.

This erratum requires the following conditions:

- A write request is performed to an uncacheable, shareable normal memory region.
- Another write request is performed into the same naturally doubleword-aligned memory region. This second write request must not be performed to the exact same bytes as the first store.

A write request to normal memory region is treated as uncacheable in the following cases:

- The write request occurs while the data cache is disabled.
- The write request is targeting a memory region marked as normal memory non-cacheable or cacheable write-through.
- The write request is targeting a memory region marked as normal memory cacheable write-back and shareable, and the CPU is in AMP mode.

## Impact

This erratum might have implications in a multi-master system where control information is passed between several processing elements in memory using a communication variable, such as a semaphore. In this type of system, it is common for communication variables to be claimed using a Load-Exclusive/ Store-Exclusive, but for the communication variable to be cleared using a non-Exclusive store. This erratum means that the clearing of such a communication variable might occur twice. This error might lead to two masters apparently claiming a communication variable, and therefore might cause data corruption to shared data.

A scenario in which this might happen is:

```
MOV r1,#0x40                    ;address is double-word aligned, mapped in
                                ;Normal Non-cacheable Shareable memory
Loop: LDREX r5, [r1,#0x0]       ;read the communication variable
CMP r5, #0                      ;check if 0
STREXEQ r5, r0, [r1]            ;attempt to store new value
CMPEQ r5, #0                    ;test if store succeeded
BNE Loop                        ;retry if not
DMB                             ;ensures that all subsequent accesses are observed
when
                                ;gaining of the communication variable has been
observed
                                ;loads and stores in the critical region can now be
performed
MOV r2,#0
MOV r0, #0
DMB                             ;ensure all previous accesses are observed before the
                                ;communication variable is cleared
STR r0, [r1]                    ;clear the communication variable with normal store
STR r2, [r1,#0x4]               ;previous STR might merge and be sent again, which
                                ;might cause undesired release of the communication
                                ;variable.
```

This scenario is valid when the communication variable is a byte, a half-word, or a word.

## Workaround

There are several possible workarounds:

- Add a `DMB` after clearing a communication variable:

```
STR r0, [r1]     ;clear the communication variable
DMB              ;ensure the previous STR is complete
```

  Also, any IRQ or FIQ handler must execute a `DMB` at the start to ensure that the clear of any communication variable is complete.
- Ensure there is no other data using the same naturally aligned 64-bit memory location as the communication variable:

```
ALIGN 64
communication_variable DCD 0
unsed_data DCD 0
LDR r1 = communication_variable
```

- Use a Store-Exclusive to clear the communication variable, rather than a non-Exclusive store.

## Category

Category 3

## 725631: ISB is Counted in Performance Monitor Events 0x0C and 0x0D

### Description

The `ISB` is implemented as a branch in the Cortex-A9 microarchitecture. Because `ISB` acts as a branch, events 0x0C (software change of PC) and 0x0D (immediate branch) are asserted when an `ISB` occurs, which is not compliant with the Arm architecture.

### Impact

The count of events 0x0C and 0x0D are not completely precise when using the Performance Monitor counters, because the `ISB` is counted together with the real software changes to the PC (for 0x0C) and immediate branches (0x0D).

This erratum also causes the corresponding PMUEVENT bits to toggle in case an `ISB` executes.

- PMUEVENT[13] relates to event 0x0C.
- PMUEVENT[14] relates to event 0x0D.

### Workaround

You can count `ISB` instructions alone with event 0x90.

You can subtract this `ISB` count from the results you obtained in events 0x0C and 0x0D, to obtain the precise count of software change of PC (0x0C) and immediate branches (0x0D).

## 729817: MainID Register Alias Addresses Are Not Mapped on Debug APB Interface

### Description

The Arm Debug Architecture specifies registers 838 and 839 as "Alias of the MainID register." They should be accessible using the APB Debug interface at addresses 0xD18 and 0xD1C. The two alias addresses are not implemented in Cortex-A9. A read access to either of these two addresses returns 0 instead of the MainID register value.

Note that read accesses to these two registers using the internal CP14 interface are trapped to UNDEF, which is compliant with the Arm Debug architecture. Therefore this erratum only applies to the alias addresses using the external Debug APB interface.

### Impact

If the debugger, or any other external agent, tries to read the MainID register using the alias addresses, it receives a faulty answer (0x0), which can cause indeterminate errors in the debugger afterwards.

### Workaround

The workaround for this erratum is to always access the MainID register at its original address, 0xD00 and not to use its alias address.

### Category

Category 3

# 729818: In Debug State, the Next Instruction is Stalled When the SDABORT Flag is Set Instead of Being Discarded

## Description

When the processor is in the debug state, an instruction written to the `Instruction Transfer Register (ITR)` after a Load/Store instruction that has aborted, gets executed on clearing the `SDABORT_l`, instead of being discarded.

This erratum can occur under the following conditions:

- The debugger has put the `extDCCmode` bits into stall mode.
- A previously issued Load/Store instruction has generated a synchronous data abort (for example, an MMU fault).
- For efficiency, the debugger does not read `Debug Status and Control External (DBGDSCRext)` register immediately, to see if the Load/Store has completed and has not aborted, but writes further instructions to the ITR, expecting them to be discarded if a problem occurs.
- The debugger reads the `Debug Status and Control (DBGDSCR)` register at the end of the sequence and discovers the Load/Store aborted.
- The debugger clears the `SDABORT_l` flag (by writing to the `clear sticky aborts` bit in `Debug Run Control (DBGDRCR)` register).

Under these conditions, the instruction that follows in the ITR might execute instead of being discarded.

## Impact

Indeterminate failures can occur because of the instruction being executed when it should not. In most cases, it is unlikely that the failure will cause any significant issue.

## Workaround

There is a selection of workarounds with increasing complexity and decreasing impact. In each case, the impact is a loss of performance when debugging:

- Do not use stall mode.
- Do not use stall mode when doing Load/Store operations.
- Always check for a sticky abort after issuing a Load/Store operation in stall mode (the cost of this probably means workaround number #2 is a preferred alternative).
- Always check for a sticky abort after issuing a Load/Store operation in stall mode before issuing any further instructions that might corrupt an important target state (such as further Load/Store instructions, instructions that write to "live" registers such as VFP, CP15).

## Category

Category 3

# 751471: DBGPCSR Format Is Incorrect

## Description

In the `Debug Program Counter Sampling (DBGPCSR)` register, the Arm architecture specifies that:

- `DBGPCSR[31:2]` contain the sampled value of bits [31:2] of the PC.
- The sampled value is an instruction address plus an offset that depends on the processor instruction set state.
- `DBGPCSR[1:0]` contain the meaning of PC Sample Value, with the following permitted values:
  - 2'b00 ((`DBGPCSR[31:2]` << 2) – 8) references an Arm state instruction
  - 2'bx1 ((`DBGPCSR[31:1]` << 1) – 4) references a Thumb or ThumbEE state instruction; "x" is a don't care.
  - 2'b10 IMPLEMENTATION DEFINED

This field encodes the processor instruction set state, so that the profiling tool can calculate the true instruction address by subtracting the appropriate offset from the value sampled in bits [31:2] of the register.

In Cortex-A9, the `DBGPCSR` samples the target address of executed branches (but possibly still speculative to data aborts), with the following encodings:

- `DBGPCSR[31:2]` contain the address of the target branch instruction, with no offset.
- `DBGPCSR[1:0]` contains the execution state of the target branch instruction:
  - 2'b00 for an Arm state instruction
  - 2'b01 for a Thumb state instruction
  - 2'b10 for a Jazelle state instruction
  - 2'b11 for a ThumbEE state instruction

## Impact

The implication of this erratum is that the debugger tools must not rely on the architected description for the value of `DBGPCSR[1:0]`, nor remove any offset from `DBGPCSR[31:2]`, to obtain the expected PC value.

Subtracting 4 or 8 from the `DBGPCSR[31:2]` value would lead to an area of code that is unlikely to have been recently executed or might not contain any executable code.

The same might be true for Thumb instructions at half-word boundaries, in which case, PC[1]=1 but `DBGPCSR[1]`=0; or ThumbEE instructions at word boundaries, with PC[1]=0 and `DBGPCSR[1]=1`. In Cortex-A9, because the `DBGPCSR` is always a branch target (in other words, the start of a basic block to the tool), the debugger should be able to spot many of these cases and attribute the sample to the right basic block.

## Workaround

The debugger tools can find the expected PC value and instruction state by reading the `DBGPCSR` register and consider it as described in the "Description" section of this erratum.

## Category

Category 3

# 752519: An Imprecise Abort Might Be Reported Twice on Non-Cacheable Reads

## Description

In the case where two outstanding read memory requests to device or non-cacheable normal memory regions are issued by the Cortex-A9, and the first one receives an imprecise external abort, then the second access might falsely report an imprecise external abort.

This erratum can only happen in systems that can generate imprecise external aborts on device or non-cacheable normal memory regions accesses.

## Impact

When this erratum occurs, a second, spurious imprecise abort might be reported to the core when it should not. In practice, the failure is unlikely to cause any significant issues to the system because imprecise aborts are usually unrecoverable failures. Because the spurious abort can only happen following a first imprecise abort, either the first abort is ignored – and the spurious abort is then ignored too, or it is acknowledged and probably generates a critical failure in the system, such as a processor reset or whole system reboot.

## Workaround

There is no practical software workaround for this erratum.

## Category

Category 3

## 754323: Repeated Store in the Same Cache Line Might Delay the Visibility of the Store

### Description

The Cortex-A9 implements a small counter that ensures the external visibility of all stores in a finite amount of time, causing an eventual drain of the Merging Store Buffer. This counter is present to avoid a situation where written data could potentially remain indefinitely in the Store Buffer.

This Store Buffer has merging capabilities and continues to merge data as long as the write accesses are performed in the same cache line. The issue that causes this erratum is that the draining counter resets each time a new data merge is performed.

In the case when a code sequence loops and continues to write data in this same cache line, then the external visibility of the written data might not be ensured. A livelock situation might consequently occur if any external agent is relying on the visibility of the written data, and where the writing processor cannot be interrupted while doing its writing loop.

This erratum can only happen on Normal Memory regions. The following examples describe scenarios that might trigger this erratum:

- The processor continues incrementing a counter, writing the same word at the same address. The external agent (possibly the other processor) polls on this address, waiting for any update of the counter value to proceed. The Store Buffer continues merging the updated value of the counter in its cache line, so that the external agent never sees any updated value, possibly leading to livelock.
- The processor writes a value in a given word to indicate completion of its task, and then continues writing data in an adjacent word in the same cache line. The external agent continues to poll the first word memory location to check when the processor completes its task. The situation is the same in the first example, because the cache line might remain indefinitely in the merging Store Buffer, creating a possible livelock in the system.

### Impact

This erratum might create performance issues, or worst case, a livelock scenario, if the external agent relies on the automatic visibility of the written data in a finite amount of time.

### Workaround

The recommended workaround for this erratum is to insert a `DMB` operation after the faulty write operation in code sequences that this erratum might affect to ensure the visibility of the written data to any external agent.

### Category

Category 3

## 756421: Sticky Pipeline Advance Bit Cannot be Cleared from Debug APB Accesses

### Description

The Sticky Pipeline Advance bit is bit[25] of the `DBGDSCR` register. This bit enables the debugger to detect whether the processor is idle. This bit is set to 1 every time the processor pipeline retires one instruction. A write to `DBGDRCR[3]` clears this bit. Because of this erratum, the Cortex-A9 does not implement any debug APB access to `DBGDRCR[3]`.

### Impact

The external debugger cannot clear the Sticky Pipeline Advance bit in the `DBGDSCR`. In practice, this makes the Sticky Pipeline Advance bit concept unusable on Cortex-A9 processors.

### Workaround

There is no practical workaround for this erratum. The only possible way to reset the Sticky Pipeline Advance bit is to assert the nDBGRESET input pin on the processor, which obviously has the side effect of resetting all debug resources in the concerned processor, and any other additional CoreSight components to which nDBGRESET is connected.

# 757119: Some Unallocated Memory Hint Instructions Generate an UNDEFINED Exception Instead of Being Treated as a NOP

## Description

The Arm Architecture specifies that Arm opcodes of the form `11110 100x001 xxxx xxxx xxxx xxxx xxxx` are "unallocated memory hint (treat as NOP)" if the core supports the MP extensions, as the Cortex-A9 does.

Because of this erratum, the Cortex-A9 generates an UNDEFINED exception when bits [15:12] of the instruction encoding are different from 4'b1111, instead of treating the instruction as a NOP.

## Impact

Because of this erratum, an unexpected UNDEFINED exception might be generated. In practice, this erratum is unlikely to cause any significant issue because such instruction encodings are not supposed to be generated by any compiler, nor used by any handcrafted program.

## Workaround

The workaround for this erratum is to modify the instruction encoding with bits[15:12]=4'b1111, so that the Cortex-A9 treats the instruction properly as a NOP.

If it is not possible to modify the instruction encoding as described, the UNDEFINED exception handler has to cope with this case and emulate the expected behavior of the instruction, that is, it must do nothing (NOP), before returning to normal program execution.

## Category

Category 3

## 761321: MRC and MCR Are Not Counted in Event 0x68

### Description

Event 0x68 counts the total number of instructions passing through the register rename pipeline stage. The event is also reported externally on PMUEVENT[9:8]. However, with this erratum, the `MRC` and `MCR` instructions are not counted in this event or reported externally on PMUEVENT[9:8]..

### Impact

The implication of this erratum is that the values of event 0x68 and PMUEVENT[9:8] are imprecise, omitting the number of `MCR` and `MRC` instructions. The inaccuracy of the total count depends on the rate of `MRC` and `MCR` instructions in the code.

### Workaround

No workaround is possible to achieve the required functionality of counting precisely how many instructions are passing through the register rename pipeline stage when the code contains some `MRC` or `MCR` instructions.

### Category

Category 3

# 764319: Read Accesses to DBGPRSR and DBGPRCR May Generate an Unexpected UNDEF

## Description

CP14 read accesses to the `Device Power-down and Reset Status (DBGPRSR)` and `Device Powerdown and Reset Control (DBGPRCR)` registers generate an unexpected UNDEFINED exception when the `DBGSWENABLE` bit, bit[31] in the `Coresight Components APB-AP Control/Status Word (CSW)` register at offset 0x00, is 0, even when the CP14 accesses are performed from a privileged mode.

## Impact

Because of this erratum, the `Device Power-down and Reset Status (DBGPRSR)` and the `Device Powerdown and Reset Control (DBGPRCR)` registers are not accessible when `DBGSWENABLE`=0.

This erratum is unlikely to cause any significant issue in Cortex-A9 based systems because these accesses are mainly intended to be used as part of debug over power-down sequences, and the Cortex-A9 does not support this feature.

## Workaround

The workaround for this erratum is to temporarily set the `DBGSWENABLE` bit to 1 so that the `DBGPRSR` and `DBGPRCR` registers can be accessed as expected. There is no other workaround for this erratum.

## Category

Category 3

# 771221: PLD Instructions Might Allocate Data in the Data Cache Regardless of the Cache Enable Bit Value

## Description

PLD instructions prefetch and allocate any data marked as write-back (either write-allocate or non-write-allocate, shared or non-shared), regardless of the processor configuration settings, including the data cache enable bit value.

## Impact

Because of this erratum, unexpected memory cacheability aliasing is created, which might result in various data consistency issues.

In practice, this erratum is unlikely to cause any significant issue. The data cache is likely to be enabled as soon as possible in most systems and not dynamically modified. Therefore, this erratum is likely to impact only boot-up code. This code is usually carefully controlled and does not usually contain any PLD instruction while the data cache is not enabled.

## Workaround

If this erratum impacts a system, a software workaround is available that is to set bit [20] in the undocumented `Control` register, which is placed in CP15 c15 0 c0 1.

This bit must be written with the following read-modify-write code sequence:

```
MRC p15,0,r0,c15,c0,1
ORR r0,r0,#0x00100000
MCR p15,0,r0,c15,c0,1
```

Setting this bit causes all PLD instructions to be treated as NOPs, with the consequence that code sequences that usually use the PLDs, such as the memcpy() routine, might suffer from a visible performance drop. Therefore, if this workaround is applied, Arm strongly recommends restricting its use to periods of time where the data cache is disabled

## Category

Category 3

# 771224: Visibility of Debug Enable Access Rights to Enable/Disable Tracking is Not Ensured by an ISB

## Description

According to the Arm architecture, any change in the `Authentication Status` register should be made visible to the processor after an exception entry or return, or an `ISB`. Although this is correctly achieved for all debug-related features, the `ISB` is not sufficient to make the changes visible to the trace flow. As a consequence, the `WPTTRACEPROHIBITEDn` signal(s) remain stuck to their old value up to the next exception entry or return, or to the next serial branch, even when an `ISB` executes.

A serial branch is one of the following:

- Data processing to PC with the S bit set (for example, `MOVS pc, r14`)
- `LDM pc ^`

## Impact

Because of this erratum, the trace flow might not start or stop as expected by the program.

## Workaround

To work around this erratum, the `ISB` must be replaced by one of the events causing the change to be visible. In particular, replacing the `ISB` by a `MOVS PC` to the next instruction achieves the correct functionality.

## Category

Category 3

## 771225: Speculative Cacheable Reads to Aborting Memory Regions Clear the Internal Exclusive Monitor and May Lead to Livelock

### Description

On the Cortex-A9, when a cacheable read receives an external abort, the aborted line is allocated as invalid in the data cache, and any allocation in the data cache clears the internal exclusive monitor.

Therefore, if a program executes a `LDREX/STREX` loop that continues to receive an abort answer in the middle of the `LDREX/STREX` sequence, then the `LDREX/STREX` sequence never succeeds, leading to a possible processor livelock.

As an example, the following code sequence might exhibit this erratum:

```
loop LDREX
...
DSB
STREX
CMP
BNE loop
...
LDR (into aborting region)
```

The `LDREX/STREX` does not succeed on the first pass of the loop, the `BNE` is mispredicted, and the `LDR` afterward is speculatively executed.

Therefore the processor keeps on executing:

```
LDR to aborting region (this speculative LDR now appears “before” the LDREX & DSB)
LDREX
DSB
STREX
```

The `LDR` misses in L1 and never gets allocated as valid because it is aborting.

The `LDREX` executes and sets the exclusive monitor.

The `DSB` executes. It waits for the `LDR` to complete, which aborts, causing an allocation (as invalid) in the data cache, clearing the exclusive monitor. The `STREX` executes, but the exclusive monitor is now cleared, so the `STREX` fails. The `BNE` might be mispredicted again, therefore the `LDR` is speculatively executed again, and the code loops back on the same failing `LDREX/STREX` sequence.

This erratum happens in systems that might generate external aborts in answer to cacheable memory requests.

### Impact

If the program reaches a stable state where the internal exclusive monitor continues to be cleared in the middle of the `LDREX/STREX` sequence, then the processor might encounter a livelock situation.

In practice, this scenario is very unlikely to happen because several conditions might prevent it:

- Normal `LDREX/STREX` code sequences do not contain any `DSB`, so it is very unlikely that the system would return the abort answer precisely in the middle of the `LDREX/STREX` sequence on each iteration.
- Some external irritants (for example, interrupts) might happen and cause timing changes that might exit the processor from its livelock situation.
- Branch prediction is usually enabled, so the final branch in the loop is usually predicted correctly after a few iterations of the loop, preventing the speculative `LDR` from being issued, so that the next iteration of the `LDREX/STREX` sequence succeeds.

## Workaround

For this erratum, either of the following workarounds fixes the problem:

- Turn on the branch prediction.
- Remove the `DSB` in the middle of the `LDREX/STREX` sequence. If a `DSB` is required, Arm recommends that you place it before the `LDREX/STREX` sequence, and implement the `LDREX/STREX` sequence as recommended by the Arm architecture.

## Category

Category 3

## 775419: PMU Event 0x0A Might Count Twice the LDM PC ^ Instruction with Base Address Register Write-Back

### Description

The `LDM PC ^` instructions with base address register write-back might be counted twice in the PMU event 0x0A, which is counting the number of exception returns. The associated PMUEVENT[11] signal is also affected by this erratum and might be asserted twice by a single `LDM PC ^` with base address register write-back.

### Impact

Because of this erratum, the count of exception returns is imprecise. The error rate depends on the ratio between exception returns of the form `LDM PC ^` with base address register write-back and the total number of exceptions returns

### Workaround

There is no workaround to this erratum.

### Category

Category 3

# 782774: A Spurious Event 0x63 Can be Reported on an LDREX That is preceded by a Write to Strongly Ordered Memory Region

## Description

A write to a strongly ordered memory region, followed by the execution of an `LDREX` instruction can cause the "`STREX`-passed" event to be signaled even if no `STREX` instruction is executed.

As a result, the event 0x63 count might be faulty, reporting too many "`STREX`-passed" events. This erratum also affects the associated PMUEVENT[27] signal. This signal will report the same spurious events.

This erratum requires the following conditions:

1. The processor executes a write instruction to a strongly-ordered memory region.
2. The processor executes an `LDREX` instruction.
3. No `DSB` instruction is executed and there is no exception call or exception return between the write and the `STREX` instructions.

Under these conditions, if the write instruction to the strongly ordered memory region receives its acknowledge (BRESP response on AXI) while the `LDREX` is being executed, this erratum can happen.

## Impact

This erratum leads to a faulty count of event 0x63 or incorrect signaling of PMUEVENT[27].

## Workaround

The workaround for this erratum is to insert a `DMB` or `DSB` instruction between the write to a strongly ordered memory region and the `LDREX` instruction.

## Category

Category 3

# Arm L2 Cache Controller

## 754670: A Continuous Write Flow Can Stall a Read Targeting the Same Memory Area

### Description

In PL310, hazard checking is done on bits [31:5] of the address. When PL310 receives a read with normal memory (cacheable or not) attributes, hazard checking is performed with the active writes of the store buffer. If an address match is detected, the read is stalled until the write completes.

Because of this erratum, a continuous flow of writes can stall a read that targets the same memory area.

This problem occurs when the following conditions are met:.

- PL310 receives a continuous write traffic targeting the same address marked with Normal Memory attributes.
- While treating this flow, PL310 receives a read that targets the same 32-byte memory area.

### Impact

When the conditions above are met, the read might be stalled until the write flow stops. Note that this erratum does not lead to any data corruption. Also, the normal software code is not expected to contain long write sequences like the one causing this erratum to occur.

### Workaround

There is no workaround and there is unlikely to be a need for a workaround for this erratum.

### Category

Category 3

## 765569: Prefetcher Can Cross 4 KB Boundary if Offset is Programmed with Value 23

### Description

When the prefetch feature is enabled (bits[29:28] of the `Auxiliary` or `Prefetch Control` register set HIGH), the prefetch offset bits of the `Prefetch Control` Register (bits[4:0]) configure the advance taken by the prefetcher compared to the current cache line. Refer to the *PL310 Cache Controller Technical Reference Manual* for more information. One requirement for the prefetcher is to not go beyond a 4KB boundary. If the prefetch offset is set to 23 (5'b10111), this requirement is not fulfilled and the prefetcher can cross a 4 KB boundary.

This problem occurs when the following conditions are met:

- One of the Prefetch Enable bits (bits [29:28] of the `Auxiliary` or `Prefetch Control` Register) is set HIGH.
- The prefetch offset bits are programmed with value 23 (5'b10111).

### Impact

When the conditions above are met, the prefetcher can issue linefills beyond a 4 KB boundary compared to the original transaction. System issues can result because those linefills can target a new 4 KB page of memory space, regardless of the page attribute settings in the L1 MMU.

### Workaround

A workaround for this erratum is to program the prefetch offset with any value except 23.

### Category

Category 3

## 729815: The High Priority for SO and Dev Reads Feature Can Cause Quality of Service Issues to Cacheable Read Transactions

### Description

The "High Priority for SO and Dev reads" feature can be enabled by setting bit[10] of the PL310 `Auxiliary Control` Register. When enabled, this feature gives priority to strongly ordered and device reads over cacheable reads in the PL310 AXI master interfaces. When PL310 receives a continuous flow of strongly ordered or device reads, this configuration can prevent cacheable reads that miss in the L2 cache from being issued to the L3 memory system.

This erratum occurs when the following conditions are met:

- Bit[10] "High Priority for SO and Dev reads enable" of the PL310 `Auxiliary Control` Register is set to 1.
- PL310 receives a cacheable read that misses in the L2 cache.
- PL310 receives a continuous flow of strongly ordered or device reads that take all address slots in the master interface.

### Impact

When the conditions above are met, the linefill resulting from the L2 cache miss is not issued until the flow of SO/Device reads stops. Note that each PL310 master interface has four address slots, so that the Quality of Service issue only appears on the cacheable read if the L1 is able to issue at least four outstanding SO/Device reads.

### Workaround

A workaround is only necessary in systems that are able to issue a continuous flow of strongly ordered or device reads. In such a case, the workaround is to disable the "High Priority for SO and Dev reads" feature, which is the default behavior.

### Category

Category 3

# Arm CoreSight PTM

## 720107: Periodic Synchronization Can Be Delayed and Cause Overflow

### Description

The Program Trace Macrocell (PTM) is required to insert synchronization information into the trace stream at intervals in order to allow a partial trace dump to be decompressed. The synchronization period can be controlled either external to the PTM or by a counter that by default, requests synchronization every 1024 bytes of trace.

Synchronization does not need to be inserted precisely at a regular interval, so allowance is made to delay synchronization if the PTM is required to generate other trace packets, or if there is not sufficient space in the FIFO. To guarantee that some synchronization packets are always inserted regardless of the conditions, a 'forced overflow' mechanism shuts off trace if a new synchronization request occurs before the previous request was satisfied. This forced overflow mechanism is minimally intrusive to the trace stream, but ensures that synchronization is inserted after no more than 2x the requested interval.

Due to this erratum, some specific sequences of instructions prevent the PTM from being able to insert any synchronization into the trace stream while that instruction sequence continues. Typically, there is just a short delay which may not be noticed and the synchronization is inserted once the particular pattern of waypoints changes. It is possible that overflows are generated in the trace regardless of the utilization of the FIFO in the PTM. In this scenario, typically only a single byte of trace (up to 5 waypoints) is lost.

The scenario does not correspond to sustained high rates of trace generation which could genuinely cause the FIFO to become full.

Scenarios that cause this erratum are rare, and are limited to code iterating around a loop many times. The loop would contain several branches and be dependent on memory accesses completing.

This erratum can occur under the following conditions:

1. Tracing is enabled.
2. Branch broadcasting is disabled.
3. Cycle accuracy is disabled.
4. The processor executes tight loops of repeating code, lasting longer than the configured synchronization period.

### Impact

This erratum reduces the frequency of periodic synchronization and potentially causes trace overflows where some trace is lost. In the case of an overflow, trace following the overflow can be correctly decompressed. This erratum is more noticeable if the periodic synchronization requests are more frequent.

### Workaround

The most appropriate workaround depends on the use-case for the trace:

- If the trace buffer is large enough, consider increasing the synchronization period by setting the `ETMSYNCFR` to a higher value.
- If no trace must be lost (and periodic synchronization does not have to be guaranteed), set `bit[0]` of `ETMAUXCR` to disable the forced overflow function. Synchronization is inserted at the earliest opportunity, dependent on the executed instruction stream.

## Category

Category 3

## 711668: Configuration Extension Register Has Wrong Value Status

### Description

The Program Trace Macrocell (PTM) implements several read only registers that provide a mechanism for tools using the PTM to determine which features are present in a specific implementation. The `Configuration Code Extension` register (0x7A, address 0x1E8) has an incorrect value of 0x000008EA. The correct value is 0x00C019A2.

### Impact

This erratum has no impact on the generation of trace or the configurations that can be enabled. Tools that read this register in order to determine the capabilities of the PTM detect fewer features than are actually present.

The missing bits include:

- Bit[23] - Return stack implemented.
- Bit[22] - Timestamp Implemented.
- Bit[12] - Reserved, for compatibility with the ETM architecture

Bits[10:3] are incorrect and should indicate 52 external inputs.

### Workaround

Tools using the PTM-A9 can read the `peripheral ID` registers (at offsets 0xFE0 to 0xFEC) to determine the version of the PTM-A9 and the features which are implemented, rather than relying on the `Configuration Code Extension` register.

### Category

Category 3

# Revision History for Arria V SX and ST SoC Errata

| Date | Version | Changes |
|---|---|---|
| July 2015 | 2015.07.14 | Added HPS erratum "SPI Slave Output Signals Cannot Be Isolated When Routed to the HPS Pins" |
| April 2015 | 2015.04.17 | Added Arm erratum 845369 |
| March 2015 | 2015.03.04 | Initial Release |