



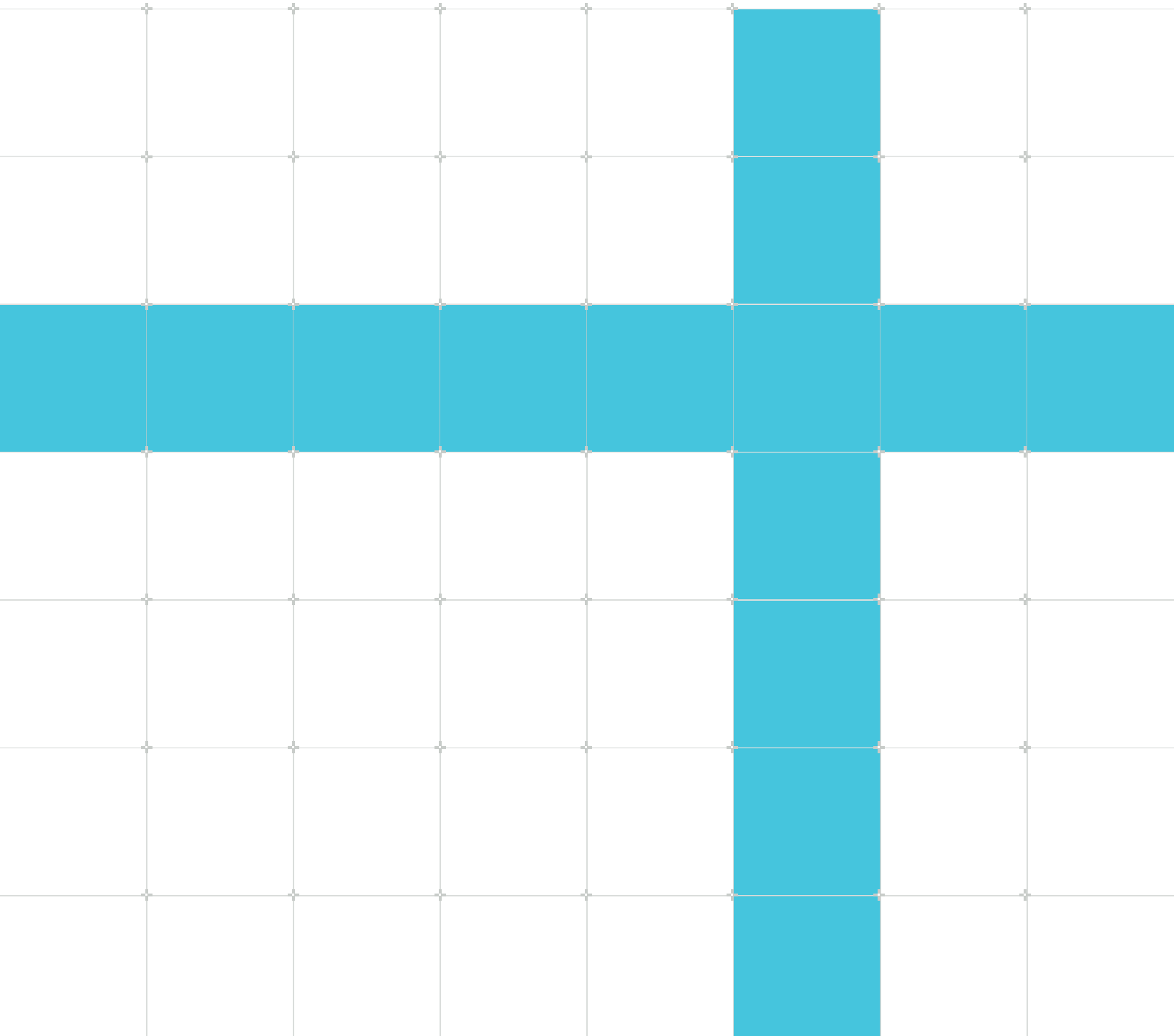
Arm[®] Architecture Reference Manual Armv8, for A-profile architecture

Known issues in Issue G.b

Non-Confidential

Issue 05

Copyright © 2020–2022 Arm Limited (or its affiliates). 102105_G.b_05
All rights reserved.



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

Known issues in Issue G.b

Copyright © 2020–2022 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
G.b-02	30 September 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 17 September 2021
G.b-03	29 October 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 22 October 2021
G.b-04	30 November 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 19 November 2021
G.b-05	31 January 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 7 January 2022

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, 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.

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 © 2020–2022 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.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

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.....	10
1.1 Conventions.....	10
1.2 Additional reading.....	11
1.3 Other information.....	11
2 Known issues.....	12
2.1 D14596.....	12
2.2 D14737.....	12
2.3 D16720.....	12
2.4 D17015.....	13
2.5 D17077.....	13
2.6 D17119.....	15
2.7 D17591.....	15
2.8 D17672.....	16
2.9 D17711.....	17
2.10 D17736.....	17
2.11 R17743.....	18
2.12 C17811.....	18
2.13 R17871.....	18
2.14 R17872.....	19
2.15 D17876.....	20
2.16 D17896.....	20
2.17 D17905.....	20
2.18 E17909.....	21
2.19 D17919.....	22
2.20 D17956.....	24
2.21 D18000.....	25
2.22 D18001.....	27
2.23 D18002.....	28
2.24 D18012.....	29
2.25 D18024.....	30
2.26 D18033.....	31

2.27 D18035.....	33
2.28 D18042.....	34
2.29 D18055.....	34
2.30 D18073.....	35
2.31 D18093.....	35
2.32 D18094.....	36
2.33 D18106.....	36
2.34 D18109.....	37
2.35 D18118.....	38
2.36 D18133.....	39
2.37 D18138.....	39
2.38 D18140.....	40
2.39 D18144.....	40
2.40 D18152.....	41
2.41 D18160.....	41
2.42 D18162.....	42
2.43 D18165.....	42
2.44 D18169.....	43
2.45 D18183.....	43
2.46 R18187.....	43
2.47 D18196.....	45
2.48 D18199.....	45
2.49 D18200.....	46
2.50 D18202.....	46
2.51 D18203.....	47
2.52 D18216.....	47
2.53 D18225.....	47
2.54 D18240.....	47
2.55 C18241.....	48
2.56 R18243.....	48
2.57 C18253.....	48
2.58 D18258.....	48
2.59 D18262.....	49
2.60 D18264.....	49
2.61 D18266.....	50
2.62 D18272.....	51

2.63 D18282.....	51
2.64 D18284.....	52
2.65 D18288.....	52
2.66 D18290.....	54
2.67 D18291.....	55
2.68 D18294.....	55
2.69 D18299.....	56
2.70 D18300.....	56
2.71 C18301.....	56
2.72 D18305.....	56
2.73 R18319.....	59
2.74 D18325.....	59
2.75 D18330.....	61
2.76 R18338.....	61
2.77 D18347.....	61
2.78 D18352.....	62
2.79 R18353.....	62
2.80 D18354.....	62
2.81 D18358.....	63
2.82 D18364.....	63
2.83 D18366.....	64
2.84 D18367.....	64
2.85 D18370.....	65
2.86 D18371.....	65
2.87 D18403.....	66
2.88 D18426.....	67
2.89 D18428.....	67
2.90 D18431.....	70
2.91 D18434.....	71
2.92 D18443.....	71
2.93 D18444.....	72
2.94 D18451.....	73
2.95 D18454.....	73
2.96 D18457.....	74
2.97 D18459.....	74
2.98 D18463.....	74

2.99 D18464.....	75
2.100 D18465.....	75
2.101 R18467.....	75
2.102 D18478.....	75
2.103 D18482.....	76
2.104 R18495.....	76
2.105 D18507.....	77
2.106 D18508.....	78
2.107 D18520.....	78
2.108 D18525.....	79
2.109 D18530.....	79
2.110 D18532.....	79
2.111 C18533.....	80
2.112 C18534.....	80
2.113 D18538.....	81
2.114 D18551.....	81
2.115 D18554.....	82
2.116 D18558.....	82
2.117 D18563.....	82
2.118 D18564.....	82
2.119 D18565.....	83
2.120 R18570.....	83
2.121 C18576.....	84
2.122 C18578.....	84
2.123 D18581.....	84
2.124 D18582.....	85
2.125 D18593.....	86
2.126 D18595.....	86
2.127 C18597.....	87
2.128 D18601.....	87
2.129 E18617.....	88
2.130 D18620.....	88
2.131 D18628.....	89
2.132 D18629.....	89
2.133 D18637.....	90
2.134 R18641.....	90

2.135 E18645.....	91
2.136 D18647.....	92
2.137 D18650.....	92
2.138 R18653.....	93
2.139 D18672.....	93
2.140 C18676.....	93
2.141 D18677.....	94
2.142 C18680.....	94
2.143 D18683.....	95
2.144 D18685.....	95
2.145 D18698.....	96
2.146 C18700.....	97
2.147 D18703.....	98
2.148 C18704.....	98
2.149 D18711.....	99
2.150 D18712.....	99
2.151 E18715.....	99
2.152 C18720.....	100
2.153 R18721.....	101
2.154 D18735.....	101
2.155 D18739.....	101
2.156 R18746.....	102
2.157 D18781.....	102

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>	Citations.
bold	Interface elements, such as menu names. Signal names. Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace bold	Language keywords when used outside example code.
monospace <u>underline</u>	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	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .
 Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
 Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
 Danger	Requirements for the system. Not following these requirements will result in system failure or damage.

Convention	Use
 Note	An important piece of information that needs your attention.
 Tip	A useful tip that might make it easier, better or faster to perform a task.
 Remember	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



Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at <http://www.adobe.com>

1.3 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 (DDI 0487), 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 D14737

In section D5.3.1 (VMSAv8-64 translation table level -1, level 0, level 1, and level 2 descriptor formats), in Figure D5-14 'VMSAv8-64 level 0, level 1 and level 2 descriptor formats with 48-bit OAs', the range of bits that are IGNORED is corrected to [58:51].

A similar change is made in Figure D5-15 'VMSAv8-64 level -1, level 0, level 1 and level 2 descriptor formats, 4KB or 16KB granule with 52-bit OAs'.

2.3 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

2.4 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 0x06.
- 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.

2.5 D17077

In section J1.2.4 (aarch32/translation), the code in the functions AArch32.S1TranslateLD() and AArch32.S1TranslateSD() reading:

```
if (regime == Regime_EL10 && AArch32.EL2Enabled(ss) &&
    (if ELStateUsingAArch32(EL2, ss == SS_Secure) then HCR.VM else
HCR_EL2.VM) == '1') then
    // Shareability of target memory subject to stage 2 translation
    // is maintained as input to stage 2
    memattrs.shareability = walkstate.memattrs.shareability;
else
    memattrs.shareability = NormaliseShareability(memattrs);
```

is updated to read:

```
// Shareability value of stage 1 translation subject to stage 2 is
IMPLEMENTATION_DEFINED
// to be either effective value or descriptor value
if (regime == Regime_EL10 && AArch32.EL2Enabled(ss) &&
    (if ELStateUsingAArch32(EL2, ss == SS_Secure) then HCR.VM else
HCR_EL2.VM) == '1' &&
    !(boolean IMPLEMENTATION_DEFINED "Apply effective shareability at stage
1")) then
    memattrs.shareability = walkstate.memattrs.shareability;
else
    memattrs.shareability = EffectiveShareability(memattrs);
```

In the same section, the code within the functions AArch32.S1WalkLD() and AArch32.S1WalkSD() reading:

```
// Shareability of target memory subject to stage 2 translation
// is maintained as input to stage 2.
if (regime == Regime_EL10 && AArch32.EL2Enabled(ss) &&
    (if ELStateUsingAArch32(EL2, ss == SS_Secure) then HCR.VM else
HCR_EL2.VM) == '1') then
    walkaddress.memattrs.shareability = walkstate.memattrs.shareability;
else
    walkaddress.memattrs.shareability =
NormaliseShareability(walkaddress.memattrs);
```

is updated to read:

```
// Shareability value of stage 1 translation subject to stage 2 is
IMPLEMENTATION DEFINED
// to be either effective value or descriptor value
if (regime == Regime_EL10 && AArch32.EL2Enabled(ss) &&
    (if ELStateUsingAArch32(EL2, ss == SS_Secure) then HCR.VM else
HCR_EL2.VM) == '1' &&
    !(boolean IMPLEMENTATION_DEFINED "Apply effective shareability at
stage 1")) then
    walkaddress.memattrs.shareability = walkstate.memattrs.shareability;
else
    walkaddress.memattrs.shareability =
EffectiveShareability(walkaddress.memattrs);
```

In section J1.1.5 (aarch64/translation), the code within the function AArch64.S1Translate() reading:

```
// Shareability of target memory subject to stage 2 translation
// is maintained as input to stage 2
if regime == Regime_EL10 && EL2Enabled() && HCR_EL2.VM == '1' then
    memattrs.shareability = walkstate.memattrs.shareability;
else
    memattrs.shareability = NormaliseShareability(memattrs);
```

is updated to read:

```
// Shareability value of stage 1 translation subject to stage 2 is
IMPLEMENTATION DEFINED
// to be either effective value or descriptor value
if (regime == Regime_EL10 && EL2Enabled() && HCR_EL2.VM == '1' &&
    !(boolean IMPLEMENTATION_DEFINED "Apply effective shareability at stage
1")) then
    memattrs.shareability = walkstate.memattrs.shareability;
else
    memattrs.shareability = EffectiveShareability(memattrs);
```

In the same section, the code within the function AArch64.S1Walk() reading:

```
// Shareability of target memory subject to stage 2 translation
// is maintained as input to stage 2.
if regime == Regime_EL10 && EL2Enabled() && HCR_EL2.VM == '1' then
    walkmemattrs.shareability = walkstate.memattrs.shareability;
else
    walkmemattrs.shareability = NormaliseShareability(walkmemattrs);
```

is updated to read:

```
// Shareability value of stage 1 translation subject to stage 2 is
IMPLEMENTATION DEFINED
// to be either effective value or descriptor value
if (regime == Regime_EL10 && EL2Enabled() && HCR_EL2.VM == '1' &&
    !(boolean IMPLEMENTATION_DEFINED "Apply effective shareability at
stage 1")) then
    walkmemattrs.shareability = walkstate.memattrs.shareability;
else
    walkmemattrs.shareability = EffectiveShareability(walkmemattrs);
```

2.6 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 ‘O’.
- ‘imm3H’ cannot be ‘000’.

2.7 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).

2.8 D17672

In section J1.1.4 (aarch64/instrs), the code in AArch64.ExceptionReturn() that currently reads as:

```
// Attempts to change to an illegal state will invoke the Illegal Execution state
mechanism
bits(2) source_el = PSTATE.EL;
SetPSTATEFromPSR(spsr);
...
if PSTATE.IL == '1' && spsr<4> == '1' && spsr<20> == '0' then
```

is updated to read as:

```
// Attempts to change to an illegal state will invoke the Illegal Execution state
mechanism
bits(2) source_el = PSTATE.EL;
boolean illegal_psr_state = IllegalExceptionReturn(spsr);
SetPSTATEFromPSR(spsr, illegal_psr_state);
...
if illegal_psr_state && spsr<4> == '1' then
```

Additionally, in section J1.3.3 (shared/functions), the code in SetPSTATEFromPSR() that currently reads as:

```
SetPSTATEFromPSR(bits(N) spsr)
...
if IllegalExceptionReturn(spsr) then
    PSTATE.IL = '1';
```

is updated to read as:

```
SetPSTATEFromPSR(bits(N) spsr)
    boolean illegal_psr_state = IllegalExceptionReturn(spsr);
    SetPSTATEFromPSR(spsr, illegal_psr_state);

SetPSTATEFromPSR(bits(N) spsr, boolean illegal_psr_state)
```



```
...  
if illegal_psr_state then  
    PSTATE.IL = '1';
```

2.9 D17711

In section A1.5.5 (NaN handling and the Default NaN), the following text is added:

A PE is forbidden to generate a NaN whose value is strongly correlated to the values of non-NaN inputs as a speculative result of a floating-point calculation not involving NaN inputs.

In section B2.3.9 (Restrictions on the effects of speculation), a bullet point is added:

When writing new instructions to memory, there is no requirement for a speculative barrier (SB) instruction to prevent speculative execution of the old code. See also K11.5.2 Instruction cache maintenance instructions.

Similar changes are also made to sections E1.3.6 (NaN handling and the Default NaN) and E2.3.9 (Restrictions on the effects of speculation).

2.10 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.
```

2.11 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.

In section D9.9 (Synchronization and Statistical Profiling), the following text is removed:

Although the SPU acts as another observer in the system, for determining the Shareability domain of the DSB instructions, the writes of sample records are treated as coming from the PE that is being profiled.

2.12 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.

2.13 R17871

In section B2.3.2 (Dependency definitions), the following definitions are added:

- Pick Basic dependency.

- Pick Address dependency.
- Pick Data dependency.
- Pick Control dependency.
- Pick dependency.

In section B2.3.3 (Ordering relations), the definition ‘Locally-ordered-before’ is updated, and the following definitions are added:

- Pick-ordered-before.
- Pick-locally-ordered-before.

In section B2.3.6 (External ordering constraints), the following definitions are updated:

- Ordered-before.
- External completion requirement.
- External global completion requirement.

The equivalent changes are made for AArch32 in the following sections:

- E2.3.2 (Dependency definitions).
- E2.3.3 (Ordering relations).
- E2.3.6 (External ordering constraints).

2.14 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 (resp. a Load-Exclusive 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 atomic instruction as R1 (resp. the successful Store-Exclusive instruction paired with the Load-Exclusive instruction that generated R1).

2.15 D17876

In section B2.3.9 (Restrictions on the effects of speculation), in the subsection ‘Restrictions on the effects of speculation’, the following text is added:

Data loaded as a result of a speculative accesses made after TLBI + DSB + ERET using a translation that was invalidated by the TLBI cannot be used to form an address, to generate condition codes, or to generate SVE predicate values to be used by other instructions in the speculative sequence and the execution timing of any other instructions in the speculative sequence is not a function of the data loaded.

A similar change is also made to section E2.3.9 (Restrictions on the effects of speculation).

2.16 D17896

This reverts the change that was originally communicated.

In section D13.2.123 (TCR_EL1, Translation Control Register (EL1)), the text under the ‘Otherwise’ heading in the HWU* fields that reads:

Reserved, RAZ/WI.

is corrected to read:

Reserved, **RES0**.

Equivalent changes are made to HWU* fields in the following sections:

- D13.2.124 (TCR_EL2, Translation Control Register (EL2)).
- D13.2.125 (TCR_EL3, Translation Control Register (EL3)).
- D13.2.148 (VTCTR_EL2, Virtualization Translation Control Register).
- G8.2.165 (TTBCR2, Translation Table Base Control Register 2).
- G8.2.171 (VTCTR, Virtualization Translation Control Register).

The equivalent change is made to the T2E field in section G8.2.164 (TTBCR, Translation Table Base Control Register).

2.17 D17905

Armv8.7 introduced the following features:

- FEAT_LS64, which supports atomic single-copy 64-byte loads and stores without return.
- FEAT_LS64_V, which supports atomic single-copy 64-byte stores with return.
- FEAT_LS64_ACCDATA, which supports atomic single-copy 64-byte ELO stores with return.

Several instances of FEAT_LS64_V and FEAT_LS64_ACCDATA are incorrectly identified as FEAT_LS64. These instances are corrected across the ArmARM.

2.18 E17909

Arm® Architecture Reference Manual Armv8, for A-profile architecture, Issue G.b introduces FEAT_WFXT2, which adds support for reporting the register number for trapped WFXT instructions in ESR_ELx. E17909 removes FEAT_WFXT2, and adds the functionality that was introduced by FEAT_WFXT2 to FEAT_WFXT.

In section A2.10.1 (Architectural features added by Armv8.7), the title that reads 'FEAT_WFXT and FEAT_WFXT2, WFE and WFI instructions with timeout' is updated to read 'FEAT_WFXT, WFE and WFI instructions with timeout', and the text under the title is updated to read:

FEAT_WFXT 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_WFXT is mandatory in Armv8.7 implementations.

The ID_AA64ISAR2_EL1.WFXT field identifies the presence of FEAT_WFXT.

In section D13.2.64 (ID_AA64ISAR2_EL1), the 'WFXT, 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_WFXT 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_WFXT 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.

2.19 D17919

In section J1.1.5 (aarch64/translation), the function `AArch64.MemSwapTableDesc()` reading:

```
(FaultRecord, bits(64)) AArch64.MemSwapTableDesc(...)
    descupdateaccess = CreateAccessDescriptor(AccType_ATOMICRW);
    if ee == '1' then
        new_desc = BigEndianReverse(new_desc);
        prev_desc = BigEndianReverse(prev_desc);

    // All observers in the shareability domain observe the
    // following memory read and write accesses atomically.
    (memstatus, mem_desc) = PhysMemRead(descupdateaddress, 8, descupdateaccess);
    if IsFault(memstatus) then
        iswrite = FALSE;
        fault = HandleExternalTTWAbort(memstatus, iswrite, descupdateaddress,
        descupdateaccess, 8, fault);
        if IsFault(fault.statuscode) then
            fault.acctype = AccType_ATOMICRW;
            return (fault, bits(64) UNKNOWN);
```

```

    if mem_desc == prev_desc then
        memstatus = PhysMemWrite(descupdateaddress, 8, descupdateaccess, new_desc);
        iswrite = TRUE;
        if IsFault(memstatus) then
            fault = HandleExternalTTWAbort(memstatus, iswrite, descupdateaddress,
            descupdateaccess, 8, fault);
            fault.acctype = memstatus.acctype;
            if IsFault(fault.statuscode) then
                fault.acctype = AccType_ATOMICRW;
                return (fault, bits(64) UNKNOWN);
            mem_desc = new_desc;

    if ee == '1' then
        mem_desc = BigEndianReverse(mem_desc);

    assert mem_desc == new_desc;

    return (fault, mem_desc);

```

is updated to read:

```

(FaultRecord, bits(64)) AArch64.MemSwapTableDesc(...)
    descupdateaccess = CreateAccessDescriptor(AccType_ATOMICRW);

    // All observers in the shareability domain observe the
    // following memory read and write accesses atomically.
    (memstatus, mem_desc) = PhysMemRead(descupdateaddress, 8, descupdateaccess);
    if ee == '1' then
        mem_desc = BigEndianReverse(mem_desc);

    if IsFault(memstatus) then
        iswrite = FALSE;
        fault = HandleExternalTTWAbort(memstatus, iswrite, descupdateaddress,
        descupdateaccess, 8, fault);
        if IsFault(fault.statuscode) then
            fault.acctype = AccType_ATOMICRW;
            return (fault, bits(64) UNKNOWN);

    if mem_desc == prev_desc then
        ordered_new_desc = if ee == '1' then BigEndianReverse(new_desc) else
        new_desc;
        memstatus = PhysMemWrite(descupdateaddress, 8, descupdateaccess,
        ordered_new_desc);

        if IsFault(memstatus) then
            iswrite = TRUE;
            fault = HandleExternalTTWAbort(memstatus, iswrite, descupdateaddress,
            descupdateaccess, 8, fault);
            fault.acctype = memstatus.acctype;
            if IsFault(fault.statuscode) then
                fault.acctype = AccType_ATOMICRW;
                return (fault, bits(64) UNKNOWN);

        // Reflect what is now in memory (in little endian format)
        mem_desc = new_desc;

    return (fault, mem_desc);

```

In section J1.1.5 (aarch64/translation), the function AArch64.S1Translate() reading:

```

(FaultRecord, AddressDescriptor) AArch64.S1Translate(...)
    ...
    (fault, descaddress, walkstate, descriptor) = AArch64.S1Walk(fault, walkparams,
    va, regime, acctype, iswrite);
    ...

```

```
(fault, mem_desc) = AArch64.MemSwapTableDesc(fault, descriptor, new_desc,
walkparams.ee, descupdateaddress);

if fault.statuscode != Fault_None then
    return (fault, AddressDescriptor UNKNOWN);
```

is updated to read:

```
(FaultRecord, AddressDescriptor) AArch64.S1Translate(...)
...
repeat
    (fault, descaddress, walkstate, descriptor) = AArch64.S1Walk(fault,
walkparams, va, regime, acctype, iswrite);
    ...
    (fault, mem_desc) = AArch64.MemSwapTableDesc(fault, descriptor,
new_desc, walkparams.ee, descupdateaddress);

until new_desc == descriptor || mem_desc == new_desc;

if fault.statuscode != Fault_None then
    return (fault, AddressDescriptor UNKNOWN);
```

In section J1.1.5 (aarch64/translation), the function AArch64.S2Translate() reading:

```
(FaultRecord, AddressDescriptor) AArch64.S2Translate(...)
...
(fault, descaddress, walkstate, descriptor) = AArch64.S2Walk(fault, ipa,
walkparams, acctype, iswrite, slaarch64);
...
(fault, mem_desc) = AArch64.MemSwapTableDesc(fault, descriptor, new_desc,
walkparams.ee, descaddress);

if fault.statuscode != Fault_None then
    return (fault, AddressDescriptor UNKNOWN);
```

is updated to read:

```
(FaultRecord, AddressDescriptor) AArch64.S2Translate(...)
...
repeat
    (fault, descaddress, walkstate, descriptor) = AArch64.S2Walk(fault, ipa,
walkparams, acctype, iswrite, slaarch64);
    ...
    (fault, mem_desc) = AArch64.MemSwapTableDesc(fault, descriptor,
new_desc, walkparams.ee, descaddress);

until new_desc == descriptor || mem_desc == new_desc;

if fault.statuscode != Fault_None then
    return (fault, AddressDescriptor UNKNOWN);
```

2.20 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'.

2.21 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.

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 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 cache due to a hardware prefetcher are counted. Otherwise, these events are not counted.

When the L1D_CACHE_RW event is not implemented, it is **IMPLEMENTATION DEFINED** whether accesses to the Level 1 data 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 Memory-read operation or demand Memory-write operation.

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

2.22 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, DC 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, DC 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.

2.23 D18002

In section D7.10.3 (Common event numbers), subsection ‘Common microarchitectural events’, the following changes are made:

- In the definitions of the STALL_FRONTEND_L<n>I, STALL_FRONTEND_MEM, and STALL_FRONTEND_TLB events (0x8159-0x815C), the text that reads ‘demand miss’ is clarified to read ‘demand instruction miss’.
- In the STALL_BACKEND_L<n>D and STALL_BACKEND_TLB events (0x8165-0x8167), the text that reads ‘demand miss’ is clarified to read ‘demand data miss’.
- In the STALL_BACKEND_MEM event (0x4005), the text that reads ‘cache miss’ is clarified to read ‘demand data miss’, and the text that reads ‘the last level of cache’ is clarified to read ‘the last level of data or unified cache’.

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.

Additionally, 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. 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.

The 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.

Implementation of this optional event requires that 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.

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.

2.24 D18012

In section J1.3.4 (shared/trace), the function that reads:

```
boolean TraceAllowed()
...
    TGE_bit = if EL2Enabled() then HCR_EL2.TGE else '0';
    case PSTATE.EL of
        when EL3 TRE_bit = if !HaveAArch64() then TRFCR.E1TRE else '0';
        when EL2 TRE_bit = TRFCR_EL2.E2TRE;
        when EL1 TRE_bit = TRFCR_EL1.E1TRE;
        when EL0 TRE_bit = if TGE_bit == '1' then TRFCR_EL2.E0HTRE else
            TRFCR_EL1.E0TRE;``,`
```

Is corrected to read:

```
boolean TraceAllowed()
...
    case PSTATE.EL of
        when EL3 TRE_bit = if !HaveAArch64() then TRFCR.E1TRE else '0';
        when EL2 TRE_bit = EffectiveE2TRE();
        when EL1 TRE_bit = EffectiveE1TRE();
        when EL0
            if EffectiveTGE() == '1' then
                TRE_bit = EffectiveE0HTRE();
            else
                TRE_bit = EffectiveE0TRE();
```

And the following functions are added:

```
// EffectiveTGE()
// =====
// Returns effective TGE value

bit EffectiveTGE()
    if EL2Enabled() then
        return if ELUsingAArch32(EL2) then HCR.TGE else HCR_EL2.TGE;
```

```

        else
            return '0';          // Effective value of TGE is zero

// EffectiveE2TRE()
// =====
// Returns effective E2TRE value

bit EffectiveE2TRE()
    return if UsingAArch32() then HTRFCR.E2TRE else TRFCR_EL2.E2TRE;

// EffectiveE1TRE()
// =====
// Returns effective E1TRE value

bit EffectiveE1TRE()
    return if UsingAArch32() then TRFCR.E1TRE else TRFCR_EL1.E1TRE;

// EffectiveE0HTRE()
// =====
// Returns effective E0HTRE value

bit EffectiveE0HTRE()
    return if ELUsingAArch32(EL2) then HTRFCR.E0HTRE else TRFCR_EL2.E0HTRE;

// EffectiveE0TRE()
// =====
// Returns effective E0TRE value

bit EffectiveE0TRE()
    return if ELUsingAArch32(EL1) then TRFCR.E0TRE else TRFCR_EL1.E0TRE;

```

2.25 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
elsif secure && el == EL2 then
    return (TRUE, FALSE); // Secure EL2 is using AArch64
elsif HighestELUsingAArch32() then
    return (TRUE, TRUE); // Highest Exception level, and therefore all levels
are using AArch32
elsif 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
elsif secure && el == EL2 then
    return (TRUE, FALSE); // Secure EL2 is using AArch64
elsif HighestELUsingAArch32() then
    return (TRUE, TRUE); // Highest Exception level, and therefore all levels
are using AArch32

```

2.26 D18033

In the following sections:

- D13.2.37 (ESR_EL1, Exception Syndrome Register (EL1)).
- D13.2.38 (ESR_EL2, Exception Syndrome Register (EL2)).
- D13.2.39 (ESR_EL3, Exception Syndrome Register (EL3)).

The following text under 'ISS encoding for an exception from a Data Abort':

Bits[12:11]

When FEAT_RAS is implemented and FEAT_LS64 is not implemented:

SET

Synchronous Error Type. When DFSC is 0b010000, describes the PE error state after taking the Data Abort exception.

0b00 Recoverable state (UER).

0b10 Uncontainable (UC).

0b11 Restartable state (UEO).

All other values are reserved.

Note: Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

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

When FEAT_LS64 is implemented:

LST

Load/Store Type. Used when an LD64B, ST64B, ST64BV, or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

0b01 An ST64BV instruction generated the Data Abort.

0b10 An LD64B or ST64B instruction generated the Data Abort.

0b11 An ST64BV0 instruction generated the Data Abort.

All other values are reserved.

This field is valid only if the DFSC code is 0b110101. It is RES0 for all other aborts.

The reset behavior of this field is:

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

Otherwise:

Reserved, RES0.

is updated and corrected to read:

When DFSC == 0b00xxxx && DFSC != 0b000000:

LST

The Load Store Type. Used when a Translation fault, Access flag fault or Permission fault generated the Data Abort.

0b00 The instruction that generated the Data Abort is not specified.

0b01 An ST64BV instruction generated the Data Abort. Applies when FEAT_LS64_V is implemented.

0b10 An LD64B or ST64B instruction generated the Data Abort. Applies when FEAT_LS64 is implemented.

0b11 An ST64BV0 instruction generated the Data Abort. Applies when FEAT_LS64_ACCDATA is implemented.

The reset behavior of this field is:

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

When DFSC == 0b010000 && FEAT_RAS is implemented:

SET

Synchronous Error Type. Used when a Synchronous External abort, not on a translation table walk or hardware update of the translation table, generated the Data abort. If FEAT_RAS is implemented, the value describes the PE error state after taking the Data Abort exception:

0b00 Recoverable state (UER).

0b10 Uncontainable (UC).

0b11 Restartable state (UEO).

All other values are reserved.

Note: Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

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

Otherwise:

Reserved, RES0.

2.27 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, $n = 0 - 30$).

2.28 D18042

In section D13.2.37 (`ESR_EL1`, Exception Syndrome Register (EL1)), in the ISS encoding for an exception from an Instruction Abort, the following Note in the IFSC, bits [5:0] field is deleted:

Note: Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

The same Note is deleted in the ISS encoding for an exception from a Data Abort, in the DFSC, bits [5:0] field.

The equivalent changes are made in the following sections:

- D13.2.38 (`ESR_EL2`, Exception Syndrome Register (EL2)).
- D13.2.38 (`ESR_EL3`, Exception Syndrome Register (EL3)).

2.29 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'.

2.30 D18073

In section D13.2.67 (ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0), in the definition of CSV3, bits [63:60], the text that reads:

0b0001 Data loaded under speculation with a permission or domain fault cannot be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence.

is clarified to read:

0b0001 Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation.

Accordingly, in section B2.3.9 (Restrictions on the effects of speculation), the first bullet under the subsection titled 'Restrictions on the effects of speculation from Armv8.5' that reads:

Data loaded under speculation with a permission or domain fault cannot be used to form an address, to generate condition codes, or to generate SVE predicate values to be used by other instructions in the speculative sequence.

is changed to read:

Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation.

Corresponding edits are made to section D13.2.87 (ID_PFR2_EL1, AArch32 Processor Feature Register 2), and to AArch32 sections G8.2.100 (ID_PFR2, Processor Feature Register 2) and E2.3.9 (Restrictions on the effects of speculation).

2.31 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>;

elsif HasUnprivileged(regime) then
```

2.32 D18094

In section J1.1.5 (aarch64/translation), the function AArch64.OAOutOfRange() reading:

```
boolean AArch64.OAOutOfRange(TTWState walkstate, bits(3) ps, TGx tgx)
// Output Address size
oasize = AArch64.PhysicalAddressSize(ps, tgx);
baseaddress = walkstate.baseaddress.address;

if oasize < 52 then
    return !IsZero(baseaddress<51:oasize>);
else
    return FALSE;
```

is corrected to read:

```
boolean AArch64.OAOutOfRange(TTWState walkstate, bits(3) ps, TGx tgx, bits(64) ia)
// Output Address size
oasize = AArch64.PhysicalAddressSize(ps, tgx);

if oasize < 52 then
    if walkstate.istable then
        baseaddress = walkstate.baseaddress.address;
        return !IsZero(baseaddress<51:oasize>);
    else
        // Output address
        oa = StageOA(ia, tgx, walkstate);
        return !IsZero(oa.address<51:oasize>);
else
    return FALSE;
```

2.33 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).

2.34 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 instruction that finishes execution, the profiling operation records:

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

A conditional compare operation is treated as a conditional operation.

It is IMPLEMENTATION DEFINED which conditional select operations (both integer and floating-point), including general-purpose, SIMD&FP, and SVE operations, are treated as conditional:

- Conditional select.
- Conditional select increment.
- Conditional select negate.
- Conditional select invert operations.

Predicated SVE operations are not conditional operations, as the conditionality of the operation is controlled by a predicate.

Micro-ops might have different characteristics from the architectural instructions they are created from. See Information collected for micro-ops on page D9-2959 for more information.

Note: If the implementation samples architectural instructions, for some alias instructions, the alias specifies the opposite condition to the machine instruction that they alias. For example, CSET and CSETM. The event as captured by SPE always captures the result of the machine instruction, not the alias.

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

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

is replaced with:

This field is valid only if the OpType defines this as either a Conditional Branch or a Conditional operation, and is **RAZ** otherwise.

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. See Additional information for each profiled conditional instruction on page D9-2962.

2.35 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;
```

2.36 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.

2.37 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.

2.38 D18140

In sections B2.3.8 (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.

The same change is made in section E2.3.8 (Ordering of instruction fetches).

2.39 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,
Exception_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,
Exception_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().

2.40 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.

2.41 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().

2.42 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>.

2.43 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]:

2.44 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
```

2.45 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.

2.46 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)
UNKNOWN);
else
    EDVIDSR.VMID = (if pc_sample.has_el2 && pc_sample.el IN {EL1,EL0} then
pc_sample.vmid else Zeros());
```

2.47 D18196

In section J1.3.5 (shared/translation), the function HasS2Translation() reading:

```
// HasS2Translation()
// =====
// Returns TRUE if stage 2 translation is present for the current translation regime

boolean HasS2Translation()
    return (EL2Enabled() && !IsInHost() && PSTATE.EL IN {EL0,EL1});
```

is removed from the pseudocode.

2.48 D18199

In section D7.10.3 (Common event numbers), in the subsection ‘Common microarchitectural events’, in the following event descriptions:

- 0x8134, ‘DLTB_HWUPD, Data TLB hardware update of translation table’.
- 0x8135, ‘ILTB_HWUPD, Instruction TLB hardware update of translation table’.

The following clarification is added:

Each attempted hardware update of a translation table entry is counted once. If the PE requires performs multiple translation table walk accesses to perform an update, this counts as a single update. If the update fails because it would not be atomic and has to be retried, each retry is counted.

2.49 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 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 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 event that counts events in progress in section D7.10.3 (Common event numbers), subsection ‘Common microarchitectural events’. For example:

- 0x8120, INST_FETCH_PERCYC, Event in progress, INST_FETCH.
- 0x8121, MEM_ACCESS_RD_PERCYC, Event in progress, MEM_ACCESS_RD.
- 0x8128, DTLB_WALK_PERCYC, Event in progress, DTLB_WALK.
- 0x8129, ITLB_WALK_PERCYC, Event in progress, ITLB_WALK.

2.50 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. Note: 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.

2.51 D18203

In section D7.10.3 (Common event numbers), subsection ‘Common microarchitectural events’, the entries for 0x8122 and 0x8123 are removed. Entries for 0x8122 and 0x8123 are also removed from section D7.10.5 (Meaningful ratios between common microarchitectural events), Table D7-7 ‘REFILL events and associated access events’.

2.52 D18216

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

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

in the operational pseudocode is removed.

2.53 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.

2.54 D18240

This communicated change was reverted before it appeared in *Arm® Architecture Reference Manual Armv8, for A-profile architecture, Issue G.b*.

2.55 C18241

In section D7.10.3 (Common event numbers), in the subsection 'Common microarchitectural events', the following text in the '0x8119, BR_TAKEN_MIS_PRED_RETIRE' PMU event description:

These are branch instructions where the branch was mispredicted and taken.

is clarified to read:

These are all branch instructions on the architecturally executed path, where the branch is mispredicted and taken.

2.56 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.

2.57 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.'

2.58 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 and HCR_EL2.E2H == 1:

2.59 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).

2.60 D18264

A new subsection is added to D9.6.3 (Additional information for each profiled memory access operation), in order to highlight how the SPE treats LD64B, ST64B, ST64BV, and ST64BV0 instructions:

FEAT_LS64, FEAT_LS64_V, and FEAT_LS64_ACCDATA

FEAT_LS64, FEAT_LS64_V, and FEAT_LS64_ACCDATA add the LD64B, ST64B, ST64BV, and ST64BV0 instructions. For SPE:

- LD64B is treated as a load.

- ST64B, ST64BV, and ST64BV0 are treated as stores. ST64BV and ST64BV0 are store with return instructions and might have performance properties similar to a Device memory load. Sampled ST64BV and ST64BV0 instructions might generate the following SPE data normally associated with loads:
- A Data Source packet. This is optional for these instructions. For example, if the Data Source packet can only describe Normal memory sources, then they are not relevant for these instructions.
- The total latency counter for the sampled operation preferably includes cycles to the point where the return value is returned to the PE.

Note: These behaviors are optional. The information in these packets might not be available or relevant for all implementations.

In section D7.10.1 (Definitions), in the ‘Memory-write operations’ entry, the text that reads:

DC ZVA is counted as a Memory-write operation.

is updated to read:

DC ZVA is counted as a Memory-write operation. ST64BV and ST64BV0 are Store with Return instructions, but for the purpose of the PMU they are treated as Memory-write operations.

2.61 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.

2.62 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_ETC 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)’.

2.63 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).

2.64 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.

2.65 D18288

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

```
if IsEventRegisterSet() then
    ClearEventRegister();
else
    if PSTATE.EL == EL0 then
        ...
```

are changed to read:

```
if IsEventRegisterSet() then
    ClearEventRegister();
// No further operation if the local timeout has expired.
elseif HaveFeatWfxT() && LocalTimeoutEvent(localtimeout) then
    EndOfInstruction();
else
    if PSTATE.EL == EL0 then
        ...
```

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

```
if !InterruptPending() then
    if PSTATE.EL == EL0 then
        ...
```

is updated to:

```
// No further operation if the local timeout has expired or an interrupt is pending.
if InterruptPending() || (HaveFeatWfxT() && LocalTimeoutEvent(localtimeout)) then
    EndOfInstruction();
else
    if PSTATE.EL == EL0 then
        ...
```

The function WaitForInterrupt() that reads:

```
WaitForInterrupt(integer localtimeout)
    if localtimeout < 0 then
        EnterLowPowerState();
    else
        if !LocalTimeoutEvent(localtimeout) then
            EnterLowPowerState();
    return;
```

is updated to:

```
WaitForInterrupt(integer localtimeout)
    if !(HaveFeatWfxT() && LocalTimeoutEvent(localtimeout)) then
        EnterLowPowerState();
```

Additionally, the comment for the function LocalTimeoutEvent() that reads:

```
// Returns TRUE if a local timeout event is generated when the value of
// CNTVCT_EL0 equals or exceeds the threshold value for the first time.
// If the threshold value is less than zero a local timeout event will
// not be generated.
```

is updated to:

```
// Returns TRUE if CNTVCT_EL0 equals or exceeds the localtimeout value.
```

In section C6.2.348 (WFE), in the operational pseudocode for WFE, the line that reads:

```
Hint_WFE(-1, WFxType_WFE);
```

is updated to:

```
integer localtimeout = 1 << 64;  
Hint_WFE(localtimeout, WFxType_WFE);
```

An equivalent change is made in section C6.2.350 (WFI), in the operational pseudocode for WFI, and in sections F5.1.299 (WFE) and F5.1.300 (WFI), for the A32 WFE and WFI instructions.

2.66 D18290

In section D7.10.3 (Common event numbers), in the subsection ‘Common microarchitectural events’, the following changes are made:

In the ‘0x8148, L2D_CACHE_RW, Level 2 data or unified cache demand access’ event description, the following statement is deleted:

It is **IMPLEMENTATION DEFINED** whether an access to the Level 2 data or unified cache due to a prefetch to another cache is counted by L2D_CACHE_RW or L2D_CACHE_PRFM.

Equivalent changes are made throughout the affected L<n>D_CACHE_RW, L<n>I_CACHE_RD, and CACHE_PRFM event descriptions.

In the ‘0x814A, L2D_CACHE_PRFM, Level 2 data or unified cache preload or prefetch’ event description, the text that reads:

The counter counts each access counted by L2D_CACHE that is due to a preload or prefetch instruction, or a prefetch to another cache.

is changed to read:

The counter counts each access counted by L2D_CACHE that is due to a preload or prefetch instruction.

Equivalent changes are made throughout the CACHE_PRFM event descriptions, including the clarification whether each event counts accesses due to a data preload or instruction prefetch as appropriate.

In the ‘0x8149, L2I_CACHE_RD, Level 2 instruction cache demand access’ event description, the following text is added:

This includes:

- Instruction prefetches made by the PE for Speculatively executed instructions.

- Accesses to the Level 2 instruction or unified cache due to a refill of another cache caused by a demand Instruction memory access.

Equivalent changes are made throughout the L<n>D_CACHE_RW, L<n>I_CACHE_RD and CACHE_PRFM event descriptions, for the Level 2, Level 3, or Last level caches, where the cache and fetch type are appropriate to the event.

In the '0x8156, L3D_CACHE_HWPRF, Level 3 data cache hardware prefetch' event description, the text that reads:

The counter counts each access counted by L3D_CACHE that is not counted by either L3D_CACHE_PRFM or L3D_CACHE_RW.

That is, accesses to the Level 3 data or unified cache that are not due to demand accesses, software prefetch or preload instructions, or a prefetch to another cache. For example, accesses due to a hardware prefetcher.

is changed to read:

The counter counts each access counted by L3D_CACHE that is due to a hardware prefetch. The hardware prefetch is generated by a hardware prefetcher at the Level 3 data or unified cache.

Equivalent changes are made throughout the affected L<n>D_CACHE_HWPRF and L<n>I_CACHE_HWPRF event descriptions.

2.67 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**.

2.68 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'.

2.69 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 address of the memory access is translated by the EL2 or EL2&0 translation regime.

2.70 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.

2.71 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.

2.72 D18305

In section D7.10.3 (Common event numbers), in the subsection ‘Common microarchitectural events’, the following changes are made:

In the SVE events 0x8000 to 0x80DE, where an event lists a set of instructions that generate operations that are counted, a statement equivalent to the following is added to the event description:

- In AArch32 state it is **IMPLEMENTATION DEFINED** which instructions and operations will be counted.

In the following event, the BFloat16 Advanced SIMD and SVE BFMLALB and BFMLALT instructions are added:

- 0x8028, FP_FMA_SPEC, Floating-point FMA Operations speculatively executed.

In the following events, the SVE FSUBR instruction is added:

- 0x8030, FP_ADDSUB_SPEC, floating-point add or subtract Operations speculatively executed.
- 0x8032, SVE_FP_ADDSUB_SPEC, SVE floating-point add or subtract Operations speculatively executed.

In the following events, the SVE SDIVR and UDIVR instructions are added:

- 0x8044, INT_DIV_SPEC, integer divide Operations speculatively executed.
- 0x8045, INT_DIV64_SPEC, 64-bit integer divide Operations speculatively executed.
- 0x8046, SVE_INT_DIV_SPEC, SVE integer divide Operations speculatively executed.
- 0x8047, SVE_INT_DIV64_SPEC, SVE 64-bit integer divide Operations speculatively executed.

In the following event, SMLS is added as an Advanced SIMD instruction, and SQMLAL is corrected to SQDMLAL:

- 0x8048, INT_MUL_SPEC, integer multiply Operations speculatively executed.

In the following event descriptions, 'integer multiply returning a 64-bit result' is corrected to '64x64-bit integer multiply'. The titles of the SVE_INT_MUL64_SPEC (0x804D), INT_MULH64_SPEC (0x804E), and SVE_INT_MULH64_SPEC (0x804F) events are updated for consistency.

- 0x804C, INT_MUL64_SPEC, integer 64x64 multiply Operations speculatively executed. The UMADDL and SMADDL instructions are removed from the Scalar list in this event description.
- 0x804D, SVE_INT_MUL64_SPEC, SVE integer 64-bit multiply Operations speculatively executed.

In the following events, 'REV16, REV32, REV64' is corrected to 'REVB, REVH, REVW':

- 0x8060, SVE_PERM_SPEC, Operation speculatively executed, SVE permute.
- 0x8061, SVE_PERM_IGRANULE_SPEC, Operation speculatively executed, SVE intra-granule permute.

In the following event, 'UNPKHI, UNPKLO' are corrected to 'UUNPKHI, UUNPKLO':

- 0x8062, SVE_PERM_XGRANULE_SPEC, Operation speculatively executed, SVE cross-granule permute.

In the following events, 'UQDECP (scalar), UQDECP (scalar)' is corrected to 'UQDECP (scalar), UQINCP (scalar)':

- 0x8064, SVE_XPIPE_SPEC, SVE cross-pipe Operations speculatively executed.
- 0x8065, SVE_XPIPE_Z2R_SPEC, SVE vector to scalar cross-pipe Operations speculatively executed.

In the following events, for example ‘each half-precision floating-point operation’ is corrected to ‘each floating-point operation where the largest type is half-precision’, and the phrase ‘ALU operation counts’ is corrected to ‘ALU operations’:

- 0x80C2, FP_HP_SCALE_OPS_SPEC, Scalable half-precision floating-point element ALU operation counts Speculatively executed.
- 0x80C3, FP_HP_FIXED_OPS_SPEC, Non-scalable half-precision floating-point element ALU operation counts Speculatively executed.
- 0x80C4, FP_SP_SCALE_OPS_SPEC, Scalable single-precision floating-point element ALU operation counts Speculatively executed.
- 0x80C5, FP_SP_FIXED_OPS_SPEC, Non-scalable single-precision floating-point element ALU operation counts Speculatively executed.
- 0x80C6, FP_DP_SCALE_OPS_SPEC, Scalable double-precision floating-point element ALU operation counts Speculatively executed.
- 0x80C7, FP_DP_FIXED_OPS_SPEC, Non-scalable double-precision floating-point element ALU operation counts Speculatively executed.

In the following events, references to ‘ALU operation counts’ are corrected to ‘Memory-read operations’ or ‘Memory-write operations’ as applicable:

- 0x80CA, LDST_SCALE_OPS_SPEC, Scalable load and store element ALU operation counts Speculatively executed
- 0x80CB, LDST_FIXED_OPS_SPEC, Non-scalable load and store element ALU operation counts Speculatively executed
- 0x80CC, LD_SCALE_OPS_SPEC, Scalable load element ALU operation counts Speculatively executed
- 0x80CD, LD_FIXED_OPS_SPEC, Non-scalable load element ALU operation counts Speculatively executed
- 0x80CE, ST_SCALE_OPS_SPEC, Scalable store element ALU operation counts Speculatively executed
- 0x80CF, ST_FIXED_OPS_SPEC, Non-scalable store element ALU operation counts Speculatively executed

In the following events, references to ‘ALU operation counts’ are removed:

- 0x80DA, LDST_SCALE_BYTES_SPEC, Scalable load and store bytes, Speculatively executed.
- 0x80DB, LDST_FIXED_BYTES_SPEC, Non-scalable load and store bytes, Speculatively executed.
- 0x80DC, LD_SCALE_BYTES_SPEC, Scalable load bytes, Speculatively executed.
- 0x80DD, LD_FIXED_BYTES_SPEC, Non-scalable load bytes, Speculatively executed.
- 0x80DE, ST_SCALE_BYTES_SPEC, Scalable store bytes, Speculatively executed.
- 0x80DF, ST_FIXED_BYTES_SPEC, Non-scalable store bytes, Speculatively executed.

2.73 R18319

In section D13.3.20 (MDSCR_EL1, Monitor Debug System Control Register), the following relaxation is added to the RXO, bit [27] 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.

When OSLSR_EL1.OSLK == 1, if bits [27,6] of the value written to MDSCR_EL1 are {1,0}, that is, the RXO bit is 1 and the ERR bit is 0, the PE sets EDSCR.{RXO,ERR} to **UNKNOWN** values.

A similar relaxation is added to the TXU, bit [26] field.

2.74 D18325

In section J1.1.5 (aarch64/translation), the function AArch64.S1ApplyTablePerms() reading:

```
Permissions AArch64.S1ApplyTablePerms(Permissions permissions, bits(64) descriptor,
                                       Regime regime, S1TTWParams walkparams)

    if walkparams.hpd == '1' then
        permissions.ap_table = Zeros();
        if HasUnprivileged(regime) then
            permissions.uxn_table = Zeros();
            permissions.pxn_table = Zeros();
        else
            permissions.xn_table = Zeros();

        return permissions;

    if regime == Regime_EL10 && EL2Enabled() && walkparams.nv1 == '1' then
        ...
```

is shortened to read:

```
Permissions AArch64.S1ApplyTablePerms(Permissions permissions, bits(64) descriptor,
                                       Regime regime, S1TTWParams walkparams)
    if regime == Regime_EL10 && EL2Enabled() && walkparams.nv1 == '1' then
        ...
```

In section J1.1.5 (aarch64/translation), the function AArch64.S1InitialTTWState() reading:

```
TTWState AArch64.S1InitialTTWState(S1TTWParams walkparams, bits(64) va, Regime
                                   regime,
                                   SecurityState ss)

    TTWState walkstate;
    FullAddress tablebase;
    ...
    tablebase.address = AArch64.TTBaseAddress(ttbr, walkparams.txsz,
                                              walkparams.ps, walkparams.ds,
                                              walkparams.tgx, startlevel);

    walkstate.baseaddress = tablebase;
    ...
    walkstate.memattrs = WalkMemAttrs(walkparams.sh, walkparams.irgn,
```

```
walkparams.orgn);
```

is corrected to read:

```
TTWState AArch64.S1InitialTTWState(S1TTWParams walkparams, bits(64) va, Regime
    regime,
                                SecurityState ss)
    TTWState    walkstate;
    FullAddress tablebase;
    Permissions permissions;
    ...
    tablebase.address = AArch64.TTBaseAddress(ttbr, walkparams.txsz,
                                                walkparams.ps, walkparams.ds,
                                                walkparams.tgx, startlevel);

    permissions.ap_table = Zeros();
    if HasUnprivileged(regime) then
        permissions.uxn_table = Zeros();
        permissions.pxn_table = Zeros();
    else
        permissions.xn_table = Zeros();

    walkstate.baseaddress = tablebase;
    ...
    walkstate.memattrs    = WalkMemAttrs(walkparams.sh, walkparams.irgn,
    walkparams.orgn);
    walkstate.permissions = permissions;
```

In section J1.1.5 (aarch64/translation), the function AArch64.S1NextWalkStateTable() reading:

```
TTWState AArch64.S1NextWalkStateTable(TTWState currentstate, Regime regime,
    S1TTWParams walkparams, bits(64) descriptor)
    ...
    nextstate.baseaddress = tablebase;
    nextstate.memattrs    = currentstate.memattrs;
    nextstate.permissions = AArch64.S1ApplyTablePerms(currentstate.permissions,
    descriptor, regime,
    walkparams);

    return nextstate;
```

is corrected to read:

```
TTWState AArch64.S1NextWalkStateTable(TTWState currentstate, Regime regime,
    S1TTWParams walkparams, bits(64) descriptor)
    ...
    nextstate.baseaddress = tablebase;
    nextstate.memattrs    = currentstate.memattrs;
    if walkparams.hpd == '0' then
        nextstate.permissions = AArch64.S1ApplyTablePerms(currentstate.permissions,
    descriptor,
    regime, walkparams);
    else
        nextstate.permissions = currentstate.permissions;

    return nextstate;
```

2.75 D18330

The ArmARM 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 ArmARM, 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).

2.76 R18338

In section D9.6.3 (Additional information for each profiled memory access operation), the text that reads:

If a sampled virtual address packet is not output:

- It is **IMPLEMENTATION DEFINED** whether the Translation latency Counter packet for the load or store is either not recorded, or recorded with a value of zero.
- It is **IMPLEMENTATION DEFINED** whether the bits corresponding to the access in the Events packet are recorded or always zero. If access does not occur, these bits are zero.

is relaxed to read:

If a sampled virtual address packet is not output, or the PE does not perform the access, then all of the following apply:

- It is **IMPLEMENTATION DEFINED** whether the Translation latency Counter packet for the load or store is either not recorded, or recorded with a value of zero. If the access does not occur and the address is not translated, the packet is not recorded.
- It is **IMPLEMENTATION DEFINED** whether the bits corresponding to the access in the Events packet are recorded or always zero. If the access does not occur, these bits are zero.

2.77 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.

2.78 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).

2.79 R18353

In section F6.1.39 (VCADD), the pseudocode that reads:

```
EncodingSpecificOperations();
CheckAdvSIMDEnabled();
for r = 0 to regs-1
    operand1 = D[n+r];
    operand2 = D[m+r];
    operand3 = D[d+r];
    for e = 0 to (elements DIV 2)-1
        ...
```

is changed to read:

```
EncodingSpecificOperations();
CheckAdvSIMDEnabled();
for r = 0 to regs-1
    operand1 = D[n+r];
    operand2 = D[m+r];
    for e = 0 to (elements DIV 2)-1
        ...
```

2.80 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:

The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control cache prefetch predictions 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).

2.81 D18358

In section D5.4.2 (About PSTATE.PAN), the following text:

When FEAT_PAN3 is implemented, the SCTLR_EL1.EPAN and SCTLR_EL2.EPAN bits are used to control whether the PAN bit affects instruction accesses from EL1 or EL2. SCTLR_EL2.EPAN is available when HCR_EL2.E2H is 1.

is corrected to read:

When FEAT_PAN3 is implemented, the SCTLR_EL1.EPAN and SCTLR_EL2.EPAN bits are used to control whether the effect of the PAN bit factors in stage 1 ELO instruction access permissions in addition to the stage ELO data permissions. SCTLR_EL2.EPAN is available when HCR_EL2.E2H is 1.

2.82 D18364

In section G8.2.20, (ATS1HW, Address Translate Stage 1 Hyp mode Write), the EL1 accessibility pseudocode:

```
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1HW(R[t]);
elseif PSTATE.EL == EL2 then
```

is corrected to:

```
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
```

The same change is made to the pseudocode in section G8.2.19 (ATS1HR, Address Translate Stage 1 Hyp mode Read).

2.83 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.

2.84 D18367

In section D13.2.67 (ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0), the MPAM, bits [43:40] field is updated to read:

Indicates the major version number of support for the MPAM Extension. Defined values are:

0b0000 The major version number of the MPAM extension is 0.

0b0001 The major version number of the MPAM extension is 1.

All other values are reserved.

When combined with the minor version number from ID_AA64PFR1_EL1.MPAM_frac, the combined 'major.minor' version is:

MPAM extension version	MPAM	MPAM_frac
Not implemented.	0b0000	0b0000
v0.1 is implemented.	0b0000	0b0001
v1.0 is implemented.	0b0001	0b0000
v1.1 is implemented.	0b0001	0b0001

For more information, see ARM Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for ARMv8-A.

The equivalent changes are made in section D13.2.68 (ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1), to the MPAM_frac, bits [19:16] field.

2.85 D18370

In section B2.5.2 (Alignment of data accesses), in the sub-section 'Non-atomic Load-Acquire/Store-Release instructions', in the bullet list 'If FEAT_LSE2 is implemented, then:', the following text is added:

- If all the bytes of the memory access lie within a 16-byte quantity aligned to 16 bytes and are to Normal Inner Write-Back, Outer Write-Back Cacheable memory, an unaligned access meeting all the semantics of the instruction is performed.

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

is modified to read:

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

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

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.

2.87 D18403

In section H9.3.2 (CTIAPPCLEAR, CTI Application Trigger Clear register), the 'Purpose' is updated to read:

Clears bits of the Application Trigger.

Within the same section, the text in the description of 'APPCLEAR<x>, bit [x], for x = 31 to 0' for the value 0b1 that reads:

Clear corresponding bit in CTIAPPTRIG to 0 and clear the corresponding channel event.

is changed to read:

Clear corresponding application trigger to 0 and clear the corresponding channel event.

In section H9.3.3 (CTIAPPPULSE, CTI Application Trigger Pulse register), the text in 'APPPULSE<x>, bit [x], for x = 31 to 0' that reads:

The CTIAPPPULSE operation does not affect the state of the Application Trigger register, CTIAPPTRIG.

is changed to read:

The CTIAPPPULSE operation does not affect the state of the application trigger.

In section H9.3.4 (CTIAPPSET, CTI Application Trigger Set register), the 'Purpose' is updated to read:

Sets bits of the Application Trigger.

Within the same section, the text in the description of 'APPSET<x>, bit [x], for x = 31 to 0' for the value 0b1 that reads:

Writing this sets the corresponding bit in CTIAPPTRIG to 1 and generates a channel event.

is changed to read:

Writing this sets the corresponding application trigger to 1 and generates a channel event.

2.88 D18426

In section B2.7.2 (Device memory), in the subsection ‘Early Write Acknowledgement’, the following text:

This means that a DSB barrier instruction, executed by the PE that performed the write to the No Early Write Acknowledgement Location, completes only after the write has reached its endpoint in the memory system.

is clarified to read:

This means that a DSB barrier instruction, executed by the PE that performed the write to the No Early Write Acknowledgement Location, completes only after the write has reached its endpoint in the memory system if that is required by the system architecture.

The same edit is made in section E2.8.2 (Device Memory), subsection ‘Early Write Acknowledgement’.

2.89 D18428

In section J1.3.5 (shared/translation), the code defining the type TTWState reading:

```
// TTWState
// =====
// Translation table walk state

type TTWState is (
    boolean          istable,
    integer           level,
    FullAddress       baseaddress,
    bit               contiguous,
    bit               guardedpage,
    SDFType           sdftype, // AArch32 Short-descriptor format walk only
    bits(4)           domain,  // AArch32 Short-descriptor format walk only
    MemoryAttributes memattrs,
    Permissions       permissions
)
```

is amended to read:

```
// TTWState
// =====
// Translation table walk state

type TTWState is (
    boolean          istable,
    integer           level,
    FullAddress       baseaddress,
    bit               contiguous,
    bit               nG,
    bit               guardedpage,
    SDFType           sdftype, // AArch32 Short-descriptor format walk only
    bits(4)           domain,  // AArch32 Short-descriptor format walk only
    MemoryAttributes memattrs,
    Permissions       permissions
)
```

```

        MemoryAttributes    memattrs,
        Permissions         permissions
    )

```

In section J1.2.4 (aarch32/translation), the code within the function AArch32.S1WalkLD() reading:

```

    walkstate.baseaddress = baseaddress;
    walkstate.level       = startlevel;
    walkstate.istable     = TRUE;
    walkstate.memattrs    = WalkMemAttrs(walkparams.sh, walkparams.irgn,
    walkparams.orgn);
    walkstate.permissions.ap_table = '00';
    walkstate.permissions.xn_table = '0';
    ...
    walkstate.permissions.pxn = descriptor<53>;
    walkstate.permissions.ap  = descriptor<7:6>:'1';
    walkstate.contiguous     = descriptor<52>;
    walkstate.baseaddress.address =
    ZeroExtend(descriptor<39:indexlsb>:Zeros(indexlsb));
    ...

```

is amended to read:

```

    walkstate.baseaddress = baseaddress;
    walkstate.level       = startlevel;
    walkstate.istable     = TRUE;
    // In regimes that support global and non-global translations, translation
    // table entries from lookup levels other than the final level of lookup
    // are treated as being non-global
    walkstate.nG          = if HasUnprivileged(regime) then '1' else '0';
    walkstate.memattrs    = WalkMemAttrs(walkparams.sh, walkparams.irgn,
    walkparams.orgn);
    walkstate.permissions.ap_table = '00';
    walkstate.permissions.xn_table = '0';
    ...
    walkstate.permissions.pxn = descriptor<53>;
    walkstate.permissions.ap  = descriptor<7:6>:'1';
    walkstate.contiguous     = descriptor<52>;
    if regime == Regime_EL2 then
        // All EL2 regime accesses are treated as Global
        walkstate.nG = '0';
    elsif ss == SS_Secure && walkstate.baseaddress.paspace == PAS_NonSecure then
        // When a PE is using the Long-descriptor translation table format,
        // and is in Secure state, a translation must be treated as non-global,
        // regardless of the value of the nG bit,
        // if NSTable is set to 1 at any level of the translation table walk.
        walkstate.nG = '1';
    else
        walkstate.nG = descriptor<11>;

    walkstate.baseaddress.address =
    ZeroExtend(descriptor<39:indexlsb>:Zeros(indexlsb));
    ...

```

In section J1.2.4 (aarch32/translation), the code in the function AArch32.S1WalkSD() reading:

```

    TTWState walkstate;
    walkstate.baseaddress = baseaddress;
    walkstate.memattrs    = WalkMemAttrs(s:nos, irgn, rgn);
    walkstate.level       = 1;
    walkstate.istable     = TRUE;
    ...
    walkstate.permissions.xn = xn;

```

```
walkstate.permissions.pxn = pxn;
walkstate.domain = domain;
...
```

is amended to read:

```
TTWState walkstate;
walkstate.baseaddress = baseaddress;
// In regimes that support global and non-global translations, translation
// table entries from lookup levels other than the final level of lookup
// are treated as being non-global. Translations in Short-Descriptor Format
// always support global & non-global translations.
walkstate.nG          = '1';
walkstate.memattrs    = WalkMemAttrs(s:nos, irgn, rgn);
walkstate.level       = 1;
walkstate.istable     = TRUE;
...
walkstate.permissions.xn = xn;
walkstate.permissions.pxn = pxn;
walkstate.domain = domain;
walkstate.nG      = nG;
...
```

In section J1.1.5 (aarch64/translation), the code in the function AArch64.S1InitialTTWState() reading:

```
...
walkstate.baseaddress = tablebase;
walkstate.level       = startlevel;
walkstate.istable     = TRUE;
walkstate.memattrs    = WalkMemAttrs(walkparams.sh, walkparams.irgn,
walkparams.orgn);
walkstate.permissions = permissions;
...
```

is amended to read:

```
...
walkstate.baseaddress = tablebase;
walkstate.level       = startlevel;
walkstate.istable     = TRUE;
// In regimes that support global and non-global translations, translation
// table entries from lookup levels other than the final level of lookup
// are treated as being non-global
walkstate.nG          = if HasUnprivileged(regime) then '1' else '0';
walkstate.memattrs    = WalkMemAttrs(walkparams.sh, walkparams.irgn,
walkparams.orgn);
walkstate.permissions = permissions;
...
```

In section J1.1.5 (aarch64/translation), the code in the function AArch64.S1NextWalkStateTable() reading:

```
...
nextstate.istable     = TRUE;
nextstate.level       = currentstate.level + 1;
nextstate.baseaddress = tablebase;
nextstate.memattrs    = currentstate.memattrs;
...
```

is amended to read:

```
...
nextstate.istable      = TRUE;
nextstate.nG           = currentstate.nG;
nextstate.level        = currentstate.level + 1;
nextstate.baseaddress  = tablebase;
nextstate.memattrs     = currentstate.memattrs;
...
```

In section J1.1.5 (aarch64/translation), the code in the function AArch64.S1NextWalkStateLast() reading:

```
TTWState AArch64.S1NextWalkStateLast(TTWState currentstate, Regime regime,
                                      S1TTWParams walkparams, bits(64) descriptor)
...
nextstate.permissions = AArch64.S1ApplyOutputPerms(currentstate.permissions,
descriptor,
                                                    regime, walkparams);
nextstate.contiguous  = AArch64.ContiguousBit(walkparams.tgx,
currentstate.level, descriptor);
nextstate.guardedpage = descriptor<50>;
...
```

is amended to read:

```
TTWState AArch64.S1NextWalkStateLast(TTWState currentstate, Regime regime,
SecurityState ss,
                                      S1TTWParams walkparams, bits(64) descriptor)
...
nextstate.permissions = AArch64.S1ApplyOutputPerms(currentstate.permissions,
descriptor,
                                                    regime, walkparams);
nextstate.contiguous  = AArch64.ContiguousBit(walkparams.tgx,
currentstate.level, descriptor);

if !HasUnprivileged(regime) then
    nextstate.nG = '0';
elseif ss == SS_Secure && currentstate.baseaddress.paspace == PAS_NonSecure then
    // In Secure state, a translation must be treated as non-global,
    // regardless of the value of the nG bit,
    // if NSTable is set to 1 at any level of the translation table walk
    nextstate.nG = '1';
else
    nextstate.nG = descriptor<11>;

nextstate.guardedpage = descriptor<50>;
```

2.90 D18431

In section G8.7.26 (CNTVOFF, Counter-timer Virtual Offset register), the accessibility pseudocode for the MCRR encoding that currently reads:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    UNDEFINED;
```

```
elseif PSTATE.EL == EL2 then
    CNTVOFF = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    CNTVOFF = R[t2]:R[t];
```

is updated to:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    UNDEFINED;
elseif PSTATE.EL == EL2 then
    CNTVOFF = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTVOFF = R[t2]:R[t];
```

The equivalent edit is also made for the MRRC code.

2.91 D18434

In section D7.5.3 (Prohibiting event and cycle counting), the following text is deleted, and moved to the end of section D7.5.2 (Freezing event counters):

If a direct read of PMOVSLR_EL0 returns a non-zero value for a subset of the overflow flags, which means an event counter <n> should not count, then a sequence of direct reads of PMEVCNTR<n>_EL0 ordered after the read of PMOVSLR_EL0 and before the PMOVSLR_EL0 flags are cleared to zero, will return the same value for each read, because the event counter has stopped counting.

Note: Direct reads of System registers require explicit synchronization for following direct reads of other System registers to be ordered after the first direct read.

2.92 D18443

In section H9.2.2 (DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 15) in the description of DBGBCR<n>_EL1.BT, bits [23:20], the following value description:

0b0101 As 0b0100, with linking enabled.

is changed to read:

0b0101 As 0b0100 but linked to a Context matching breakpoint.

An equivalent change is also made in G8.3.2 (DBGBCR<n>, Debug Breakpoint Control Registers, n = 0 - 15) in the description of DBGBCR<n>.BT [23:20] for both values 0b0001 and 0b0101.

2.93 D18444

In section J1.1.5 (aarch64/translation), the function AArch64.S1Translate() reading:

```
if (AArch64.VAIsOutOfRange(va, acctype, regime, walkparams) ||
    (!ispriv && walkparams.e0pd == '1')) then
    fault.statuscode = Fault_Translation;
    fault.level      = 0;
    return (fault, AddressDescriptor UNKNOWN);
```

is corrected to read:

```
if (AArch64.S1InvalidTxSZ(walkparams) ||
    (!ispriv && walkparams.e0pd == '1') ||
    AArch64.VAIsOutOfRange(va, acctype, regime, walkparams)) then
    fault.statuscode = Fault_Translation;
    fault.level      = 0;
    return (fault, AddressDescriptor UNKNOWN);
```

In section J1.1.5 (aarch64/translation), the function AArch64.S1Walk() reading:

```
if AArch64.S1InvalidTxSZ(walkparams) then
    fault.statuscode = Fault_Translation;
    fault.level      = 0;
    return (fault, AddressDescriptor UNKNOWN, TTWState UNKNOWN, bits(64)
UNKNOWN);
```

is removed.

In section J1.1.5 (aarch64/translation), the function AArch64.S2Translate() reading:

```
if AArch64.IPAIsOutOfRange(ipa.paddress.address, walkparams) then
    fault.statuscode = Fault_Translation;
    fault.level      = 0;
    return (fault, AddressDescriptor UNKNOWN);
```

is corrected to read:

```
if (AArch64.S2InvalidTxSZ(walkparams, slaarch64) ||
    AArch64.S2InvalidSL(walkparams) ||
    AArch64.S2InconsistentSL(walkparams) ||
    AArch64.IPAIsOutOfRange(ipa.paddress.address, walkparams)) then
    fault.statuscode = Fault_Translation;
    fault.level      = 0;
    return (fault, AddressDescriptor UNKNOWN);
```

In section J1.1.5 (aarch64/translation), the function AArch64.S2Walk() reading:

```
if (AArch64.S2InvalidTxSZ(walkparams, slaarch64) ||
    AArch64.S2InvalidSL(walkparams) ||
    AArch64.S2InconsistentSL(walkparams)) then
    fault.statuscode = Fault_Translation;
    fault.level      = 0;
```



```
return (fault, AddressDescriptor UNKNOWN, TTWState UNKNOWN, bits(64)
UNKNOWN);
```

is removed.

2.94 D18451

In section D1.16.1 (Wait for Event mechanism and Send event), in the subsection 'WFE wake-up events in AArch64 state', the following text:

When FEAT_WFxT or FEAT_WFxT2 is implemented, for WFIT instructions, a local timeout event caused by the virtual count threshold value, expressed in CNTVCT_ELO, being equaled or exceeded.

is corrected to read:

When FEAT_WFxT is implemented, for WFET instructions, a local timeout event caused by the virtual count threshold value, expressed in CNTVCT_ELO, being equaled or exceeded.

Similarly, in section D1.16.2 (Wait For Interrupt), in the subsection 'WFI wake-up events', the equivalent text is updated to read:

When FEAT_WFxT is implemented, for WFIT instructions, a local timeout event caused by the virtual count threshold value, expressed in CNTVCT_ELO, being equaled or exceeded.

2.95 D18454

In section G8.7.2 (CNTHCTL, Counter-timer Hyp Control register), the accessibility pseudocode for the MRC encoding that currently reads:

```
...
elseif PSTATE.EL == EL3 then
    return CNTHCTL;
```

is modified to:

```
...
elseif PSTATE.EL == EL3 then
    if SCR_NS == '0' then
        UNDEFINED;
    else
        return CNTHCTL;
```

The equivalent edit is made for the MCR code.

Equivalent changes to EL3 accessibility are made in the following sections:

- G8.7.3 (CNTHP_CTL, Counter-timer Hyp Physical Timer Control register).

- G8.7.4 (CNTHP_CVAL, Counter-timer Hyp Physical CompareValue register).
- G8.7.5 (CNTHP_TVAL, Counter-timer Hyp Physical Timer TimerValue register).

2.96 D18457

In section G8.2.65 (HCR2, Hyp Configuration Register 2), the following fields:

- TTLBIS, bit [22].
- TOCU, bit [20].
- TICAB, bit [18].
- TID4, bit [17].

Have the following text added:

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

2.97 D18459

In section D4.2.1 (Virtual address space overflow), the text that reads:

If the address calculation performed after executing an instruction overflows 0xFFFFFFFFFFFFFFFF, the program counter becomes **UNKNOWN**.

is further clarified and relaxed to read:

If the address calculation performed after executing an instruction overflows 0xFFFFFFFFFFFFFFFF, the program counter becomes **UNKNOWN**. Similarly, if the address calculation based on the program counter for the value for the link register or exception link register overflows 0xFFFFFFFFFFFFFFFF, then those registers similarly become **UNKNOWN**.

2.98 D18463

In section D13.4.7 (PMCR_ELO, Performance Monitors Control Register), in the description for DP, bit [5], the following text:

For more information see Controlling the PMU counters on page D7-2859.

is corrected to read:

For more information, see Prohibiting event and cycle counting on page D7-2861.

2.99 D18464

In section J1.1.1 (aarch64/debug), the function `AArch64.CountEvents()` does not consider the interaction between the “freeze on overflow” feature and `PMCR.DP`. The function is updated such that the cycle counter would not count if freeze-on-overflow has disabled counting of the event counters that are not reserved for EL2.

In section J1.2.1 (aarch32/debug), equivalent changes are made for `AArch32.CountEvents()`.

2.100 D18465

In section D13.2.117 (SCTLR_EL2, System Control Register (EL2)), for all of the bits that are described as having a function when `HCR_EL2.E2H==1 && HCR_EL2.TGE==1` and being **RESO** otherwise, it is clarified that these bits:

- Are **RESO** when `HCR_EL2.E2H==0`, so software should write the value 0.
- Are ignored by hardware when `HCR_EL2.E2H==1 && HCR_EL2.TGE==0`, but software doesn't have to set the value 0.
- Have their described effect when `HCR_EL2.E2H==1 && HCR_EL2.TGE==1`.

2.101 R18467

In section H6.4.1 (Powerup request mechanism if `FEAT_DoPD` is implemented), the following text:

If the powerup request mechanism is implemented, the powerup request must be a CoreSight Class 0x9 ROM table block that contains both:

is relaxed to read:

If the powerup request mechanism is implemented, Arm strongly recommends that the powerup request is a CoreSight Class 0x9 ROM table block that contains both:

2.102 D18478

In section D5.2.3 (Controlling address translation stages), in the subsection ‘Input address size’, the line that currently reads:

If `TxSZ` is programmed to a value smaller than the effective minimum value when `FEAT_LVA` is supported, then any use of the `TxSZ` value generates a stage 1 level 0 Translation fault.

is clarified to read:

If TxSZ is programmed to a value smaller than the effective minimum value when FEAT_LVA is supported, then any use of the TxSZ value when translating an address generates a stage 1 level 0 Translation fault.

2.103 D18482

In section I5.3.13 (PMCID1SR, CONTEXTIDR_EL1 Sample Register), the text in the CONTEXTIDR_EL1, bits [31:0] field that reads:

Context ID. The value of FEAT_DoPD that is associated with the most recent PMPCSR sample.
is corrected to read:

Context ID. The value of CONTEXTIDR that is associated with the most recent PMPCSR sample.
Additionally, the text within the same field that reads:

An instruction that writes to FEAT_DoPD
is corrected to read:

An instruction that writes to CONTEXTIDR

The equivalent changes are made in section H9.2.16 (EDCIDSR, External Debug Context ID Sample Register), where instances of FEAT_PCSRv8p2 are corrected to read CONTEXTIDR_EL1.

2.104 R18495

In section D5.4.12 (Hardware management of the Access flag and dirty state), subsection 'Hardware management of dirty state', the following text is deleted:

If the hardware update of dirty state mechanism is enabled and a write to memory is prevented by a Synchronous Tag Check Fault, the AP[2] and S2AP[1] bits associated with that write are permitted to be updated. For more information, see Chapter D6 Memory Tagging Extension.

Within the same section, in the subsection 'Hardware management of the Access flag', the bullet points that read:

- The PE sets the value of the Access flag to 1 in the translation table descriptor in memory, in a coherent manner, by an atomic read-modify-write of the Translation Table descriptor, if both of the following conditions are true:
 - The descriptor does not generate a Permission fault or an Alignment fault based on the memory type.
 - If the hardware update mechanism was disabled or not implemented, the access would have generated an Access flag fault. When the PE updates the Access flag in this way no Access flag fault is generated.

- It is **CONSTRAINED UNPREDICTABLE** whether the PE sets the value of the Access flag in the translation table entry in memory to 1, in a coherent manner, by an atomic read-modify-write of the Translation Table descriptor, if both of the following conditions are true:
 - The descriptor generates a Permission fault or an Alignment fault based on the memory type.
 - If the hardware update mechanism was disabled or not implemented, the access would have generated an Access flag fault.

are relaxed to read:

- The PE sets the value of the Access flag to 1 in the translation table descriptor in memory, in a coherent manner, by an atomic read-modify-write of the Translation Table descriptor, if both of the following conditions are true:
 - The descriptor does not generate a Permission fault, an Alignment fault based on the memory type or any fault which has a lower priority than a Permission fault from that stage of translation.
 - If the hardware update mechanism was disabled or not implemented, the access would have generated an Access flag fault. When the PE updates the Access flag in this way no Access flag fault is generated.
- It is **CONSTRAINED UNPREDICTABLE** whether the PE sets the value of the Access flag in the translation table entry in memory to 1, in a coherent manner, by an atomic read-modify-write of the Translation Table descriptor, if both of the following conditions are true:
 - The descriptor generates a Permission fault, an Alignment fault based on the memory type or any fault which has a lower priority than a Permission fault from that stage of translation.
 - If the hardware update mechanism was disabled or not implemented, the access would have generated an Access flag fault.

2.105 D18507

In section D9.3.3 (Initializing the sample interval counters), the text that reads:

- If PMSIRR_EL1.RND is 1 and PMSIDR_EL1.ERnd is 1:
 - PMSICR_EL1.COUNT[[31:8] is set to PMSIRR_EL1.INTERVAL.
 - PMSICR_EL1.COUNT[7:0] is set to a random or pseudorandom value in the range 0x00 to 0xFF.

is corrected to read:

- If PMSIRR_EL1.RND is 1 and PMSIDR_EL1.ERnd is 1:
 - PMSICR_EL1.COUNT[31:8] is set to PMSIRR_EL1.INTERVAL.
 - PMSICR_EL1.COUNT[7:0] is set to 0x00.

2.106 D18508

In section D5.2.7 (The VMSAv8-64 translation table format), in the subsection ‘Use of concatenated translation tables for the initial stage 2 lookup’, in Table D5-23 ‘Possible uses of concatenated translation tables for level 1 lookup, 4KB granule’, in the ‘Size’ column, the following changes are made:

The rows that currently appear as:

IPA range	Size	Required level 0 entries
IPA[38:0]	2 ³⁶	-
IPA[39:0]	2 ³⁷	2
IPA[40:0]	2 ³⁸	4
IPA[41:0]	2 ³⁹	8
IPA[42:0]	2 ⁴⁰	16

are replaced by the following rows:

IPA range	Size	Required level 0 entries
IPA[38:0]	2 ³⁹	-
IPA[39:0]	2 ⁴⁰	2
IPA[40:0]	2 ⁴¹	4
IPA[41:0]	2 ⁴²	8
IPA[42:0]	2 ⁴³	16

2.107 D18520

In section I5.8.31 (ERR<n>PFGF, Pseudo-fault Generation Feature Register, n = 0 - 65534), the text in MV, bit [12] that reads:

0b0 When an injected error is recorded, the node might update ERR<n>MISC<m>. If any syndrome is recorded by the node in ERR<n>MISC<m>, then ERR<n>STATUS.MV is set to 0b1. ERR<n>PFGCTL.MV is **RESO**.

is updated to read:

0b0 ERR<n>PFGCTL.MV not supported. When an injected error is recorded, the node might update ERR<n>MISC<m>. If any syndrome is recorded by the node in ERR<n>MISC<m>, then ERR<n>STATUS.MV is set to 0b1. If the node always sets ERR<n>.STATUS.MV to 0b1 when recording an injected error, then ERR<n>PFGCTL.MV might be RAO/WI. Otherwise, ERR<n>PFGCTL.MV is **RESO**.

Corresponding updates are made to section I5.8.30 (ERR<n>PFGCTL, Pseudo-fault Generation Control Register, n = 0 - 65534), for bit [12] ‘when the node supports this control’. Similar corrections are made for the ERR<n>PFGF.AV and ERR<n>PFGCTL.AV controls.

2.108 D18525

The text in section D1.9.1 (PE state on reset to AArch64 state) is removed, and is replaced with the following text:

Immediately after a reset, much of the PE state is architecturally **UNKNOWN**. However, some of the PE state is defined as defined for the individual registers. The state that is reset is sufficient to permit predictable initial execution at the highest Exception level, such that this execution is then capable of initialising the remaining state of the system where necessary before use.

The equivalent changes are made to section G1.17.1 (PE state on reset into AArch32 state).

2.109 D18530

In section D2.10.5 (Determining the memory location that caused a Watchpoint Exception), in the subsection 'Address recorded for Watchpoint exceptions generated by instructions other than data cache maintenance instructions', the text that reads:

The address recorded must be both:

is changed to read:

For Watchpoints exceptions generated by the DC ZVA, DC GVA, or DC GZVA instruction, the address recorded is an address accessed by the operation that triggered the watchpoint.

Otherwise, the address recorded must be both:

2.110 D18532

In the following sections:

- I5.8.25 (ERR<n>MISC0, Error Record Miscellaneous Register 0, n = 0 - 65534).
- I5.8.26 (ERR<n>MISC1, Error Record Miscellaneous Register 1, n = 0 - 65534).
- I5.8.27 (ERR<n>MISC2, Error Record Miscellaneous Register 2, n = 0 - 65534).
- I5.8.28 (ERR<n>MISC3, Error Record Miscellaneous Register 3, n = 0 - 65534).

The text that reads:

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERR<q>PFGF.MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 1. See ERR<n>PFGF.MV for more information.

is corrected to read:

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and `ERR<q>PFGF.MV` is 1, then some parts of this register are read/write when `ERR<n>STATUS.MV` is 0. See `ERR<n>PFGF.MV` for more information.

2.111 C18533

In section D5.4.6 (Access permissions for instruction execution), the Note that currently reads:

Although the execute-never controls apply to speculative fetching, on a speculative instruction fetch from an execute-never location, no Permission fault is generated unless the PE attempts to execute the instruction that would have been fetched from that location.

is clarified to read:

Although the execute-never controls apply to speculative fetching, on an attempted speculative instruction fetch from an execute-never location, no Permission fault is generated unless the PE attempts to execute the instruction that would have been fetched from that location.

2.112 C18534

In the Glossary, in the definition of 'Speculative', the text that currently reads:

Memory effects (M2) generated by load, store, or barrier instructions (LSB2) appearing in program order after load, store, or barrier instructions (LSB1) that generate memory effects (M1) where all of the following apply:

- M1 is locally-ordered-before M2.
- LSB1 has not been executed before LSB2.

is clarified to read:

Memory effects (M2) generated by load or store instructions (LS2) appearing in program order after load or store instructions (LS1) that generate memory effects (M1) where all of the following apply:

- M1 is locally-ordered-before M2.
- LS1 has not been executed before LS2.

2.113 D18538

In section C4.1.3 (Branches, Exception Generating and System instructions), sub-section 'Unconditional branch (register)', the table rows that appear as:

opc	op2	op3	Rn	op4	Instruction page	Feature
0010	11111	000010	!=11111	!=11111	Unallocated.	-
0010	11111	000010	11111	11111	RETAA, RETAB - RETAA variant	FEAT_PAuth
0010	11111	000011	!=11111	!=11111	Unallocated.	-
0010	11111	000011	11111	11111	RETAA, RETAB - RETAB variant	FEAT_PAuth

are replaced by:

opc	op2	op3	Rn	op4	Instruction page	Feature
0010	11111	000010	!=11111	!=11111	Unallocated.	-
0010	11111	000010	!=11111	11111	Unallocated.	-
0010	11111	000010	11111	!=11111	Unallocated.	-
0010	11111	000010	11111	11111	RETAA, RETAB - RETAA variant	FEAT_PAuth
0010	11111	000011	!=11111	!=11111	Unallocated.	-
0010	11111	000011	!=11111	11111	Unallocated.	-
0010	11111	000011	11111	!=11111	Unallocated.	-
0010	11111	000011	11111	11111	RETAA, RETAB - RETAB variant	FEAT_PAuth

2.114 D18551

In section D5.10.1 (General TLB maintenance requirements), the text that reads:

The architecture permits the caching of any translation table entry that has been returned from memory without a fault, provided that the entry does not, itself, cause a Translation fault, an Address size fault, or an Access Flag fault.

is expanded to read:

The architecture permits the caching of any translation table entry that has been returned from memory without a fault, provided that the entry does not, itself, cause a Translation fault, an Address size fault, or an Access Flag fault, and, for leaf entries if hardware updates of the Access Flag are enabled, does not have the Access Flag set.

2.115 D18554

In section F6.1 (Alphabetical list of Advanced SIMD and floating-point instructions), the following text is added to each floating-point load and store instruction:

Operational information If CPSR.DIT is 1, the timing of this instruction is insensitive to the value of the data being loaded or stored.

2.116 D18558

In section D5.4.4 (About PSTATE.BTYPE), in Table D5-27 'Values taken by PSTATE.BTYPE on execution of an instruction', the list in the final row that reads:

Any instruction other than BR, BRAA, BRAAZ, BRAB, BRABZ, BLR, BLRAA, BLRAAZ, BLRAB, BLRABZ, RET, RETAA, RETAB

is corrected to read:

Any instruction other than BR, BRAA, BRAAZ, BRAB, BRABZ, BLR, BLRAA, BLRAAZ, BLRAB, BLRABZ

2.117 D18563

In section I2.2.1 (Control of counter operating frequency and increment), in the subsection 'The Frequency modes table', the text that reads:

When the system timer is operating at a lower frequency than the base frequency, the increment applied at each counter update is given by:

is corrected to read:

When the system counter is operating at a lower frequency than the base frequency, the increment applied at each counter update is given by:

2.118 D18564

In section D4.4.2 (Cache identification), the text that reads:

A single Cache Size Selection Register, CSSELR_EL1, that selects the cache level and cache type of the current Cache Size Identification Register.

is clarified to read:

A single Cache Size Selection Register, CSSELR_EL1, that selects the cache level and sort of cache (Instruction, Data/Unified/Tag) of the current Cache Size Identification Register.

In sections D13.2.25 (CCSIDR2_EL1, Current Cache Size ID Register 2), D13.2.26 (CCSIDR_EL1, Current Cache Size ID Register), D13.2.33 (CSSELR_EL1, Cache Size Selection Register) and K1.2.12 (Programming the CSSELR_EL1.Level for a cache level that is not implemented), the text that reads:

If CSSELR_EL1.Level is programmed to a cache level that is not implemented:

is corrected to read:

If CSSELR_EL1.{Level, InD, TnD} is programmed to describe a cache that is not implemented:

The equivalent edits are made to section G4.4.2 (Cache identification).

2.119 D18565

In section I5.7.14 (CNTSCR, Counter Scale Register), in the definition of ScaleVal, bits[31:0], the text that reads:

When counter scaling is enabled, ScaleVal is the amount added to the counter value for every counter tick. Counter tick is defined as one period of the current operating frequency of the Generic counter.

is clarified to read:

When counter scaling is enabled, ScaleVal is the average amount added to the counter value for one period of the frequency of the Generic counter as described in the CNTFRQ register. The actual rate of update of the counter value is determined by the counter update frequency.

2.120 R18570

In section B2.9.5 (Load-Exclusive and Store-Exclusive instruction usage restrictions), the bullet that currently reads:

- Between the Load-Exclusive and the Store-Exclusive, there are no explicit memory effects, preloads, direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, or indirect branches.

is clarified and relaxed to read:

- Between the Load-Exclusive and the Store-Exclusive, there are no explicit memory effects, preloads, direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, ISB barriers, or indirect branches.

The equivalent edit is made in section E2.10.5 (Load-Exclusive and Store-Exclusive instruction usage restrictions).

2.121 C18576

In section H3.2.8 (Syndrome information on Halting Step), the entry in the bullet list:

- The instruction being stepped generated a Halting Step debug event before the instruction was executed.

is clarified to read:

- A different exception is taken before the Halting Step debug event.

2.122 C18578

In the following sections:

- G8.4.10 (PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30).
- G8.4.11 (PMEVTYPER<n>, Performance Monitors Event Type Registers, n = 0 - 30).
- G8.4.19 (PMXEVCNTR, Performance Monitors Selected Event Count Register).
- G8.4.20 (PMXEVTYPER, Performance Monitors Selected Event Type Register).

The accessibility pseudocode for accesses at EL1 using AArch32 is updated to indicate that FEAT_FGT causes a Trap exception to EL2.

2.123 D18581

In section C5.2.17 (SPSR_EL1, Saved Program Status Register (EL1)), in the definition of the M[3:0], bits [3:0] field, for 'when exception taken from AArch64 state', the text that reads:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1 and copied to PSTATE.EL on executing an exception return operation in EL1.

is corrected to read:

- M[3:2] : On an exception to EL1:
 - If the effective values of HCR_EL2.{NV, NV1} != {1,0} or the exception is not taken from EL1, then M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
 - If the effective values of HCR_EL2.{NV, NV1} == {1,0} and the exception is not taken from EL1, then M[3:2] is set to 10.
 - M[3:2] is copied to PSTATE.EL on executing a legal exception return operation in EL1.

Additionally, in the following sections:

- C5.2.17 (SPSR_EL1, Saved Program Status Register (EL1)).
- C5.2.18 (SPSR_EL1, Saved Program Status Register (EL2)).
- C5.2.19 (SPSR_EL1, Saved Program Status Register (EL3)).

The statements which concern values being copied to PSTATE on an exception return are enhanced to describe behavior on an illegal exception return.

2.124 D18582

In section D13.2.83 (ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4), the text in the accessibility pseudocode that reads:

```
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!(IsZero(ID_MMFR4_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3"))
    && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

is corrected to read:

```
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || (!(IsZero(ID_MMFR4_EL1) ||
boolean
IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3")) && HCR_EL2.TID3 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

Equivalent corrections are made in the following sections:

- D13.2.63 (ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2).
- D13.2.66 (ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2).
- D13.2.71 (ID_DFR1_EL1, AArch32 Debug Feature Register 1).
- D13.2.78 (ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6).
- D13.2.83 (ID_MMFR5_EL1, AArch32 Memory Model Feature Register 5).
- G8.2.84 (ID_DFR1, Debug Feature Register 1).
- G8.2.91 (ID_ISAR6, Instruction Set Attribute Register 6).
- G8.2.96 (ID_MMFR4, Memory Model Feature Register 4).
- G8.2.97 (ID_MMFR5, Memory Model Feature Register 5).

2.125 D18593

In section D1.13.4 (Prioritization and recognition of interrupts), the text that reads:

Any interrupt that is pending before a Context synchronization event in the following list, is taken before the first instruction after the context synchronizing event, provided that the pending interrupt is not masked:

is clarified to read:

Any interrupt that is pending before a Context synchronization event in the following list, is taken before the first instruction after the context synchronizing event, provided that the interrupt is not masked and remains pending:

2.126 D18595

In section D7.10.3 (Common event numbers), in the subsection 'Common microarchitectural events', the text in the description of the event '0x0001, L1I_CACHE_REFILL, Level 1 instruction cache refill' that reads:

The counter counts each access counted by L1I_CACHE that causes a refill of any of the Level 1 caches outside the Level 1 caches of this PE.

is changed to read:

The counter counts each access counted by L1I_CACHE that causes a refill of the Level 1 instruction or unified cache from outside of the Level 1 instruction or unified cache.

Additionally, the text in the description of the event '0x0028, L2I_CACHE_REFILL, Attributable Level 2 instruction cache refill' that reads:

The counter counts each access counted by L2I_CACHE that causes a refill of any of the Level 1 or 2 caches outside the Level 1 or 2 caches of this PE.

is changed to read:

The counter counts each access counted by L2I_CACHE that causes a refill of the Level 2 instruction or unified cache, or any Level 1 data, instruction, or unified cache of this PE, from outside of those caches.

Equivalent changes are made to other cache events throughout the section.

2.127 C18597

In section C5.5.59 (TLBI VAE2IS, TLBI VAE2ISNXS, TLB Invalidate by VA, EL2, Inner Shareable), in 'Purpose', the bullet that reads:

- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.

is clarified to read:

- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the current value of the HCR_EL2.E2H bit, for the Security state.

The equivalent edit is made in the following sections:

- C5.5.58 (TLBI VAE2, TLBI VAE2NXS, TLB Invalidate by VA, EL2).
- C5.5.60 (TLBI VAE2OS, TLBI VAE2OSNXS, TLB Invalidate by VA, EL2, Outer Shareable).
- C5.5.67 (TLBI VALE2, TLBI VALE2NXS, TLB Invalidate by VA, Last level, EL2).
- C5.5.68 (TLBI VALE2IS, TLBI VALE2ISNXS, TLB Invalidate by VA, Last level, EL2, Inner Shareable).
- C5.5.69 (TLBI VALE2OS, TLBI VALE2OSNXS, TLB Invalidate by VA, Last level, EL2, Outer Shareable).
- C5.5.34 (TLBI RVAE2, TLBI RVAE2NXS, TLB Range Invalidate by VA, EL2).
- C5.5.35 (TLBI RVAE2IS, TLBI RVAE2ISNXS, TLB Range Invalidate by VA, EL2, Inner Shareable).
- C5.5.36 (TLBI RVAE2OS, TLBI RVAE2OSNXS, TLB Range Invalidate by VA, EL2, Outer Shareable).
- C5.5.43 (TLBI RVALE2, TLBI RVALE2NXS, TLB Range Invalidate by VA, Last level, EL2).
- C5.5.44 (TLBI RVALE2IS, TLBI RVALE2ISNXS, TLB Range Invalidate by VA, Last level, EL2, Inner Shareable).
- C5.5.45 (TLBI RVALE2OS, TLBI RVALE2OSNXS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable).

2.128 D18601

In section D4.7.2 (Basic memory access), the text that reads:

The PhysMemRead() and PhysMemRead() functions

is corrected to read:

The PhysMemRead() and PhysMemWrite() functions

2.129 E18617

In section C6.2.249 (ST64BV), the code for instruction ST64BV that reads:

```
if !HaveFeatLS64() then UNDEFINED;
```

Is corrected to read:

```
if !HaveFeatLS64_V() then UNDEFINED;
```

In section C6.2.250 (ST64BV0), the code for instruction ST64BV0 that reads:

```
if !HaveFeatLS64() then UNDEFINED;
```

Is corrected to read:

```
if !HaveFeatLS64_ACCDATA() then UNDEFINED;
```

The following functions are added in section J1.3.3 (shared/functions):

```
// HaveFeatLS64_V()
// =====
// Returns TRUE if the ST64BV instruction is
// supported, and FALSE otherwise.

boolean HaveFeatLS64_V()
    return (HasArchVersion(ARMv8p7) && HaveFeatLS64() &&
        boolean IMPLEMENTATION_DEFINED "Has Store 64-Byte with return
        instruction support");

// HaveFeatLS64_ACCDATA()
// =====
// Returns TRUE if the ST64BV0 instruction is
// supported, and FALSE otherwise.

boolean HaveFeatLS64_ACCDATA()
    return (HasArchVersion(ARMv8p7) && HaveFeatLS64_V() &&
        boolean IMPLEMENTATION_DEFINED "Has Store 64-Byte EL0 with return
        instruction support");
```

2.130 D18620

In section B2.5.2 (Alignment of data accesses), the text that reads:

If FEAT_LSE2 is implemented:

- If all the bytes of the memory access lie within a 16-byte quantity aligned to 16 bytes and are to Normal Inner Write-Back, Outer Write-Back Cacheable memory, the memory access is single-copy atomic. For a Load-Pair or Store-Pair, including load non-temporal pair, instructions the entire memory access will be single-copy atomic.

is corrected to read:

If FEAT_LSE2 is implemented:

- If all the bytes of the memory access lie within a 16-byte quantity aligned to 16 bytes and are to Normal Inner Write-Back, Outer Write-Back Cacheable memory, the memory access is single-copy atomic. For LDP, LDNP or STP instructions, the entire memory access will be single-copy atomic.

2.131 D18628

In section J1.3.3 (shared/functions), the EL2Enabled() function that reads:

```
boolean EL2Enabled()  
    return HaveEL(EL2) && (!HaveEL(EL3) || SCR_EL3.NS == '1' ||  
        IsSecureEL2Enabled());
```

is modified to:

```
boolean EL2Enabled()  
    return HaveEL(EL2) && (!HaveEL(EL3) || SCR_GEN[].NS == '1' ||  
        IsSecureEL2Enabled());
```

2.132 D18629

In section C5.2.3 (DIT, Data Independent Timing), “RET” is removed from the list of data processing instructions.

In section C6.2.220 (RET), the following text is deleted:

If PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

2.133 D18637

In section D13.2.63 (ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2), the description in the RPRES field that reads:

When FPCR.AH is 1, indicates support for 12 bits of mantissa in reciprocal and reciprocal square root instructions in AArch64 state. Defined values are:

is replaced by:

Indicates support for 12 bits of mantissa in reciprocal and reciprocal square root instructions in AArch64 state, when FPCR.AH is 1. Defined values are:

and the values:

0b0000 Reciprocal and reciprocal square root estimates give 8 bits of mantissa.

0b0001 Reciprocal and reciprocal square root estimates give 12 bits of mantissa.

are replaced by:

0b0000 Reciprocal and reciprocal square root estimates give 8 bits of mantissa, when FPCR.AH is 1.

0b0001 Reciprocal and reciprocal square root estimates give 12 bits of mantissa, when FPCR.AH is 1.

2.134 R18641

In section H9.3.19 (CTIDEVID, CTI Device ID register 0), in the NUMTRIG, bits [13:8] definition, the text that reads:

Number of triggers implemented. IMPLEMENTATION DEFINED. This is one more than the index of the largest trigger, rather than the actual number of triggers.

is updated to read:

Upper bound for number of triggers. The indices of all implemented input and output triggers are less than this value. The value of this field is IMPLEMENTATION DEFINED.

and the text that reads:

There is no guarantee that any of the implemented triggers, including the highest numbered, are connected to any components.

is updated to read:

There is no guarantee that all of the input and output triggers, including the highest numbered, are connected to any components, or that the implementation of input and output triggers is symmetrical.

In section H9.3.24 (CTIINEN<n>, CTI Input Trigger to Output Channel Enable registers, n = 0 - 31), the text in 'Configurations' that reads:

If input trigger n is not connected, the behavior of CTIINEN<n> is IMPLEMENTATION DEFINED.
is changed to read:

If input trigger n is not connected, CTIINEN<n> is **RES0**.

Equivalent changes are made in other CTI registers, such that the behavior of unimplemented trigger controls is changed from IMPLEMENTATION DEFINED to **RES0**.

2.135 E18645

In section D11.2.1 (The physical counter), after the text that currently reads:

Neither view of the physical counter ensures that:

- Context changes occurring in program order before the read of the counter have been synchronized.
- Accesses to memory appearing in program order after the read of the counter are executed before the counter has been read.

the following text is added:

Where there is a dependency through registers dependency from the read of the physical counter to a Register effect generated by a read or write, the read or write will be executed after the read of the counter.

The equivalent change is made in section D11.2.2 (The virtual counter).

Also in section D11.2.1 (The physical counter), example D11-2 'Ensuring reads of the physical counter occur after a previous memory access' is changed to read:

To ensure that all previous memory accesses have completed and all previous context changes have been synchronized before the read of the counter, one of the following sequences should be used:

either:

DSB ISB MRS Xn, CNTPCT{SS}_ELO ; either view of the physical counter has the same effect in this example

or

DMB LDR Xa, [Xd] ; this could be any memory location, for example the stack pointer CBZ Xa, next

next ISB ; this ISB is not needed if the MRS is accessing CNTPCTSS_ELO MRS Xn,CNTPCT{SS}_ELO

To ensure that a memory access only occurs after a read of the counter, then either of the following sequences should be used:

MRS Xn, CNTPCT{SS}_ELO ; either view of the physical counter has the same effect in this example ISB LDR Xa, [Xb] ; this load will be executed after the counter has been read

or MRS Xn, CNTPCT{SS}_ELO ; either view of the physical counter has the same effect in this example EOR Xm, Xn, Xn ; LDR Xa, [Xb, Xm] ; this load will be executed after the counter has been read

In section D11.2.2 (The virtual counter), the equivalent change is made to Example D11-4 'Ensuring reads of the virtual counter occur after a previous memory access'.

2.136 D18647

In D13.2.115 (SCR_EL3, Secure Configuration Register), the following text is added to the 'Otherwise' case of SIF, bit [9]:

When FEAT_PAN3 is implemented, it is **IMPLEMENTATION DEFINED** whether SCR_EL3.SIF is also used to determine instruction access permission for the purpose of PAN.

2.137 D18650

In D13.2.42 (FAR_EL3, Fault Address Register (EL3)), the text that reads:

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state'.

is corrected to read:

For a Data Abort exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state'.

2.138 R18653

In section A2.2.1 (Additional functionality added to Armv8.0 in later releases) for the entry associated with FEAT_SSBS, FEAT_SBSS2, Speculative Store Bypass Safe, the line that reads:

This feature is OPTIONAL in Armv8.0 implementations and mandatory in Armv8.5 implementations.

is relaxed to read:

This feature is OPTIONAL in Armv8.0 implementations.

2.139 D18672

In the following sections:

- D1.14 (Configurable instruction enables and disables, and trap controls).
- D1.16 (Mechanisms for entering a low-power state).

Incorrect instances of 'Wait for Event with Interrupt' are corrected to 'Wait for Interrupt with Timeout'.

2.140 C18676

In section I1.1 (Supported access sizes), the text that reads:

If any PE in the system implements AArch32, word-aligned 32-bit accesses to either half of a 64-bit register that is mapped to a doubleword-aligned pair of adjacent 32-bit locations.

is updated to read:

If the system includes any component with direct memory access to the memory-mapped component which needs to access the component but does not support making 64-bit accesses, word-aligned 32-bit accesses to either half of a 64-bit register that is mapped to a doubleword-aligned pair of adjacent 32-bit locations. This includes, but is not limited to:

- A PE that supports AArch32 at any Exception level.
- A PE that implements a 32-bit ISA that does not include 64-bit memory operations. For example: Armv8-M T32.
- A Debug Access Port that cannot make 64-bit accesses.

Arm deprecates support for 32-bit accesses to either half of 64-bit registers.

Note: Although AArch32 implementations might make 64-bit accesses for LDP and STP instructions, this is not architecturally required. To guarantee ordering of accesses, portable

AArch32 software should use LDR and STR. For compatibility with such software, the memory-mapped component should treat the PE that supports AArch32 as not making 64-bit accesses using portable code.

2.141 D18677

In section F5.1.98 (LDRT), the Operation for all encodings for the LDRT instruction that reads:

```
if ConditionPassed() then
    if PSTATE.EL == EL2 then UNPREDICTABLE; // Hyp mode
    EncodingSpecificOperations();
```

is modified to read:

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNPREDICTABLE; // Hyp mode
```

An equivalent change is made in the following sections:

- F5.1.78 (LDRBT).
- F5.1.89 (LDRHT).
- F5.1.93 (LDRSBT).
- F5.1.97 (LDRSHT).
- F5.1.243 (STRT).
- F5.1.233 (STRBT).
- F5.1.242 (STRHT).

2.142 C18680

In section B2.2.1 (Requirements for single-copy atomicity), in the subsection ‘Changes to single-copy atomicity in Armv8.4’, all of the lines that begin:

Otherwise, it is **IMPLEMENTATION DEFINED** whether

are deleted, and the following text is added at the beginning of the subsection:

In addition to the single-copy atomicity requirements above:

Additionally, each case described in the subsection is clarified to indicate that all of the listed conditions must be true.

2.143 D18683

In section J1.3.3 (shared/functions), the function GenMPAMcurEL() that reads:

```
MPAMInfo GenMPAMcurEL(AccType acctype)
...
    if UsingAArch32() then
        (validEL, mpamEL) = ELFromM32(PSTATE.M);
    else
        mpamEL = PSTATE.EL;
        validEL = TRUE;
```

is changed to:

```
MPAMInfo GenMPAMcurEL(AccType acctype)
...
    if UsingAArch32() then
        (validEL, mpamEL) = ELFromM32(PSTATE.M);
    else
        mpamEL = if acctype == AccType_NV2REGISTER then EL2 else PSTATE.EL;
        validEL = TRUE;
```

2.144 D18685

In section H6.3 (Core power domain power states), the text in the description of the ‘Standby’ state that reads:

In a typical implementation, the PE enters standby by executing a WFI or WFE instruction, and exits on a wake-up event.

and the text in the description of the ‘Retention’ state that reads:

The OS takes some measures, including **IMPLEMENTATION DEFINED** code sequences and registers, to reduce energy consumption.

and the text in the description of the ‘Powerdown’ state that reads:

The OS takes some measures to reduce energy consumption by turning the Core power domain off.

are replaced by the following text:

Mechanisms for entering a low-power state on page D1-2536 describes the architectural mechanisms for entering low-power states.

Also in the description of the ‘Standby’ state, the text that reads:

- An External Debug Request debug event is a wake-up event when halting is allowed. This means that the PE must exit standby to handle the debug event. If the PE executed a WFE or a WFI instruction to enter standby, then it retires that instruction.

and the text in the description of the ‘Retention’ state that reads:

External Debug Request debug events stay pending and registers in the Core power domain cannot be accessed.

are removed, and the following text is added:

When the PE enters a low-power state other than Powerdown by executing a WFI, WFIT, WFE, or WFET instruction, it remains in that low-power state until it receives a *wake-up event*. See Mechanisms for entering a low-power state on page D1-2536 for the definition of wake-up events. When halting is allowed, an External Debug Request is a wake-up event. In addition, if the PE enters the Standby state for any reason, it will leave that state to service an External Debug Request debug event.

Additionally, in section H3.5.1 (Synchronization and External Debug Request debug events), the text that reads:

Otherwise, when halting is allowed, External Debug Request debug events must be taken in finite time, without requiring the synchronization of any necessary change to the external authentication interface.

is clarified to read:

Otherwise, when all of the following are true, External Debug Request debug events must be taken in finite time, without requiring the synchronization of any necessary change to the external authentication interface:

- Halting is allowed.
- The PE is not in a low-power state that is not required to exit on an External Debug Request. See Core power domain power states on page H6-7440.

External Debug Request is a wake-up event for WFI, WFIT, WFE, or WFET instructions. See Mechanisms for entering a low-power state on page D1-2536.

2.145 D18698

In section D9.6.2 (Additional information for each profiled branch or exception return), the text that currently reads:

If the optional behavior in FEAT_SPEv1p2 is implemented, the target address of the most recently executed sampled branch that was taken and retired in program order before the sampled operation.

is corrected to read:

If the optional behavior in FEAT_SPEv1p2 is implemented, the target address of the most recently executed branch that was taken and retired in program order before the sampled operation.

In the same section, the subsection ‘Last Branch Target’ is renamed to ‘Previous Branch Target’, and the text in this subsection that reads:

If FEAT_SPEv1p2 is implemented, PMSIDR_EL1.PBT optionally adds the capability to record a packet for each event that provides the target address of the previous taken branch.

If enabled, the profiling operation records the target address of the most recently sampled branch that was taken and retired in program order before the sampled operation.

is corrected to read:

FEAT_SPEv1p2 adds an optional capability to record a packet for each event that provides the target address of the previous taken branch. PMSIDR_EL1.PBT describes whether this feature is implemented.

When implemented, the profiling operation records the target address of the most recent branch that was taken and retired in program order before the sampled operation.

2.146 C18700

In section D2.4 (Enabling debug exceptions from the current Exception level), Table D2-6 ‘Whether debug exceptions are enabled from the current Exception level’ is replaced with the following text:

Debug exceptions are enabled from the current Exception level, where EL_D is the Exception level that Table D2-3 on page D2-2550 defines, as follows:

When executing at any Exception level that is higher than EL_D :

- Breakpoint instruction exceptions are enabled.
- All other debug exceptions are disabled.

When executing at EL_D :

- Breakpoint instruction exceptions are enabled.
- All other debug exceptions are disabled if either of the following is true:
 - The Local (kernel) Debug Enable bit, MDSCR_EL1.KDE, is 0.
 - The Debug exception mask bit, PSTATE.D, is 1.Otherwise enabled. This means that a debugger must explicitly enable these debug exceptions from EL_D by setting MDSCR_EL1.KDE to 1 and PSTATE.D to 0.

When executing at EL1, EL_D is EL2, and FEAT_NV2 is implemented:

- Watchpoint debug exceptions generated by a System register access converted to a memory access because HCR_EL2.NV2 is 1 are disabled if MDSCR_EL1.KDE is 0 and enabled if MDSCR_EL1.KDE is 1. The value of PSTATE.D is ignored.
- All other debug exceptions are enabled.

Otherwise:

- All debug exceptions are enabled.

Additionally, in section D2.3 (Routing debug exceptions), the text that reads:

The routing of debug exceptions is as follows:

is moved to after Table D2-2, 'Whether debug exceptions are enabled from the current Security state'.

2.147 D18703

In section J1.3.1 (shared/debug), the code in function ExternalDebugInterruptsDisabled() that reads as:

```
if Havev8p4Debug() then
    if target == EL3 || IsSecure() then
        int_dis = (EDSCR.INTdis[0] == '1' &&
ExternalSecureInvasiveDebugEnabled());
    else
        int_dis = (EDSCR.INTdis[0] == '1');
```

is updated to read as:

```
SecurityState ss = SecurityStateAtEL(target);
if Havev8p4Debug() then
    if EDSCR.INTdis[0] == '1' then
        case ss of
            when SS_NonSecure int_dis = ExternalInvasiveDebugEnabled();
            when SS_Secure    int_dis = ExternalSecureInvasiveDebugEnabled();
    else
        int_dis = FALSE;
```

2.148 C18704

In section D1.14 (Configurable instruction enables and disables, and trap controls), after the text that reads:

- The instruction is not UNPREDICTABLE or **CONSTRAINED UNPREDICTABLE** in the PE state it is executed in. UNPREDICTABLE and **CONSTRAINED UNPREDICTABLE** instructions can generate exceptions as a result of these controls, but the architecture does not require them to do so.

A note is added that reads:

Note: Many **CONSTRAINED UNPREDICTABLE** behaviors for instructions include an allowance that the **CONSTRAINED UNPREDICTABLE** instruction behaves the same way as a closely related instruction that is not **CONSTRAINED UNPREDICTABLE**. In those cases, the instruction enable, disable or trap

control that causes in exception on the closely related instruction will cause the same exception on the **CONSTRAINED UNPREDICTABLE** instruction.

2.149 D18711

In section J1.3.3 (Shared pseudocode), the code in HaveFGTExt() that reads as:

```
boolean HaveFGTExt()
return HasArchVersion(ARMv8p6) && !ELUsingAArch32(EL2);
```

is updated to read as:

```
boolean HaveFGTExt()
return HasArchVersion(ARMv8p6);
```

2.150 D18712

In section H8.6 (External debug interface registers), in Table H8-2 'External debug interface register map', the following rows:

Offset	Mnemonic	Register, or additional information
0xFC0	EDDEVID2	EDDEVID, External Debug Device ID register 0 on page H9-7528
0xFC4	EDDEVID1	EDDEVID1, External Debug Device ID register 1 on page H9-7530
0xFC8	EDDEVID	EDDEVID2, External Debug Device ID register 2 on page H9-7531

are changed to read:

Offset	Mnemonic	Register, or additional information
0xFC0	EDDEVID2	EDDEVID2, External Debug Device ID register 2 on page H9-7531
0xFC4	EDDEVID1	EDDEVID1, External Debug Device ID register 1 on page H9-7530
0xFC8	EDDEVID	EDDEVID, External Debug Device ID register 0 on page H9-7528

2.151 E18715

In section C5.2.3 (DIT, Data Independent Timing), in the description of DIT, bit [24], the instructions SQRDMULH, SQDMULH, and SQRDMLAH are added to the bullet list that begins 'A subset of those instructions which use the SIMD&FP register file. These instructions are:'.

Additionally, in the following sections:

- C7.2.288 (SQDMULH (by element)).
- C7.2.289 (SQDMULH (vector)).

- C7.2.293 (SQRDMLAH (by element)).
- C7.2.294 (SQRDMLAH (vector)).
- C7.2.297 (SQRDMULH (by element)).
- C7.2.298 (SQRDMULH (vector)).

The following is added to the instruction descriptions:

Operational information

If PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

2.152 C18720

In section D9.8.3 (Faults and watchpoints), an entry is added to Table D9-5 'Faults' for 'External abort on translation table walk or translation table update', with the Conditions:

The translation of a virtual address to a physical address might generate an External abort on the translation table walk or translation table update.

In the same section, following the list starting 'If a write to the Profiling Buffer generates a fault and PMBSR_EL1.S is 0, then a Profiling Buffer management event is generated:', the following text is added:

If a write to the Profiling Buffer generates an External abort on a translation table walk or translation table update, it is **IMPLEMENTATION DEFINED** whether PMBSR_EL1.EA is set to 1 or unchanged.

In section D9.8.4 (External aborts), following the list of options starting 'It is an **IMPLEMENTATION DEFINED** choice whether such an External abort:', the following text is added:

The choice is not required to be the same for each of an External abort on the write to the Profiling Buffer, an External abort on a translation table walk, and an External abort on a translation table update.

An External abort on a translation table walk or translation table update might be treated as a synchronous MMU fault, as described by Faults and watchpoints on page D9-2974.

2.153 R18721

In section D13.2.52 (HFGITR_EL2, Hypervisor Fine-Grained Instruction Trap Register), the following changes are made:

In the definitions of DCCVAU, bit [7], ICIVAU, bit [2], ICIALLU, bit [1], and ICIALLUIS, bit [0], the following text is added:

If the Point of Unification is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

In the definitions of DCCVAC, bit [54], DCCIVAC, bit [10], and DCIVAC, bit [3], the following text is added:

If the Point of Coherency is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

In the definition of DCCVAP, bit [8], the following text is added:

If the Point of Persistence is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

In the definition of DCCVADP, bit [9], the following text is added:

If the Point of Deep Persistence is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

2.154 D18735

In section D5.9.1 (Use of ASIDs and VMIDs to reduce TLB maintenance requirements), the following text is deleted:

For a symmetric multiprocessor cluster where a single operating system is running on the set of processing elements, the Arm architecture requires all ASID values to be assigned uniquely within any single Inner Shareable domain. In other words, each ASID value must have the same meaning to all processing elements in the system.

2.155 D18739

In section J1.1.1 (aarch64/debug), in the CollectTimeStamp() function, each case of the following code:

```
return TimeStamp_Virtual;
```

is changed to read:

```
return if IsInHost() then TimeStamp_Physical else TimeStamp_Virtual;
```

2.156 R18746

In section B2.7.2 (Device memory), in the subsection ‘Multi-register loads and stores that access Device memory’, the following paragraph is added:

The architecture permits that the non-speculative execution of an instruction that loads or stores more than one general-purpose or floating-point & SIMD register might result in repeated accesses to the same address.

The equivalent edit is made in section E2.8.2 (Device Memory), in the subsection ‘Multi-register loads and stores that access Device memory’.

2.157 D18781

In section B2.2.5 (Concurrent modification and execution of instructions), the following bullet item is added to the Note in point 2:

- In a multiprocessor system, the DC CVAU and IC IVAU are broadcast to all PEs within the Inner Shareable domain of the PE running this sequence.

In the same section, point 3 is changed to read:

When the modified instructions are observable, each PE that is executing the modified instructions must issue execute an ISB or perform a context synchronizing event to ensure execution of the modified instructions: