



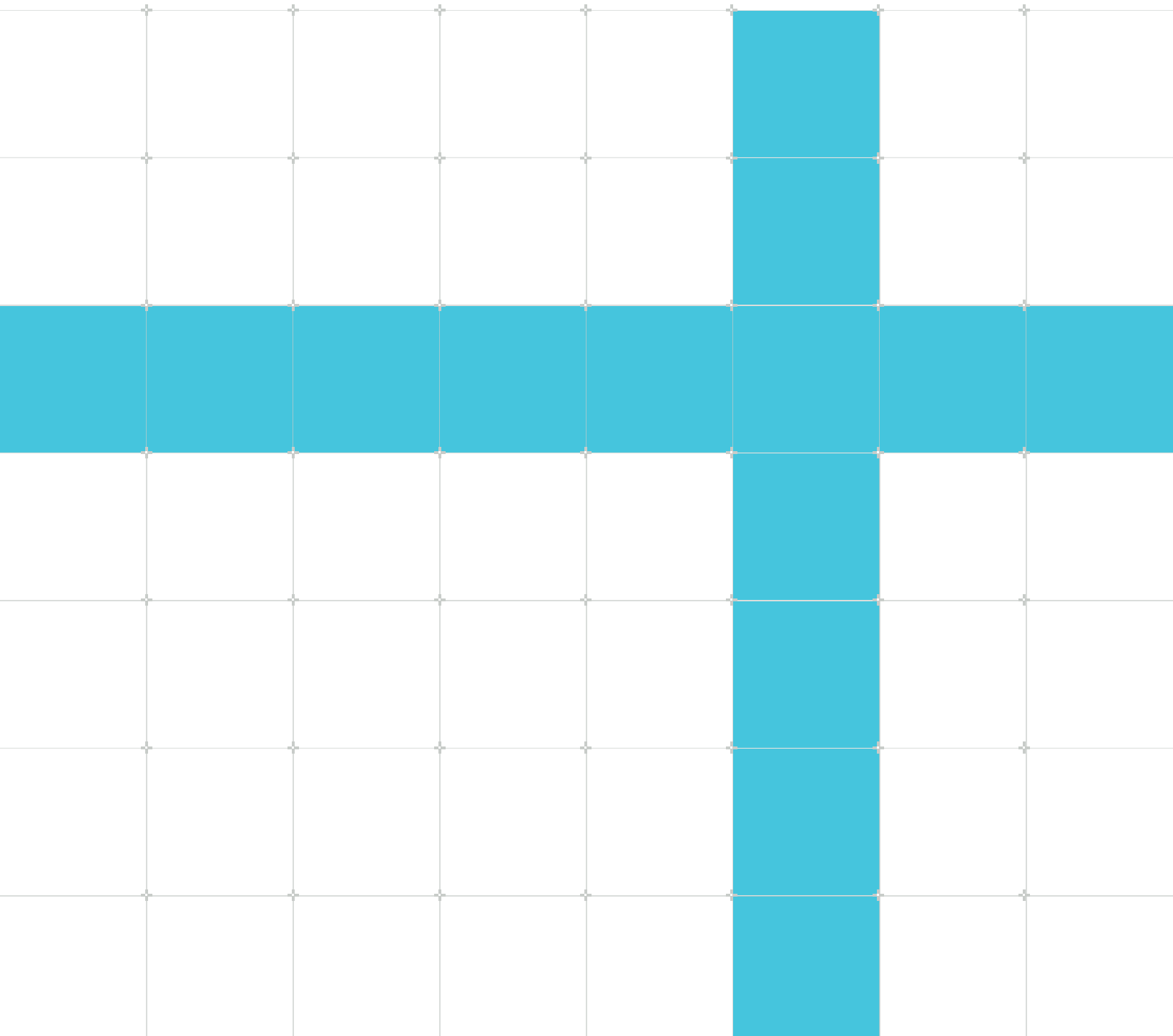
Arm[®] Architecture Reference Manual Armv8, for A-profile architecture

Known issues in Issue G.b

Non-Confidential

Issue 01

Copyright © 2020–2021 Arm Limited (or its affiliates). 102105_G.b_01_en
All rights reserved.



Arm® Architecture Reference Manual Armv8, for A-profile architecture

Known issues in Issue G.b

Copyright © 2020–2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
F.c-04	18 December 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 18 December 2020
G.a-05	30 June 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 18 June 2021
G.b-00	22 July 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 9 July 2021
G.b-01	31 August 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 20 August 2021

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2020–2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

The information in this document is Final, that is for a developed product.

Web address

developer.arm.com

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1 Introduction.....	7
1.1 Conventions.....	7
1.2 Additional reading.....	8
1.3 Feedback.....	8
1.4 Other information.....	9
2 Known issues.....	10
2.1 D14596.....	10
2.2 D16720.....	10
2.3 D17015.....	10
2.4 D17119.....	11
2.5 D17591.....	11
2.6 D17736.....	12
2.7 R17743.....	12
2.8 C17811.....	13
2.9 R17872.....	13
2.10 E17909.....	14
2.11 D17956.....	15
2.12 D18000.....	16
2.13 D18001.....	18
2.14 D18002.....	18
2.15 D18024.....	20
2.16 D18035.....	20
2.17 D18055.....	21
2.18 D18093.....	21
2.19 D18106.....	22
2.20 D18109.....	23
2.21 D18118.....	23
2.22 D18133.....	24
2.23 D18138.....	24
2.24 D18140.....	25
2.25 D18144.....	25

2.26	D18152.....	26
2.27	D18160.....	26
2.28	D18162.....	27
2.29	D18165.....	27
2.30	D18169.....	27
2.31	D18183.....	28
2.32	R18187.....	28
2.33	D18200.....	30
2.34	D18202.....	30
2.35	D18216.....	31
2.36	D18225.....	31
2.37	D18240.....	31
2.38	R18243.....	32
2.39	C18253.....	32
2.40	D18258.....	32
2.41	D18262.....	32
2.42	D18266.....	33
2.43	D18272.....	34
2.44	D18282.....	34
2.45	D18284.....	35
2.46	D18288.....	35
2.47	D18291.....	36
2.48	D18294.....	36
2.49	D18299.....	37
2.50	D18300.....	37
2.51	C18301.....	37
2.52	R18319.....	37
2.53	D18330.....	38
2.54	D18347.....	38
2.55	D18352.....	38
2.56	D18354.....	39
2.57	D18366.....	39
2.58	D18371.....	39

1 Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.




Glossary




The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
<i>italic</i>	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<i>monospace italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
monospace bold	Denotes language keywords when used outside example code.
monospace <u>underline</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the <i>Arm Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .
 Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
 Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
 Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.

Convention	Use
 Note	This represents an important piece of information that needs your attention.
 Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
 Remember	This is a reminder of something important that relates to the information you are reading.

1.2 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Table 1-2: Arm publications

Document Name	Document ID	Licensee only
Arm® Architecture Reference Manual Armv8, for A-profile architecture, Issue G.b	DDI 0487G.b	No

1.3 Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title Arm® Architecture Reference Manual Armv8, for A-profile architecture Known issues in Issue G.b.
- The number 102105_G.b_01_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.



Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

1.4 Other information

See the Arm website for other relevant information.

- [Arm® Developer](#).
- [Arm® Documentation](#).
- [Technical Support](#)
- [Arm® Glossary](#).

2 Known issues

This document records known issues in the Arm Architecture Reference Manual, Armv8, for A-profile architecture (DD10487), Issue G.b.

Key

- C = Clarification.
- D = Defect.
- R = Relaxation.
- E = Enhancement.

2.1 D14596

In section I5.7.15 (CNTSR, Counter Status Register), the bit assignment is changed to the following:

- [31:18] **RES0**
- [17:8] FCACK

2.2 D16720

In section D5.7.2 (Enhanced support for nested virtualization), the following table entry is added to Table D5-50 'Redirection of accesses to special-purpose registers at EL2':

Special register access instruction	Named EL2 register	Actual register accessed
op1 = 4, CRm=6, op2=0	TFSR_EL2	TFSR_EL1

2.3 D17015

Details of traps will be added through the use of new LDC and STC accessibility pseudocode in sections G8.3.17 (DBGDTRRXint) and G8.3.18 (DBGDTRTXint). This accessibility pseudocode is the same as for the equivalent MRC and MCR instructions, except that:

- The reported exception syndrome value, if applicable, is 0×06 .
- For LDC instructions the accessibility pseudocode loads the value to be written to the System register from 'MemA[address, 4]', where 'address' is the virtual address calculated by the LDC instruction.

2.4 D17119

In sections F3.1.10 (Advanced SIMD shifts and immediate generation), sub-section 'Advanced SIMD two registers and shift amount' and F4.1.22 (Advanced SIMD shifts and immediate generation), sub-section 'Advanced SIMD two registers and shift amount', the following constraints are added to VMOVL:

- 'L' must be '0'.
- 'imm3H' cannot be '000'.

2.5 D17591

In section G8.2.123 (RMR, Reset Management Register), the MCR accessibility code that currently reads:

```
if PSTATE.EL IN {EL1, EL3} && IsHighestEL(PSTATE.EL) then
    RMR = R[t];
else
    UNDEFINED;
```

is modified to:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        RMR = R[t];
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    UNDEFINED;
elseif PSTATE.EL == EL3 then
    if CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        RMR = R[t];
```

In section G8.2.114 (MVBAR, Monitor Vector Base Address Register), the MCR accessibility code that currently reads:

```
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        MVBAR = R[t];
```

is modified to:

```
elseif PSTATE.EL == EL3 then
    if CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        MVBAR = R[t];
```

Similar changes are made in the following sections to exclude checks for the SCR.NS bit:

- G8.2.119 (NSACR, Non-Secure Access Control Register).
- G8.3.35 (SDER, Secure Debug Enable Register).
- G8.3.34 (SDCR, Secure Debug Control Register).

2.6 D17736

In section J1.3.1 (shared/debug), the description for ExternalNoninvasiveDebugEnabled() that reads:

```
// ExternalNoninvasiveDebugEnabled()
// =====
// This function returns TRUE if the FEAT_Debugv8p4 is implemented, otherwise this
// function is IMPLEMENTATION DEFINED.
// In the recommended interface, ExternalNoninvasiveDebugEnabled returns the state
// of the (DBGEN
// OR NIDEN) signal.
```

is updated to read:

```
// ExternalNoninvasiveDebugEnabled()
// =====
// This function returns TRUE if the FEAT_Debugv8p4 is implemented.
// Otherwise, this function is IMPLEMENTATION DEFINED, and, in the
// recommended interface, ExternalNoninvasiveDebugEnabled returns
// the state of the (DBGEN OR NIDEN) signal.
```

2.7 R17743

In section D9.7.3 (Memory access types and coherency), the following text is removed:

The SPU acts as a separate observer in the system and is subject to the rules regarding coherency.

2.8 C17811

In section I5.8.32 (ERR<n>STATUS, Error Record Primary Status Register, n = 0 - 65534), under the heading 'Accessing the ERR<n>STATUS', the text that reads:

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:

- Read ERR<n>STATUS and determine which fields need to be cleared to zero.
- Write ones to all the W1C fields that are nonzero in the read value.
- Write zero to all the W1C fields that are zero in the read value.
- Write zero to all the RW fields.

is clarified to read:

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:

- Read ERR<n>STATUS and determine which fields need to be cleared to zero.
- In a single write to ERR<n>STATUS:
 - Write ones to all the W1C fields that are nonzero in the read value.
 - Write zero to all the W1C fields that are zero in the read value.
 - Write zero to all the RW fields.
- Read back ERR<n>STATUS after the write to confirm no new fault has been recorded.

2.9 R17872

In section B2.3.3 (Ordering relations), the text in the definition of Atomic-ordered-before that currently reads:

- RW1 is a memory write effect W1 generated by an atomic instruction or a successful Store-Exclusive instruction and RW2 is a memory read effect R2 generated by an instruction with Acquire or AcquirePC semantics such that R2 is a Local read successor of W1.

is relaxed to read:

- RW1 is a read memory effect R1 generated by an atomic instruction and RW2 is a read memory effect R2 generated by an instruction with Acquire or AcquirePC semantics such that R2 is a Local read successor of the write memory effect W3 generated by the same generated by an atomic instruction or a successful Store-Exclusive instruction and RW2 is a memory read effect R2 generated by the same atomic instruction as R1.

2.10 E17909

Arm® Architecture Reference Manual Armv8, for A-profile architecture, Issue G.b introduces FEAT_WFxF2, which adds support for reporting the register number for trapped WFxF instructions in ESR_ELx. E17909 removes FEAT_WFxF2, and adds the functionality that was introduced by FEAT_WFxF2 to FEAT_WFxF.

In section A2.10.1 (Architectural features added by Armv8.7), the title that reads 'FEAT_WFxF and FEAT_WFxF2, WFE and WFI instructions with timeout' is updated to read 'FEAT_WFxF, WFE and WFI instructions with timeout', and the text under the title is updated to read:

FEAT_WFxF introduces WFET and WFIT. These instructions support the generation of a local timeout event to act as a wake-up event for the PE when the virtual count in CNTVCT_ELO equals or exceeds the value supplied by the instruction for the first time. The register number that holds the timeout value for trapped WFET and WFIT instructions is reported in ESR_ELx.

These instructions are added to the A64 instruction set only.

FEAT_WFxF is mandatory in Armv8.7 implementations.

The ID_AA64ISAR2_EL1.WFxF field identifies the presence of FEAT_WFxF.

In section D13.2.64 (ID_AA64ISAR2_EL1), the 'WFxF, bits [3:0]' field is updated to read:

Indicates support for the WFET and WFIT instructions in AArch64 state. Defined values are:

0b0000 WFET and WFIT are not supported.

0b0010 WFET and WFIT are supported, and the register number is reported in the ESR_ELx on exceptions.

All other values are reserved.

FEAT_WFxF implements the functionality identified by the value 0b0010.

From Armv8.7, the only permitted value is 0b0010.

Correspondingly, in sections D13.2.37 (ESR_EL1), D13.2.38 (ESR_EL2), and D13.2.39 (ESR_EL3), in the ISS encoding an exception from a WF* instruction, the following fields are added:

RN, bits [9:5]

When FEAT_WFxF is implemented:

Register Number. Indicates the Register Number supplied for a WFET or WFIT instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, **RES0**.

RV, bit [2]

When FEAT_WFxT is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped WFET or WFIT instruction.

0b0 Register field invalid.

0b1 Register field valid.

If TI[1] == 0, then this field is **RES0**.

This field is set to 1 on a trap on WFET or WFIT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, **RES0**.

2.11 D17956

In section K11.3.3 (Ticket Locks), the text that currently reads:

Releasing the ticket lock simply involves incrementing the current ticket number, that is still assumed to be in R3, and doing a Store-Release:

is corrected to read:

Releasing the ticket lock simply involves incrementing the current ticket number, which is assumed in this example to be in R6, and doing a Store Release:

Within the same section, the AArch64 code that reads:

```
ADD W5, W5, #0x10000 ; increment the next number
STXR W6, W5, [X1] ; and update the value
```

is corrected to read:

```
ADD W3, W5, #0x10000 ; increment the next number
```

```
STXR W6, W3, [X1] ; and update the value
```

Similarly, the AArch32 code within the same section that reads:

```
ADD R5, R5, #0x10000 ; increment the next number  
STREX R6, R5, [R1] ; and update the value
```

is corrected to read:

```
ADD R3, R5, #0x10000 ; increment the next number  
STREX R6, R3, [R1] ; and update the value
```

The AArch32 code within the same section that reads:

```
BEQ block_start
```

is enhanced to read:

```
MOV R6, R5  
BEQ block_start
```

The equivalent changes for this enhancement are made in section K11.3.4 (Use of Wait For Event (WFE) and Send Event (SEV) with locks), in the AArch32 code within the subsection 'Ticket lock'.

2.12 D18000

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', the following events are redefined as counting the accesses made by the hardware prefetcher, rather than the refills:

- 0x8145, L1I_CACHE_HWPRF.
- 0x814D, L2I_CACHE_HWPRF.
- 0x8154, L1D_CACHE_HWPRF.
- 0x8155, L2D_CACHE_HWPRF.
- 0x8156, L3D_CACHE_HWPRF.

For example, '0x8154, L1D_CACHE_HWPRF, Level 1 data cache hardware prefetch' is defined as:

The counter counts each access counted by L1D_CACHE that is not counted by L1D_CACHE_RW or L1D_CACHE_PRFM.

Correspondingly, the events L<n>I_CACHE_REFILL_HWPRF and L<n>D_CACHE_REFILL_HWPRF are defined. For example, L1D_CACHE_REFILL_HWPRF is defined as:

The counter counts each hardware prefetch access counted by L1D_CACHE_HWPRF that causes a refill of the Level 1 data or unified cache from outside the Level 1 data or unified cache of this PE.

Equivalent event definitions are added for the other L<n>D_CACHE and L<n>I_CACHE cache levels.

Within the same subsection, the definitions of the L<n>D_CACHE, L<n>I_CACHE, L<n>D_CACHE_REFILL, and L<n>I_CACHE_REFILL events are redefined to include the hardware prefetcher. For example, the text in '0x0004, L1D_CACHE, Level 1 data cache access' that reads:

When the L1D_CACHE_PRFM and L1D_CACHE_RW events are implemented, accesses to the Level 1 data or unified cache due to a preload or prefetch instruction are counted. Otherwise, it is **IMPLEMENTATION DEFINED** whether accesses to the Level 1 data or unified cache due to a preload or prefetch instruction are counted.

is changed to read:

When the L1D_CACHE_RW event is implemented:

- If the L1D_CACHE_PRFM event is implemented, accesses to the Level 1 data or unified cache due to a preload or prefetch instruction are counted. Otherwise, these are not counted.
- If the L1D_CACHE_HWPRF event is implemented, accesses to the Level 1 data or unified cache due to a hardware prefetcher are counted. Otherwise, these are not counted.

When the L1D_CACHE_RW event is not implemented, it is **IMPLEMENTATION DEFINED** whether accesses to the Level 1 data or unified cache due to a preload or prefetch instructions or due to a hardware prefetcher are counted.

Equivalent changes are made to the other L<n>D_CACHE and L<n>I_CACHE cache access events.

Also within the same subsection, the following text in '0x8140, L1D_CACHE_RW, Level 1 data or unified cache demand access':

The counter counts each access counted by L1D_CACHE that is not counted by L1D_CACHE_PRFM.

is changed to read:

The counter counts each access counted by L1D_CACHE that is due to a demand read or demand write access.

Equivalent changes are made to the other L<n>D_CACHE_RW and L<n>I_CACHE_RD demand cache access events.

2.13 D18001

In section D2.10.6 (Watchpoint behavior on other instructions), the Note that reads:

Note: Despite its mnemonic, the DC ZVA, Data Cache Zero by VA instruction is not a data cache maintenance instruction.

is clarified to read:

Note: Despite their mnemonics, the DC GVA, DC GZVA, and DC ZVA instructions are not data cache maintenance instructions.

The equivalent Notes in sections D2.10.5 (Determining the memory location that caused a Watchpoint exception) and D4.4.8 (A64 Cache maintenance instructions), subsection 'The data cache maintenance instruction (DC)', are similarly clarified.

The following changes are made in section D4.4.8 (A64 Cache maintenance instructions), subsection 'Ordering and completion of data and instruction cache instructions':

- The references to 'data cache instructions, other than DC ZVA' are clarified to 'data cache instructions, other than DC ZVA, GC GVA, and DC GZVA'.
- In the list for all data cache maintenance instructions that do not specify an address, the reference 'other than Data Cache Zero' is clarified to 'other than DC ZVA, GC GVA, and DC GZVA'.

In section D5.4.2 (About PSTATE.PAN), the following item in the list of instructions that the PAN bit does not affect:

Data Cache instructions other than DC ZVA.

is clarified to read:

Data cache instructions other than DC GVA, DC GZVA, and DC ZVA.

2.14 D18002

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', in the definitions of the STALL_FRONTEND_L<n>I and STALL_FRONTEND_MEM events, the text that reads 'demand miss' is clarified to read 'demand instruction miss'. Similarly, in the STALL_BACKEND_L<n>D and STALL_BACKEND_MEM events, the text that reads 'demand miss' is clarified to read 'demand data miss'. For example, in STALL_BACKEND_L2D the text that reads:

The counter counts each cycle counted by STALL_BACKEND_MEMBOUND when there is a demand miss in the second level data or unified cache.

is clarified to read:

The counter counts each cycle counted by `STALL_BACKEND_MEMBOUND` when there is a demand data miss in the second level data or unified cache.

In section D8.3.1 (Architected event counters), the text in the definition of the '0x4005, `STALL_BACKEND_MEM`, Memory stall cycles' AMU event, that reads:

The counter counts cycles in which the PE is unable to dispatch instructions from the frontend to the backend of the PE due to a backend stall caused by a miss in the last level of cache within the PE clock domain or, if Armv8.7 is implemented, non-cacheable access in progress.

If Armv8.7 is not implemented, it is **IMPLEMENTATION DEFINED** whether the counter counts backend stall cycles when a non-cacheable access is in progress.

is changed to read:

The counter counts cycles in which the PE is unable to dispatch instructions from the frontend to the backend of the PE due to a backend stall caused by a demand data miss in the last level of data or unified cache within the PE clock domain or, if Armv8.7 is implemented, a non-cacheable data access in progress.

If Armv8.7 is not implemented, it is **IMPLEMENTATION DEFINED** whether the counter counts backend stall cycles when a non-cacheable data access is in progress.

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', the `STALL_FRONTEND_L<n>I` and `STALL_BACKEND_L<n>D` events are modified such that they are only counted if the corresponding `STALL_FRONTEND_L<n+1>I` or `STALL_BACKEND_L<n+1>D` event is not counted. Similarly, these event definitions and the MEM event definitions are also updated, such that: if an `LnI` or `LnD` event is an alias for MEM, then the `LnI` or `LnD` event is not implemented, and the counter does not count. For example, '0x8165, `STALL_BACKEND_L1D`, Backend stall cycles, level 1 data cache' is updated to read:

The counter counts each cycle counted by `STALL_BACKEND_MEMBOUND` when there is a demand data miss in the first level of data or unified cache.

This counter does not count the cycle if any of the following are true:

- The `STALL_BACKEND_L2D` event is implemented and there is a demand data miss in the second level of data or unified cache, meaning the `STALL_BACKEND_L2D` event counts the cycle.
- There is a demand data miss in the last level of data or unified cache within the PE clock domain, meaning the `STALL_BACKEND_MEM` event counts the cycle.

This event is only implemented if the first level of data or unified cache is implemented within the PE clock domain and is not the last level of data or unified cache within the PE clock domain.

2.15 D18024

In section J1.3.3 (shared/functions), in the routine ELStateUsingAArch32K() the code which reads:

```

if !HaveAArch32EL(el) then
    return (TRUE, FALSE); // Exception level is using AArch64
elseif secure && el == EL2 then
    return (TRUE, FALSE); // Secure EL2 is using AArch64
elseif HighestELUsingAArch32() then
    return (TRUE, TRUE); // Highest Exception level, and therefore all levels
are using AArch32
elseif el == HighestEL() then
    return (TRUE, FALSE); // This is highest Exception level, so is using
AArch64

```

is updated to read:

```

if !HaveAArch32EL(el) then
    return (TRUE, FALSE); // Exception level is using AArch64
elseif secure && el == EL2 then
    return (TRUE, FALSE); // Secure EL2 is using AArch64
elseif HighestELUsingAArch32() then
    return (TRUE, TRUE); // Highest Exception level, and therefore all levels
are using AArch32

```

2.16 D18035

In section D13.4.9 (PMEVTYPER<n>_ELO, Performance Monitors Event Type Registers, n = 0 - 30), the text in the evtCount field that reads:

If evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- If 16-bit evtCount is implemented, for the range 0x4000 to 0x403F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For **IMPLEMENTATION DEFINED** events, it is **UNPREDICTABLE** what event, if any, is counted, and the value returned by a direct or external read of the evtCount field is **UNKNOWN**.

is corrected to read:

If PMEVTYPER<n>_ELO.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_ELO.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range 0x4000 to 0x403F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_ELO.evtCount field is the value written to the field.

- For other values, it is **UNPREDICTABLE** what event, if any, is counted, and the value returned by a direct or external read of the `PMEVTYPER<n>_ELO.evtCount` field is **UNKNOWN**.

Additionally, the text in the same field that reads:

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include an event from a set of common **IMPLEMENTATION DEFINED** events, then no event is counted and the value read back on `evtCount` is the value written.

is replaced by the following text:

Arm recommends that for all values that represent reserved or unsupported events, no events are counted and the value returned by a direct or external read of the `PMEVTYPER<n>_ELO.evtCount` field is the value written to the field.

The equivalent changes are made in sections G8.4.11 (`PMEVTYPER<n>`, Performance Monitors Event Type Registers, $n = 0 - 30$), and I5.3.24 (`PMEVTYPER<n>_ELO`, Performance Monitors Event Type Registers, $n = 0 - 30$).

2.17 D18055

In section D1.12.4 (Synchronous exception prioritization for exceptions taken to AArch64 state), the following bullet point is added to the list of priority 13 cases:

- When `FEAT_FGT` and `FEAT_PMUv3` are implemented, executing an MRS or MSR instruction in AArch64 state, or an MRC or MCR instruction in AArch32 state, that accesses a register associated with an unimplemented event counter.

A similar change is made in the equivalent section in AArch32, G1.12.2 (Exception prioritization for exceptions taken to AArch32 state), subsection 'Synchronous exception prioritization for exceptions taken to AArch32 state'.

2.18 D18093

In section J1.1.5 (aarch64/translation), in the pseudocode function `AArch64.S1ApplyOutputPerms()`, the lines that read:

```
if regime == Regime_EL10 && EL2Enabled() && walkparams.nvl == '1' then
    permissions.ap<2:1> = descriptor<7>:'0';
    permissions.pxn    = descriptor<54>;

    return permissions;

if HasUnprivileged(regime) then
```

are corrected to read:

```
if regime == Regime_EL10 && EL2Enabled() && walkparams.nvl == '1' then
```

```
permissions.ap<2:1> = descriptor<7>:'0';  
permissions.pxn     = descriptor<54>;  
  
elseif HasUnprivileged(regime) then
```

2.19 D18106

In section H7.1.3 (Permitted behavior that might make the PC Sample-based profiling registers **UNKNOWN**), the text that reads:

If no instruction has been retired since the PE left Debug state, Reset state, or a state where PC Sample-based profiling is prohibited, the sampled value is **UNKNOWN**. If an instruction has been retired but this is the first time the PMPCSR or EDPCSR is read since the PE left Reset state, the sampled value is permitted but not required to return the value `0xFFFFFFFF`.

is relaxed to read:

If no branch instruction has been retired since the PE left Debug state, reset state, or a state where PC Sample-based profiling is prohibited, the sampled value is **UNKNOWN**. If a branch instruction has been retired but this is the first time the PMPCSR or EDPCSR is read since the PE left reset state, the sampled value is permitted but not required to return the value `0xFFFFFFFF`.

Similarly, in section H9.2.32 (EDPCSR, External Debug Program Counter Sample Register), the text in the Bits [31:0] description that reads:

If an instruction has retired since the PE left Reset state, then the first read of EDPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`. EDPCSRlo reads as an **UNKNOWN** value when any of the following are true:

- The PE is in Reset state.
- No instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No instruction has retired since the last read of EDPCSR[31:0].

is relaxed to read:

If a branch instruction has retired since the PE left reset state, then the first read of EDPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`. EDPCSRlo reads as an **UNKNOWN** value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of EDPCSR[31:0].

The equivalent changes are made in section I5.3.33 (PMPCSR, Program Counter Sample Register).

2.20 D18109

In section D9.6.4 (Additional information for each profiled conditional instruction), the text that reads:

For an Architecturally executed sampled conditional select, conditional move, or conditional increment operation finishes execution, the profiling operation records:

- That the sampled operation was conditional.
- Whether the condition passed or failed.

is changed to read:

For an Architecturally executed sampled conditional select or conditional compare operation that finishes execution, the profiling operation records:

- That the sampled operation was conditional.
- Whether the condition passed or failed. Conditional select operations include conditional select, conditional select increment, conditional select negate, and conditional select invert operations, including general-purpose, FP&SIMD, and SVE operations, as well as aliases such as conditional set and conditional increment. It does not include SVE operations where the conditionality of the operation is controlled by a predicate.

In section D10.2.6 (Events packet), subsection 'Events packet payload', in the E[6] field, the following Note is deleted:

This Event includes branches, selects, CCMP (register), and CCMP (immediate).

In section D10.2.7 (Operation Type packet), subsection 'Operation Type packet payload (Other)', in the COND field, the text that reads:

1 Conditional operation or select.

is corrected to read:

1 Conditional select or conditional compare operation.

2.21 D18118

In section J1.2.2 (aarch32/exceptions), in the function AArch32.CheckForWFXTrap(), the code that reads:

```
boolean is_wfe = wfxtype IN {WfxType_WFE, WfxType_WFET};
```

is updated to read:

```
boolean is_wfe = wfxtype == WfxType_WFE;
```

2.22 D18133

In section D13.8.15 (CNTKCTL_EL1, Counter-timer Kernel Control register) in the definition of EVNTEN, bit [2], the text that reads:

When FEAT_VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from the counter register CNTVCT_ELO.

is clarified to read:

When FEAT_VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from the counter register CNTVCT_ELO as seen from EL1.

All other references to CNTVCT_ELO as part of the event stream in this section are clarified in a similar way.

Similarly, in section D13.8.2 (CNTHCTL_EL2, Counter-timer Hypervisor Control register) for the EVNTEN, bit [2] the text that reads:

Enables the generation of an event stream from the counter register CNTPCT_ELO.

is clarified to read:

Enables the generation of an event stream from the counter register CNTPCT_ELO as seen from EL2.

All other references to CNTPCT_ELO as part of the event stream in this section are clarified in a similar way.

2.23 D18138

In section C7.2.146 (FRECPX), the text that reads:

This instruction finds an approximate reciprocal exponent for each vector element in the source SIMD&FP register, places the result in a vector, and writes the vector to the destination SIMD&FP register.

is replaced by the following text:

This instruction finds an approximate reciprocal exponent for the source SIMD&FP register and writes the result to the destination SIMD&FP register.

2.24 D18140

In sections B2.3.8 (Ordering of instruction fetches) and E2.3.8 (Ordering of instruction fetches), the text that reads:

For two memory locations A and B, if A has been written to and been made coherent with the instruction fetches of the shareability domain, before an update to B by an observer in the same shareability domain, then the instruction stream of each observer in the shareability domain will not see the updated value of B without also seeing the updated value of A.

is corrected to read:

For two memory locations A and B:

If A has been written to with an updated value and been made coherent with the instruction fetches of the shareability domain, before B has been written to with an updated value by an observer in the same shareability domain, then where, for an observer in the shareability domain, an instruction read from B appears in program order before an instruction fetched from A, if the instruction read from B contains the updated value of B then the instruction read from A appearing later in program order will contain the updated value of A.

2.25 D18144

In sections J1.1.2 (aarch64/exceptions), AArch64.AArch32SystemAccessTrapSyndrome() which currently reads:

```

bits(20) iss = Zeros();

if exception.exceptype IN {Exception_FPIDTrap, Exception_CP14RTTrap, Excep\
tion_CP15RTTrap} then
    // Trapped MRC/MCR, VMRS on FPSID
    if exception.exceptype != Exception_FPIDTrap then // When trap is not for
VMRS
        ...
            if instr<19:16> == '1111' then // Rn==15, LDC(Literal addressing)/STC
                iss<9:5> = bits(5) UNKNOWN;
                iss<3> = '1';
            elsif exception.exceptype == Exception_Uncategorized then
                // Trapped for unknown reason
                iss<9:5> = LookUpRIndex(UInt(instr<19:16>), PSTATE.M)<4:0>; // Rn
                iss<3> = '0';

        iss<0> = instr<20>; // Direction
        ...

```

is updated to read:

```

bits(20) iss = Zeros();

if exception.exceptype == Exception_Uncategorized then
    return exception;
elseif exception.exceptype IN {Exception_FPIDTrap, Exception_CP14RTTrap, Excep\
tion_CP15RTTrap} then

```

```

// Trapped MRC/MCR, VMRS on FPSID
if exception.exceptype != Exception_FPIDTrap then // When trap is not for
VMRS
    ...
    if instr<19:16> == '1111' then // Rn==15, LDC(Literal addressing)/STC
        iss<9:5> = bits(5) UNKNOWN;
        iss<3> = '1';
    iss<0> = instr<20>; // Direction
    ...

```

In section J1.2.2 (aarch32/exceptions), equivalent changes are made to AArch32.SystemAccessTrapSyndrome().

2.26 D18152

In section D7.5.2 (Freezing event counters), in the bullet list that is introduced by the paragraph 'When FEAT_PMUv3p7 is implemented, the PMU can be configured to freeze event counters...', the bullet point that reads:

- If EL2 is implemented, MDCR_EL2.HPMN is less than PMCR_EL0.N and n is in the range [MDCR_EL2.HPMN .. (PMCR_EL0.N-1)], when PMOVSLR_EL0[(PMCR_EL0.N-1):MDCR_EL2.HPMN] is non-zero, indicating an unsigned overflow in one of the event counters in the range, event counter n does not count when PMCR_EL0.FZO is 1.

is corrected to read:

- If EL2 is implemented, MDCR_EL2.HPMN is less than PMCR_EL0.N and n is in the range [MDCR_EL2.HPMN .. (PMCR_EL0.N-1)], when PMOVSLR_EL0[(PMCR_EL0.N-1):MDCR_EL2.HPMN] is non-zero, indicating an unsigned overflow in one of the event counters in the range, event counter n does not count when MDCR_EL2.HPMFZO is 1.

2.27 D18160

In section J1.1.1 (aarch64/debug), the code in AArch64.CountEvents() that reads:

```

// PMCR_EL0.DP disables the cycle counter when event counting is prohibited
if enabled && prohibited && n == 31 then
    enabled = PMCR_EL0.DP == '0';

```

is corrected to read:

```

// PMCR_EL0.DP disables the cycle counter when event counting is prohibited
if prohibited && n == 31 then
    enabled = enabled && PMCR_EL0.DP == '0';
    prohibited = FALSE; // Otherwise whether event counting is prohibited does
not affect the cycle counter

```

A similar correction is made in section J1.2.1 (aarch32/debug) to `AArch32.CountEvents()`.

2.28 D18162

In section D10.2.6 (Events packet), subsection 'Events packet payload', the text in 'E[17], byte 2 bit [17], when `SZ == 0b10`, or `SZ == 0b11`' that reads:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented this Event is required to be implemented consistently with `SVE_PRED_EMPTY_SPEC` and `SVE_PRED_PARTIAL_SPEC` in the Arm Architecture Reference Manual Supplement, the Scalable Vector Extension, for v8-A.

is corrected to read:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented, this Event is required to be implemented consistently with `<xref: SVE_PRED_NOT_FULL_SPEC>`.

Similarly, the text in 'E[18], byte 2 bit [18], when `SZ == 0b10`, or `SZ == 0b11`' that reads:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented this Event is required to be implemented consistently with `SVE_PRED_EMPTY_SPEC` in the Arm Architecture Reference Manual Supplement, the Scalable Vector Extension, for v8-A.

is clarified to read:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented, this Event is required to be implemented consistently with `<xref: SVE_PRED_EMPTY_SPEC>`.

2.29 D18165

In section D5.9.1 (Use of ASIDs and VMIDs to reduce TLB maintenance requirements), subsection 'VMID size', the line that reads:

When the value of `VTCR_EL2.VS` is 0, `VMID[63:56]`:

is corrected to read:

When the value of `VTCR_EL2.VS` is 0, `VTTBR_EL2[63:56]`:

2.30 D18169

In section J1.3.3 (shared/functions), in the function `GenMPAMcurEL()`, the following line of code:

```
if HaveEMPAMExt() && pspace == PIdSpace_Secure then
```

is changed to:

```
if HaveEMPAMExt() && security == SS_Secure then
```

Within the same section, in the function PARTIDspaceFromSS(), the following code is removed:

```
if HaveEMPAMExt() && MPAM3_EL3.FORCE_NS == '1' then  
    return PIdSpace_NonSecure;  
else  
    return PIdSpace_Secure;
```

and is replaced with:

```
return PIdSpace_Secure
```

2.31 D18183

In section C6.2.79 (DGH), the description that reads:

DGH is a hint instruction. A DGH instruction is not expected to be performance optimal to merge memory accesses with Normal Non-cacheable or Device-GRE attributes appearing in program order before the hint instruction with any memory accesses appearing after the hint instruction into a single memory transaction on an interconnect.

is updated to read:

Data Gathering Hint is a hint instruction that indicates that it is not expected to be performance optimal to merge memory accesses with Normal Non-cacheable or Device-GRE attributes appearing in program order before the hint instruction with any memory accesses appearing after the hint instruction into a single memory transaction on an interconnect.

2.32 R18187

In section I5.3.14 (PMCID2SR, CONTEXTIDR_EL2 Sample Register), in the description of CONTEXTIDR_EL2, bits [31:0], the text that reads:

When the most recent PMPCSR sample was generated:

- If EL2 is using AArch64, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- If EL2 is using AArch32, then this field is set to an **UNKNOWN** value.

is changed to read:

When the most recent PMPCSR sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an **UNKNOWN** value.

Similarly, the text in section H9.2.43 (EDVIDSR, External Debug Virtual Context Sample Register), in the description of CONTEXTIDR_EL2, bits [31:0], that reads:

When the most recent EDPCSR sample was generated:

- If EL2 was using AArch64 and the PE was executing in Non-secure state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- If EL2 was using AArch32 or the PE was executing in Secure state, then this field is set to an **UNKNOWN** value.

is changed to:

When the most recent EDPCSR sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an **UNKNOWN** value.

In section J1.3.1 (shared/debug), within the pseudocode function CreatePCSample(), the code that reads:

```
pc_sample.has_el2 = EL2Enabled();
if EL2Enabled() then
    ...
```

is changed to read:

```
pc_sample.has_el2 = PSTATE.EL != EL3 && EL2Enabled();
if pc_sample.has_el2 then
    ...
```

Also in section J1.3.1 (shared/debug), within the pseudocode function EDPCSRlo[], the code that reads:

```
if (HaveVirtHostExt() || HaveV82Debug()) && EDSCR.SC2 == '1' then
    EDVIDSR = (if HaveEL(EL2) && pc_sample.ns == '1' then pc_sample.contextidr_el2
               else bits(32) UNKNOWN);
else
    if HaveEL(EL2) && pc_sample.ns == '1' && pc_sample.el IN {EL1,EL0} then
        EDVIDSR.VMID = pc_sample.vmid;
    else
        EDVIDSR.VMID = Zeros();
```

is simplified to read:

```
if (HaveVirtHostExt() || HaveV82Debug()) && EDSCR.SC2 == '1' then
    EDVIDSR = (if pc_sample.has_el2 then pc_sample.contextidr_el2 else bits(32) UN\
KNOWN);
```

```
else
    EDVIDSR.VMID = (if pc_sample.has_el2 && pc_sample.el IN {EL1,EL0} then pc_sam\
ple.vmid else Zeros());
```

2.33 D18200

In section D7.10.1 (Definitions), subsection 'Definition of terms', a new definition 'Event in progress' is added:

Some events count when another event or condition is *in progress*. This might mean that the event counts the occupancy of a queue or other microarchitectural structure tracking the event. It is usually IMPLEMENTATION_DEFINED when an event is in progress.

For example, the MEM_ACCESS_RD_PERCYC event counts when the MEM_ACCESS_RD event is in progress, meaning on each *Processor cycle*, the counter increments by the number of *Memory-read* operations that are in progress.

These events can be used to calculate the average number of events in progress. In the case of MEM_ACCESS_RD_PERCYC event, this is also the average read latency. However, the user has to be aware that the definition of *in progress* might not include all parts of the operation.

Example: In an example implementation a Memory-read_operation generated by a load instruction which occupies 3 pipeline stages in the PE before generating a MEM_ACCESS_RD event when the PE starts to access memory. The event is then considered to be *in progress* until the data is returned to the PE. In the case of a Normal Cacheable access, the PE first looks in the Level 1 data cache, and if the address is cached, returns data in 2 cycles. Once the data is returned, it is another cycle before the result can be forwarded to any other instruction.

In this example, if all loads hit in the Level 1 cache, the average read latency calculated using the MEM_ACCESS_RD_PERCYC and MEM_ACCESS_RD events might be 2 cycles. Although this value is correct for the implementation-specific definition of this event, it has to be adjusted by a constant 4 additional cycles to match the more commonly understood definition of Level 1 cache access latency, which for this example would be quoted as 6 cycles.

A cross-reference to this definition is added to each of the MEM_ACCESS_RD_PERCYC, INST_FETCH_PERCYC, BUS_REQ_RD_PERCYC, DTLB_WALK_PERCYC, and ITLB_WALK_PERCYC events, in section D7.10.3 (Common event numbers).

2.34 D18202

In section D10.2.7 (Operation Type packet), subsection 'Operation Type packet payload (load/store)', the PRED field definition is updated to include the following text:

Predicated SVE operation. The operation is one of the following:

- If FEAT_SPEv1p2 is implemented, a predicated load operation that writes to one or more vector destination registers under a Governing predicate using zeroing predication.

- A predicated store of one or more vector registers. If FEAT_SPEv1p2 is not implemented, it is **IMPLEMENTATION DEFINED** whether this field is 0b0 or 0b1 for a predicated load operation that writes to one or more vector destination registers under a Governing predicate using zeroing predication.

2.35 D18216

In section C7.2.53 (FCADD), the line:

```
bits(datasize) operand3 = V[d];
```

in the operational pseudocode is removed.

2.36 D18225

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', the definition of '0x8140, L1D_CACHE_RW, Level 1 data or unified cache demand access' that reads:

The counter counts each access counted by L1D_CACHE that is due to a demand read or demand write access.

is updated to read:

The counter counts each access counted by L1D_CACHE that is due to a demand read or demand write access. This includes accesses made for translation table walks made by demand accesses.

Similarly, each L<n>D_CACHE_RW, L<n>I_CACHE_RD, CACHE_PRFM, and CACHE_HWPRF event definition is updated to mention translation table walks.

2.37 D18240

In section G8.2.75 (HTCR, Hyp Translation Control Register), the description in HWU62, bit [28] that currently reads:

Otherwise:

Reserved, **RES0**.

is corrected to read:

Otherwise:

Reserved, **RAZ/WI**.

Equivalent changes are made in HWU61, bit [27], HWU60, bit [26], and HWU59, bit [25].

2.38 R18243

In section D13.2.48 (HCR_EL2, Hypervisor Configuration Register), the following text is added to the description of DCT, bit [57]:

| This bit is permitted to be cached in a TLB.

2.39 C18253

In section I3.1.4 (Access permissions for external views of the Performance Monitors), in Table I3-1 'Access permissions for the Performance Monitors registers', the following changes are made:

- The entries that read 'Access is **IMPLEMENTATION DEFINED**' are changed to read '**IMPLEMENTATION DEFINED** registers', with a link to a new footnote.
- The new footnote that is added to the table reads: 'See **IMPLEMENTATION DEFINED** registers on page H8-7470.'

2.40 D18258

In section D13.2.117 (SCTLR_EL2, System Control Register (SCTLR_EL2)), the text in the description of EPAN, bit [57] that reads:

| When FEAT_PAN3 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

is corrected to read:

| When FEAT_PAN3 is implemented, HCR_EL2.E2H == 1:

2.41 D18262

In section D1.16.1 (Wait for Event mechanism and Send event), the text that reads:

| The architecture does not define the exact nature of the low-power state, except that the execution of a WFE or a WFET instruction, must not cause a loss of memory coherency.

is clarified to read:

| The architecture does not define the exact nature of the low-power state, except that the execution of a WFE or a WFET instruction, must not cause a loss of memory coherency or architectural state.

In section D1.16.2 (Wait For Interrupt) the text that reads:

The architecture does not define the exact nature of the low-power state, except that the execution of a WFI or WFIT instruction must not cause a loss of memory coherency.

is clarified to read:

The architecture does not define the exact nature of the low-power state, except that:

- The execution of a WFI or WFIT instruction must not cause a loss of memory coherency.
- If the system is configured such that the WFI or WFIT instruction can be completed, then the WFI or WFIT instruction must not cause a loss of architectural state.

Note: In some implementations, based on the configuration of system specific registers, WFI can be used as part of a powerdown sequence where no interrupts will cause WFI wakeup events, and restoration of power involves resetting of the PE. In those cases, the WFI is permitted to cause a loss of architectural state, as it is assumed that this state will have been saved by software as part of the powerdown sequence before the WFI.

The equivalent changes are made to sections G1.18.1 (Wait for Event and Send Event) and G1.18.2 (Wait for Interrupt).

2.42 D18266

In section H2.4.7 (Exceptions in Debug state), the bullet point that reads:

- Any attempt to execute an instruction bit pattern that is an allocated instruction at the current Exception level, but is listed in Executing instructions in Debug state on page H2-7349 as undefined in Debug state, generates an exception that is taken to the current Exception level, or to EL1 if executing at ELO.

is changed to read:

- Any attempt to execute an instruction bit pattern that is an allocated instruction at the current Exception level, but is listed in Executing instructions in Debug state on page H2-7349 as **UNDEFINED** in Debug state, generates an exception.

and the indented bullet point that reads:

- When the value of EDSCR.SDD is 1, are treated as **UNDEFINED** and generate an exception that is taken to the current Exception level, or to EL1 if the instruction is executed at ELO. If the exception is taken to an Exception level that is using AArch32 it is taken as an Undefined Instruction exception.

is changed to read:

- When the value of EDSCR.SDD is 1, are treated as **UNDEFINED** and generate an exception.

Additionally, the sentence that reads:

Otherwise configurable traps, enables, and disables for instructions are unaffected by Debug state, and executing an affected instruction generates the appropriate exception.

is deleted, and the sentence that reads:

Otherwise, synchronous exceptions, including Data Aborts, are generated as they would be in Non-debug state and taken to the appropriate Exception level in Debug state.

is changed to read:

Otherwise, synchronous exceptions are generated as they would be in Non-debug state and taken to the appropriate Exception level in Debug state.

2.43 D18272

In section B2.3.10 (Memory barriers), in the subsection 'Data Synchronization Barrier (DSB)', the text that currently reads:

In addition, no instruction that appears in program order after the DSB instruction can alter any state of the system or perform any part of its functionality until the DSB completes other than:

- Being fetched from memory and decoded.
- Reading the general-purpose, SIMD and floating-point, Special-purpose, or System registers that are directly or indirectly read without causing side-effects.

is relaxed to read:

In addition, no instruction that appears in program order after the DSB instruction can alter any state of the system or perform any part of its functionality until the DSB completes other than:

- Being fetched from memory and decoded.
- Reading the general-purpose, SIMD and floating-point, Special-purpose, or System registers that are directly or indirectly read without causing side-effects.
- If FEAT_ETS is not implemented, having any virtual addresses of loads and stores translated.

The equivalent changes are made in section E2.3.10 (Memory barriers), in the subsection 'Data Synchronization Barrier (DSB)'.

2.44 D18282

In section F6.1.124 VMLAL (integer), the operand "<type><size>" is removed, and is replaced with the operand "<dt>", which reads:

Is the data type for the elements of the operands, encoded in "U:size":

- S8 when U = 0, size = 00

- S16 when U = 0, size = 01
- S32 when U = 0, size = 10
- U8 when U = 1, size = 00
- U16 when U = 1, size = 01
- U32 when U = 1, size = 10

The same change is made in section F6.1.129 VMLSL (integer).

Similarly, in section F6.1.122 VMLA (integer), the operand "<type><size>" is removed, and is replaced with the operand "<dt>", which reads:

Is the data type for the elements of the operands, encoded in "size":

- I8 when size = 00
- I16 when size = 01
- I32 when size = 10

The same change is made in section F6.1.127 VMLS (integer).

2.45 D18284

In section D2.12.9 (Exception syndrome information and preferred return address), subsection 'Exception syndrome information', the text that reads:

When an instruction has been stepped, if the stepped instruction was a conditional Load-Exclusive instruction that failed its Condition code test, then ESR_ELx.EX is set to a **CONSTRAINED UNPREDICTABLE** choice of 0 or 1.

is corrected to read:

When an instruction has been stepped, if the stepped instruction was a conditional Load-Exclusive instruction that failed its Condition code test, then ESR_ELx.ISV is set to 1 and ESR_ELx.EX is set to a **CONSTRAINED UNPREDICTABLE** choice of 0 or 1.

2.46 D18288

In section J1.3.3 (shared/functions), in the function Hint_WFE(), the lines that read:

```
if IsEventRegisterSet() then
    ClearEventRegister();
else
```

are changed to read:

```
if IsEventRegisterSet() then
    ClearEventRegister();
else
    if localtimeout < 0 || !LocalTimeoutEvent(localtimeout)
```

and all the remaining pseudo-code in the function is indented by a further 4 spaces.

Similarly, in the function Hint_WFI(), the line that reads:

```
if !InterruptPending() then
```

is changed to read:

```
if !InterruptPending() || (localtimeout >=0 && LocalTimeoutEvent(localtimeout) )
    then
```

2.47 D18291

In sections C6.2.125 (LDGM), C6.2.261 (STGM), and C6.2.311 (STZGM), the following text is removed:

■ If ID_AA64PFR1_EL1 != 0b0010, this instruction is **UNDEFINED**.

2.48 D18294

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', in the description of event '0x8130, L1D_TLB_RW, Level 1 data or unified TLB demand access', the text that reads:

■ The counter counts each access counted by L1D_TLB that is not counted by L1D_TLB_PRFM.

is corrected to read:

■ The counter counts each access counted by L1D_TLB that is due to a demand Memory-read operation or demand Memory-write operation.

Equivalent changes are made to the description of event '0x8131, L1I_TLB_RD, Level 1 instruction TLB demand access'.

2.49 D18299

In section D5.7.2 (Enhanced support for nested virtualization), subsection 'Loads and stores generated by transforming register accesses', the bullet point that reads:

- The addressees the memory access is translated by the EL2 translation regime.

is corrected to read:

- The addressees the memory access is translated by the EL2 or EL2&O translation regime.

2.50 D18300

In section D5.5.1 (The stage 1 memory region attributes), subsection 'Stage 1 definition of the XS attribute', the reference to a MAIR_ELx.Attrn encoding of 0b1010000 in the following bullet point is corrected to 0b10100000:

- Inner Write-through Cacheable and Outer Write-through Cacheable memory types that use the MAIR_ELx.Attrn encoding 0b1010000.

2.51 C18301

In section D5.2.5 (Translation tables and the translation process), subsection 'Ordering of memory accesses from translation table walks', the text that reads:

The explicit memory write effect to the translation tables is guaranteed to be observable, to the extent required by the shareability attributes, only after the execution of a DSB instruction.

is modified to read:

The explicit memory write effect to the translation tables is guaranteed to be observable, to the extent required by the shareability attributes, after the execution of a DSB instruction.

2.52 R18319

In section D13.3.20 (MDSCR_EL1, Monitor Debug System Control Register), the following relaxation is added to the RXO field:

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.RXO. Reads and writes of this bit are indirect accesses to EDSCR.RXO. A write to MDSCR_EL1 when OSLSR_EL1.OSLK == 1 with RXO=1 and ERR=0 sets EDSCR.{RXO,ERR} to **UNKNOWN** values.

A similar relaxation is added to the TXU field.

2.53 D18330

The Arm ARM is somewhat inconsistent in its use of "prefetch" and "preload" to describe the bringing in of items into caches either by hardware prediction or as a result of some prefetch or preload instructions. In future versions of the Arm ARM, this will be cleaned up. The term "prefetch" will be used for this functionality, with "hardware prefetch" used where the prefetch is predicted by hardware, and "software prefetch" used where the prefetch is prompted by particular instructions (such as the AArch64 PRFM or AArch32 PLD instructions).

2.54 D18347

In section D13.2.68 (ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1), in the SSBS field, the text that reads:

FEAT_SSBS implements the functionality identified by the value 0b0010.

is replaced with:

FEAT_SSBS2 implements the functionality identified by the value 0b0010.

2.55 D18352

In section D13.2.66 (ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2), the text in the accessibility pseudocode that currently reads:

```
if EL2Enabled() && (!IsZero(ID_AA64MMFR2_EL1) || boolean IMPLEMENTATION_DEFINED
    "ID_AA64MMFR2 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
```

is corrected to:

```
if EL2Enabled() && (!IsZero(ID_AA64MMFR2_EL1) || boolean IMPLEMENTATION_DEFINED
    "ID_AA64MMFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
```

A similar change is made in section D13.2.71 (ID_DFR1_EL1, Debug Feature Register 1).

2.56 D18354

In section C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context), the line that currently reads:

Cache prefetch predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

is clarified to read:

Cache prefetch predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot influence speculative execution occurring after the instruction is complete and synchronized.

Equivalent clarifications are made in sections C6.2.65 (CPP) and G8.2.34 (CPPRCTX, Cache Prefetch Prediction Restriction by Context).

2.57 D18366

In section D13.2.103 (PAR_EL1, Physical Address Register), in the description of the ATTR field, the text that reads:

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

is clarified to read:

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

2.58 D18371

In section D11.2.4 (Timers), in the subsection 'Operation of the CompareValue views of the timers', the text that reads:

Counter The physical counter value, that can be read from the CNTPCT_ELO register.

is clarified to read:

Counter The physical counter value, that can be read from the CNTPCT_ELO register when read from EL2 or EL3.

Within the same section, in the subsection 'Operation of the TimerValue views of the timers', the text that reads:

This view of a timer depends on the following behavior of accesses to TimerValue registers:

Reads $\text{TimerValue} = (\text{CompareValue} - (\text{Counter} - \text{Offset}))[\text{31:0}]$ Writes $\text{CompareValue} = ((\text{Counter} - \text{Offset})[\text{63:0}] + \text{SignExtend}(\text{TimerValue}))[\text{63:0}]$

is modified to read:

This view of a timer depends on the following behavior of accesses to TimerValue registers:

Reads $\text{TimerValue} = (\text{CompareValue} - (\text{Counter} - \text{ModOffset}))[\text{31:0}]$ Writes $\text{CompareValue} = ((\text{Counter} - \text{ModOffset})[\text{63:0}] + \text{SignExtend}(\text{TimerValue}))[\text{63:0}]$

Where ModOffset is:

- Offset for all timer registers other than the EL1 physical timer accessed through CNTP_CTL_ELO.
- Offset for the EL1 physical timer accessed through CNTP_CTL_ELO if ID_AA64MMFR0_EL1.ECV is less than 0b10 or CNTHCTL_EL2.ECV is 0b0.
- Offset for the EL1 physical timer accessed through CNTP_CTL_ELO if ID_AA64MMFR0_EL1.ECV is 0b10 and CNTHCTL_EL2.ECV is 0b1 and the access is from ELO or EL1.
- 0 for the EL1 physical timer accessed through CNTP_CTL_ELO if ID_AA64MMFR0_EL1.ECV is 0b10 and CNTHCTL_EL2.ECV is 0b1 and the access is from EL2 or EL3.

Similarly, in section D13.8.18 (CNTP_TVAL_ELO, Counter-timer Physical Timer TimerValue register), the following Note is added to the TimerValue, bits [31:0] description:

Note: the value of CNTPCT_ELO used in these calculations is the value seen at the Exception Level that the CNTPCT_ELO register is being read/written from.