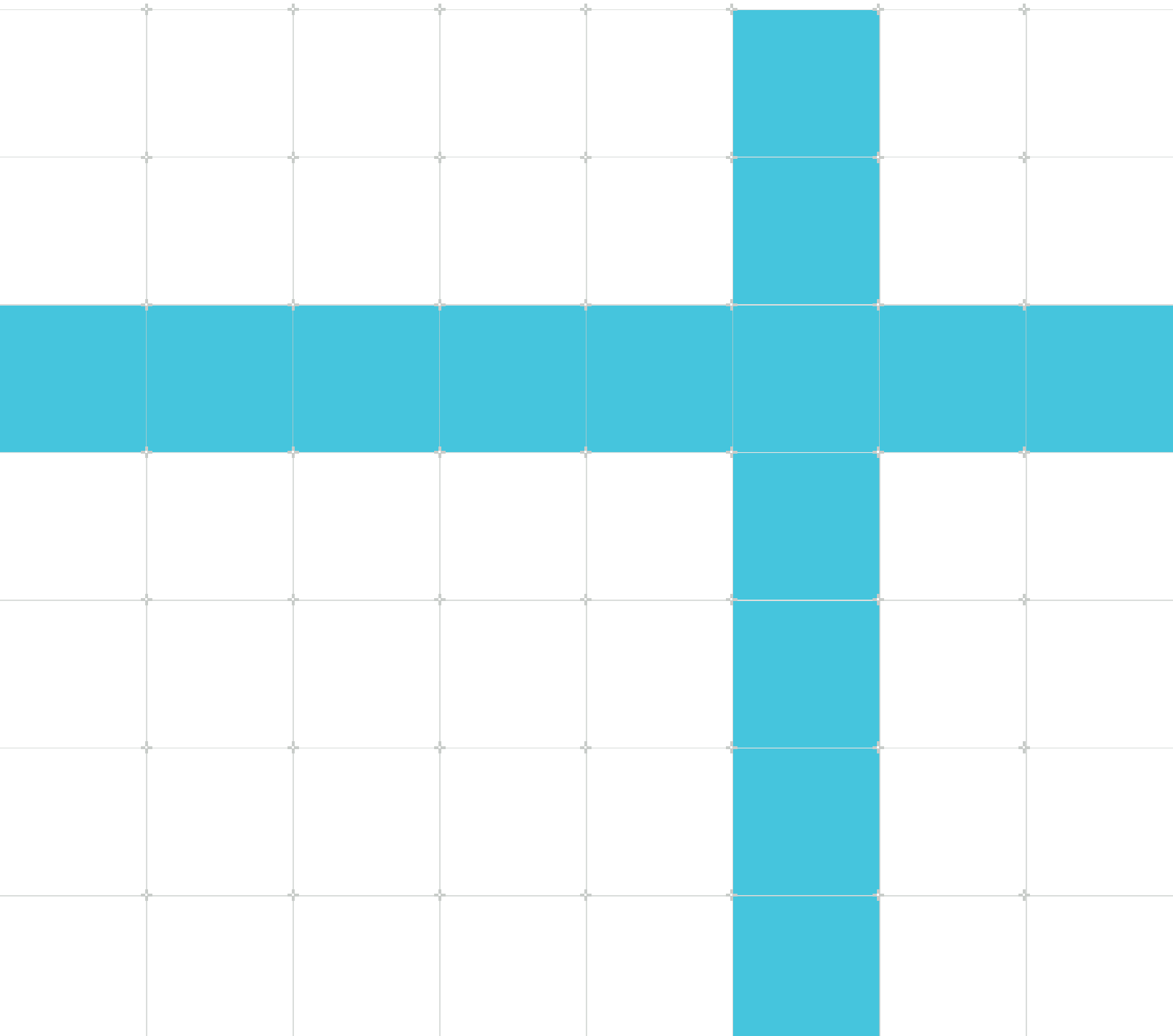
# Arm® Architecture Reference Manual for A-profile architecture

## Known issues in Issue I.a



**Issue 00**
102105_I.a_00_en



# Arm® Architecture Reference Manual for A-profile architecture

## Known issues in Issue I.a



## 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.b-05 | 31 January 2022 | Non-Confidential | Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 7 January 2022 |
| H.a-06 | 22 July 2022 | Non-Confidential | Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 22 July 2022 |
| I.a-00 | 5 August 2022 | Non-Confidential | Known Issues in Arm® Architecture Reference Manual, Issue I.a, as of 5 August 2022 |

## Proprietary Notice

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.

Unrestricted Access is an Arm internal classification.

## Product Status

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

## 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

## 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 |
|---|---|
| *italic* | Citations. |
| **bold** | Interface elements, such as menu names.<br><br>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 underline` | 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.<br><br>For example:<br><br>`MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>` |
| SMALL CAPITALS | Terms that have specific technical meanings as defined in the *Arm® Glossary*. 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. |
| Note | An important piece of information that needs your attention. |

| Convention | Use |
|---|---|
|   | A useful tip that might make it easier, better or faster to perform a task. |
|   | A reminder of something important that relates to the information you are reading. |

## 1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

| Arm product resources | Document ID | Confidentiality |
|---|---|---|
| *Arm® Architecture Reference Manual for A-profile architecture, Issue I.a* | DDI 0487I.a | Non-Confidential |


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 for A-profile architecture (DDI 0487), Issue I.a.

Key

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

## 2.1 C15788

In section D8.13.5 (TLB maintenance instructions), in the subsection 'TLB maintenance instructions that apply to a range of addresses', the following text is added:

> It is possible for a TLB range maintenance instruction for a translation regime that supports two VA ranges to be be issued with an address in the TTBR1 half of the virtual address space, and SCALE and NUM values such that the range exceeds the top of the address space. In this scenario, the address is not considered to wrap on overflow and the PE is not required to invalidate any entries inserted for the TTBR0 half of the VA space.

## 2.2 C16212

In section D17.2.156 (VSTTBR_EL2, Virtualization Secure Translation Table Base Register) and D17.2.158 (VTTBR_EL2, Virtualization Translation Table Base Register), in the 'BADDR, bits [47:1]' field, the references to:

> stage 1 translation table base

are corrected to read:

> stage 2 translation table base

## 2.3 D17015

Details of traps will be added through the use of new LDC and STC accessibility pseudocode in sections G8.3.17 (DBGDTRRXint, Debug Data Transfer Register, Receive) and G8.3.19 (DBGDTRTXint, Debug Data Transfer Register, Transmit). 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.

## 2.4 D17119

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

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

## 2.5 R17661

In section D9.2 (Allocation Tags), the following Notes are removed:

> Note: The value `0b1111` may incur a higher performance overhead than other Allocation Tag encodings.
>
> Note: Arm recommends that software does not use instructions which write `0b1111` as an Allocation Tag to memory.

## 2.6 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.

## 2.7 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).

## 2.8 D18465

In section D17.2.119 (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 **RES0** otherwise, it is clarified that these bits:

- Are **RES0** 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.

## 2.9 R18485

In section I5.8.8 (ERRDEVAFF, Device Affinity Register), the following text is added to the end of the Purpose section:

> Depending on the **IMPLEMENTATION DEFINED** nature of the system, it might be possible that ERRDEVAFF is read before system firmware has configured the group of error records and/or the PE or group of PEs that the group of error records has affinity with. When this is the case, ERRDEVAFF reads as zero.

## 2.10 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 **RES0**.

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 **RES0**.

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.

## 2.11 D18736

In section I5.8.5 (ERRCRIC0, Critical Error Interrupt Configuration Register 0), under the heading 'Accessing the ERRCRICR0', the following text is added:

> If the implementation does not use the recommended layout for the ERRIRQCR<n> registers, accesses to ERRCRICR0 are IMPLEMENTATION DEFINED.
>
> ERRCRICR0 ignores writes if all of the following are true:
>
> - The implementation uses the recommended layout for the ERRIRQCR<n> registers.
> - ERRCRICR2.NSMSI configures the physical address space for message signaled interrupts as Secure.
> - Accessed as a Non-secure access.

The equivalent changes are made in the following sections:

- I5.8.6 (ERRCRICR1, Critical Error Interrupt Configuration Register 1).
- I5.8.7 (ERRCRICR2, Critical Error Interrupt Configuration Register 2).
- I5.8.11 (ERRERICR0, Error Recovery Interrupt Configuration Register 0).
- I5.8.12 (ERRERICR1, Error Recovery Interrupt Configuration Register 1).
- I5.8.13 (ERRERICR2, Error Recovery Interrupt Configuration Register 2).

- I5.8.14 (ERRFHICR0, Fault Handling Interrupt Configuration Register 0).
- I5.8.15 (ERRFHICR1, Faulting Handling Interrupt Configuration Register 1).
- I5.8.16 (ERRFHICR2, Faulting Handling Interrupt Configuration Register 2).

In section I5.8.7 (ERRCRICR2, Critical Error Interrupt Configuration Register 2), the text in the description of NSMSI, bit [6], that reads:

> When the component supports configuring the Security attribute for messaged signaled interrupts and the component does not allow Non-secure writes to ERRCRICR2:
>
> Security attribute. Defines the physical address space for message signaled interrupts.
>
> `0b0` Secure.
>
> `0b1` Non-secure.
>
> The reset behavior of this field is:
>
> - On a Error recovery reset,this field resets to an IMPLEMENTATION DEFINED VALUE.
>
> When the component allows Non-secure writes to ERRCRICR2:
>
> Reserved, **RES0**.
>
> Security attribute. Defines the physical address space for message signaled interrupts.
>
> The Security attribute used for message signaled interrupts is Non-secure.

is changed to read:

> When the component supports configuring the physical address space for message signaled interrupts:
>
> Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.
>
> `0b0` Secure physical address space.
>
> `0b1` Non-secure physical address space.
>
> The reset behavior of this field is:
>
> - On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED VALUE.
>
> Accessing this field has the following behavior:
>
> - If accessed as a Non-secure access, access to this field is RES1.
> - Otherwise, access to this field is RW.

The equivalent changes are made in the following sections:

- I5.8.13 (ERRERICR2, Error Recovery Interrupt Configuration Register 2).
- I5.8.16 (ERRFHICR2, Faulting Handling Interrupt Configuration Register 2).

## 2.12 D18800

In section D17.5.17 (PMUSERENR_EL0, Performance Monitors User Enable Register), the EN, bit [0] description is updated to read:

> Enable EL0 accesses to the Performance Monitor registers. This applies to the following register accesses:
>
> AArch64:
>
> - MRS or MSR accesses to PMCR_EL0, PMOVSCLR_EL0, PMSELR_EL0, PMCCNTR_EL0, PMXEVTYPER_EL0, PMXEVCNTR_EL0, PMCNTENSET_EL0, PMCNTENCLR_EL0, PMOVSSET_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMCCFILTR_EL0.
> - MSR accesses to PMSWINC_EL0.
> - MRS accesses to PMCEID0_EL0, PMCEID1_EL0.
>
> AArch32:
>
> - MRC and MCR accesses to PMCR, PMOVSR, PMSELR, PMCCNTR, PMXEVTYPER, PMXEVCNTR, PMCNTENSET, PMCNTENCLR, PMOVSSET, PMEVCNTR<n>, PMEVTYPER<n>, PMCCFILTR.
> - MCR accesses to PMSWINC.
> - MRC accesses to PMCEID0, PMCEID1.
> - If FEAT_PMUv3p1 is implemented, MRC accesses to PMCEID2, and PMCEID3.
>
> `0b0` EL0 access to the specified registers is trapped, unless access is enabled by another field in this register.
>
> `0b1` EL0 access to the specified registers is allowed, unless trapped by a higher priority exception.
>
> When not enabled by any of the PMUSERENR_EL0.{ER, CR, SW, EN} controls, an accesses to the register at EL0 is trapped to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.
>
> Trapped MRS and MSR accesses are reported using EC syndrome value `0x18`.
>
> Trapped MRC and MCR accesses are reported using EC syndrome value `0x03`.
>
> Trapped MRRC and MCRR accesses are reported using EC syndrome value `0x04`.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

Equivalent changes are made to the {ER, CR, SW} fields, and to the PMUSERENR.{ER, CR, SW, EN} fields in section G8.4.18 (PMUSERENR, Performance Monitors User Enable Register).

## 2.13 C18842

In section I5.5.14 (AMDEVARCH, Activity Monitors Device Architecture Register), the text in the ARCHID, bits [15:0] description that reads:

> For AMU:
>
> - Bits [15:12] are the architecture version, `0x0`.
> - Bits [11:0] are the architecture part number, `0xA66`.
>
> This corresponds to AMU architecture version AMUv1.

is changed to read:

> For AMU:
>
> - Bits [19:16] are the minor architecture version, `0x0`.
> - Bits [15:12] are the major architecture version, `0x0`.
> - Bits [11:0] are the architecture part number, `0xA66`.
>
> This corresponds to a generic AMU, version 1.0.

## 2.14 C18843

The current description of FEAT_LPA2 in *Arm® Architecture Reference Manual, for A-profile architecture Issue I.a* lacks clarity between the ability to describe the size of the output address as having 52 bits, and there being 52 bits of physical address. This will be rectified in a future release of *Arm® Architecture Reference Manual for A-profile architecture*.

## 2.15 D18853

In section D17.2.107 (RGSR_EL1, Random Allocation Tag Seed Register), the field descriptions are changed to read:

> When GCR_EL1.RRND == 0:
>
> Bits [63:24]
>
> Reserved, RES0.
>
> SEED, bits [23:8]

Seed register used for generating values returned by RandomAllocationTag().

The reset behavior of this field is:

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

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

The reset behavior of this field is:

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

When GCR_EL1.RRND == 1:

Bits [63:56]

Reserved, RES0.

SEED, bits [55:8]

IMPLEMENTATION DEFINED

Note: Software is recommended to avoid writing SEED[15:0] with a value of zero, unless this has been generated by the PE in response to an earlier value with SEED being non-zero.

The reset behavior of this field is:

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

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

The reset behavior of this field is:

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

## 2.16 D18928

In section A3.1 (Armv9-A architecture extensions), in the feature introduction of FEAT_ETEv1p2, the text that reads:

> This feature requires FEAT_RME and FEAT_ETEv1p1.

is corrected to read:

> This feature requires FEAT_ETEv1p1.
>
> This feature is required if FEAT_RME is implemented and FEAT_ETE is implemented.

## 2.17 C19027

In section D11.11.3 (Common event numbers), subsection 'Common microarchitectural events', the following text is added to the descriptions of MEM_ACCESS_CHECKED (`0x4024`), MEM_ACCESS_CHECKED_RD (`0x4025`), and MEM_ACCESS_CHECKED_WR (`0x4026`):

> It is **IMPLEMENTATION DEFINED** whether the counter increments on a Tag Checked access made when Tag Check Faults are configured to be ignored by SCTLR_ELx.TCF or SCTLR_ELx.TCF0.

## 2.18 C19047

In section D17.2.27 (CLIDR_EL1, Cache Level ID Register), the following Note is added to the descriptions of LoUU, bits [29:27], and LoUIS, bits [23:21]:

> Note: This field does not describe the requirements for instruction cache invalidation. See CTR_EL0.DIC.

The equivalent changes are made in section G8.2.27 (CLIDR, Cache Level ID Register).

## 2.19 D19116

In section D17.11.21 (CNTPS_CTL_EL1, Counter-timer Physical Secure Timer Control register), the following text is added under 'Configurations':

> This register is present only when EL3 is implemented. Otherwise, direct accesses to CNTPS_CTL_EL1 are **UNDEFINED**.

Equivalent changes are made in the following sections:

- D17.11.23 (CNTPS_CVAL_EL1, Counter-timer Physical Secure Timer CompareValue register).
- D17.11.24 (CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue register).

## 2.20 D19121

In section D17.2.118 (SCTLR_EL1, System Control Register (EL1)), in field 'C, bit [2]', the text that reads:

> When the value of the HCR_EL2.DC bit is 1, the PE ignores SCTLR.C. This means that Non-secure EL0 and Non-secure EL1 data accesses to Normal memory are Cacheable.
>
> When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

is changed to read:

> When the Effective value of the HCR_EL2.DC bit in the current Security state is 1, the PE ignores SCTLR_EL1.C. This means that EL0 and EL1 data accesses to Normal memory are Cacheable.
>
> When FEAT_VHE is implemented, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

Similarly in field 'M, bit [0]', the text that reads:

> If the value of HCR_EL2.{DC, TGE} is not {0, 0} then in Non-secure state the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.
>
> When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

is changed to read:

> If the Effective value of HCR_EL2.{DC, TGE} in the current Security state is not {0, 0} then the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.
>
> When FEAT_VHE is implemented, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The equivalent changes are made in section G8.2.126 (SCTLR, System Control Register).

## 2.21 D19178

In section J1.1.3 (aarch64/functions), the function AddressSupportsLS64(), that reads as:

```
boolean AddressSupportsLS64(bits(64) address)
```

Is updated to read as:

```
boolean AddressSupportsLS64(bits(52) paddress);
```

The following changes are also made in the same section:

In MemStore64B(), the code that reads:

```
MemStore64B(bits(64) address, bits(512) value, AccType acctype)
    boolean iswrite = TRUE;
    constant integer size = 64;
    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);
    if !AddressSupportsLS64(address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};
        if c == Constraint_FAULT then
            ...
        else
            // Accesses are not single-copy atomic above the byte level.
            for i = 0 to 63
                AArch64.MemSingle[address+8*i, 1, acctype, aligned] = value<7+8*i :
 8*i>;
    else
        -= MemStore64BWithRet(address, value, acctype);  // Return status is ignored
 by ST64B
return;
```

Is updated to read:

```
MemStore64B(bits(64) address, bits(512) value, AccType acctype)
    boolean iswrite = TRUE;
    constant integer size = 64;
    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);
    AddressDescriptor memaddrdesc = AArch64.TranslateAddress(address, acctype,
 iswrite,
                                                             istagaccess, aligned,
 size);

    // Check for aborts or debug exceptions
    if IsFault(memaddrdesc) then
        AArch64.Abort(address, memaddrdesc.fault);

    // Effect on exclusives
    if memaddrdesc.mematttrs.shareability != Shareability_NSH then
        ClearExclusiveByAddress(memaddrdesc.paddress, ProcessorID(), 64);

    // Memory array access
    accdesc = CreateAccessDescriptor(acctype);
    if !AddressSupportsLS64(memaddrdesc.paddress.address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};

        if c == Constraint_FAULT then
            ...
        else
            // Accesses are not single-copy atomic above the byte level.
            accdesc.acctype = AccType_ATOMIC;
            for i = 0 to size-1
                memstatus = PhysMemWrite(memaddrdesc, 1, accdesc, value<8*i+7:8*i>);
                if IsFault(memstatus) then
                    HandleExternalWriteAbort(memstatus, memaddrdesc, 1, accdesc);
                memaddrdesc.paddress.address = memaddrdesc.paddress.address+1;
    else
```

```
        memstatus = PhysMemWrite(memaddrdesc, size, accdesc, value);
        if IsFault(memstatus) then
            HandleExternalWriteAbort(memstatus, memaddrdesc, size, accdesc);
    return;
```

In MemLoad64B(), the code that reads:

```
bits(512) MemLoad64B(bits(64) address, AccType acctype)
    ...

    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);

    if !AddressSupportsLS64(address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};

        if c == Constraint_FAULT then
            // Generate a stage 1 Data Abort reported using the DFSC code of 110101.
            boolean secondstage = FALSE;
            boolean s2fs1walk = FALSE;
            FaultRecord fault = AArch64.ExclusiveFault(acctype, iswrite,
 secondstage, s2fs1walk);
            AArch64.Abort(address, fault);
        else
            // Accesses are not single-copy atomic above the byte level
            for i = 0 to 63
                data<7+8*i : 8*i> = AArch64.MemSingle[address+8*i, 1, acctype,
 aligned];
            return data;

    AddressDescriptor memaddrdesc;
    memaddrdesc = AArch64.TranslateAddress(address, acctype, iswrite, istagaccess,
 aligned, size);

    // Check for aborts or debug exceptions
    if IsFault(memaddrdesc) then
        ...
    accdesc = CreateAccessDescriptor(acctype);
    PhysMemRetStatus memstatus;
    (memstatus, data) = PhysMemRead(memaddrdesc, size, accdesc);
    if IsFault(memstatus) then
        HandleExternalReadAbort(memstatus, memaddrdesc, size, accdesc);
    return data;
```

Is updated to read as:

```
bits(512) MemLoad64B(bits(64) address, AccType acctype)
    ...

    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);

    AddressDescriptor memaddrdesc;
    memaddrdesc = AArch64.TranslateAddress(address, acctype, iswrite, istagaccess,
 aligned, size);

    // Check for aborts or debug exceptions
    if IsFault(memaddrdesc) then
       ...
    accdesc = CreateAccessDescriptor(acctype);
    if !AddressSupportsLS64(memaddrdesc.paddress.address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};

        if c == Constraint_FAULT then
            // Generate a stage 1 Data Abort reported using the DFSC code of 110101.
            boolean secondstage = FALSE;
```

```
            boolean s2fs1walk = FALSE;
            FaultRecord fault = AArch64.ExclusiveFault(acctype, iswrite,
secondstage, s2fs1walk);
            AArch64.Abort(address, fault);
        else
            // Accesses are not single-copy atomic above the byte level.
            accdesc.acctype = AccType_ATOMIC;
            for i = 0 to size-1
                PhysMemRetStatus memstatus;
                (memstatus, data<8*i+7:8*i>) = PhysMemRead(memaddrdesc, 1, accdesc);
                if IsFault(memstatus) then
                    HandleExternalReadAbort(memstatus, memaddrdesc, 1, accdesc);
                memaddrdesc.paddress.address = memaddrdesc.paddress.address + 1;
    else
        PhysMemRetStatus memstatus;
        (memstatus, data) = PhysMemRead(memaddrdesc, size, accdesc);
        if IsFault(memstatus) then
            HandleExternalReadAbort(memstatus, memaddrdesc, size, accdesc);
    return data;
```

## 2.22 D19239

In section D17.2.49 (HCRX_EL2, Extended Hypervisor Configuration Register), in the fields MSCEN, MCE2, CMOW, and SMPME, the text that reads:

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

is corrected to read:

> On a Warm reset:
>
> * When EL3 is not implemented and EL2 is implemented, this field resets to 0.
>
> * Otherwise, this field resets to an architecturally **UNKNOWN** value.

In the same register, in the fields VFNMI, VINMI, TALLINT, FGTnXS, FnXS, EnASR, EnALS and EnAS0, the text that reads:

> On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

is corrected to read:

> On a Warm reset:
>
> * When EL3 is not implemented and EL2 is implemented, this field resets to 0.
>
> * Otherwise, this field resets to an architecturally **UNKNOWN** value.

## 2.23 R19370

In section E2.8.1 (Normal memory), after the text that reads:

> Writes to a memory location with the Normal memory type that is either Non-cacheable or Write-Through cacheable for both the Inner and Outer Cacheability must reach the endpoint for that location in the memory system in finite time. Two writes to the same location, where at least one is using the Normal memory type, might be merged before they reach the endpoint unless there is an ordered-before relationship between the two writes.

The following text is added:

> For the purposes of this requirement, the endpoint for a location in Conventional memory is the PoC.

## 2.24 D19372

In section D17.2.107 (RGSR_EL1, Random Seed Allocation Tag Seed Register), the following text is added under 'Configurations':

> When GCR_EL1.RRND==0b0, direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

## 2.25 D19451

In section C6.2.378 (TLBI), the following statements are added to the definition of <tlbi_op> in the 'Assembler symbols' subsection:

> When FEAT_RME is implemented, the following values are also valid:

| | |
|---|---|
| PAALLOS | when op1 = 110, CRn = 1000, CRm = 0001, op2 = 100 |
| RPAOS | when op1 = 110, CRn = 1000, CRm = 0100, op2 = 011 |
| RPALOS | when op1 = 110, CRn = 1000, CRm = 0100, op2 = 111 |
| PAALL | when op1 = 110, CRn = 1000, CRm = 0111, op2 = 100 |

## 2.26 D19452

Following the update communicated as D18736, in section I5.8.7 (ERRCRICR2, Critical Error Interrupt Configuration Register 2), the text that reads:

> If accessed as a Non-secure access, access to this field is **RES1**.

is updated to read:

> If accessed as a Non-secure or Realm access, access to this field is WI.

The equivalent changes are made in the following sections:

- I5.8.13 (ERRERICR2, Error Recovery Interrupt Configuration Register 2).
- I5.8.16 (ERRFHICR2, Faulting Handling Interrupt Configuration Register 2).

## 2.27 R19519

In section B2.3.10 (Restrictions on the effects of speculation), in the subsection 'Restrictions on the effects of speculation from Armv8.5', the sub bullet point that reads as follows:

> - Data Value predictions based on data value from execution in context1.

is updated to include the following Note:

> PSTATE.{N,Z,C,V} values from context1 are not considered a data value for this purpose.

The equivalent change is made in section E2.3.9 (Restrictions on the effects of speculation), in the subsection 'Further restrictions on the effects of speculation from Armv8.5'.

In section C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context), the following note is added:

> The prediction of the PSTATE.{N,Z,C,V} values is not considered a data value for this purpose.

The equivalent change is made in section G8.2.50 (DVPRCTX, Data Value Prediction Restriction by Context).

## 2.28 D19521

In section C5.2.25 (SVCR, Streaming Vector Control Register), for the field ZA, bit [1], the text that reads:

> When a write to SVCR.ZA changes the value of PSTATE.ZA, the following applies:
>
> When changed from 0 to 1, all implemented bits of the storage are set to zero.
>
> When changed from 1 to 0, there is no observable change to the storage.
>
> Changes to this field do not have an affect on the SVE vector and predicate registers and FPSR.

is corrected to read:

> When a write to SVCR.ZA changes the value of PSTATE.ZA from 0 to 1, all implemented bits of the storage are set to zero.

> Changes to this field do not have an effect on the SVE vector and predicate registers and FPSR.

## 2.29 D19549

In section D11.11.3 (Common event numbers), in the subsection 'Common microarchitectural events', for each TRCEXTOUT<n> event, where <n> is 0 to 3, the text that reads:

> This event must be implemented if FEAT_ETE is implemented.

is updated to read:

> This event must be implemented if FEAT_ETE is implemented and the ETE implements External output <n>.

## 2.30 D19560

In section D17.2.26 (CCSIDR_EL1, Current Cache Size Register), the text in LineSize, bits [2:0] when FEAT_CCIDX is implemented that reads:

> When FEAT_MTE is implemented and enabled, where a cache only holds Allocation tags, this field is **RES0**.

is changed to read:

> When FEAT_MTE is implemented, where a cache only holds Allocation tags, this field is **RES0**.

The following text is added to LineSize, bits [2:0] when FEAT_CCIDX is not implemented:

> When FEAT_MTE is implemented, where a cache only holds Allocation tags, this field is **RES0**.

## 2.31 D19561

In section D17.2.107 (RGSR_EL1, Random Allocation Tag Seed Register), the text that reads:

> When GCR_EL1.RRND=0, direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

is changed to read:

> Direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

## 2.32 D19581

In the function AArch64.RestrictPrediction() in section J1.1.4 (aarch64/instrs), the code that reads:

```
// If the instruction is executed at an EL lower than the specified
// level, it is treated as a NOP.
if UInt(target_el) > UInt(PSTATE.EL) then return;
```

Is updated to read:

```
// If the target EL is not implemented or the instruction is executed at an
// EL lower than the specified level, the instruction is treated as a NOP.
if !HaveEL(target_el) || UInt(target_el) > UInt(PSTATE.EL) then EndOfInstruction();
```

This affects the A64 System instructions in the following sections:

- C5.6.1 (CFP RCTX, Control Flow Prediction Restriction by Context).
- C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context).
- C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context).

An equivalent change is made in AArch32.RestrictPrediction() affecting the AArch32 System Registers in the following sections:

- G8.2.26 (CFPRCTX, Control Flow Prediction Restriction by Context).
- G8.2.34 (CPPRCTX, Cache Prefetch Prediction Restriction by Context).
- G8.2.50 (DVPRCTX, Data Value Prediction Restriction by Context).

## 2.33 D19642

In section D11.11.3 (Common event numbers), subsection 'Common microarchitectural events', the PMU events that read:

> `0x4025`, MEM_ACCESS_RD_CHECKED, Checked data memory access, read
>
> `0x4026`, MEM_ACCESS_WR_CHECKED, Checked data memory access, write

are corrected to read:

> `0x4025`, MEM_ACCESS_CHECKED_RD, Checked data memory access, read
>
> `0x4026`, MEM_ACCESS_CHECKED_WR, Checked data memory access, write

## 2.34 C19644

In section D11.11.3 (Common event numbers), subsection 'Common microarchitectural events', the text in the descriptions of MEM_ACCESS_CHECKED_RD (`0x4025`) and MEM_ACCESS_CHECKED_WR (`0x4026`) that reads:

> Implementation of this optional event requires that FEAT_MTE is implemented.

is corrected to read:

> Implementation of this optional event requires that FEAT_MTE2 is implemented.

This text is also added to the MEM_ACCESS_CHECKED (`0x4024`) event description.

## 2.35 C19649

In section B2.7.2 (Device Memory), in subsection 'Reordering', the bullet point in the note that reads:

> The non-Reordering property is only required by the architecture to apply the order of arrival of accesses to a single memory-mapped peripheral of an **IMPLEMENTATION DEFINED** size, and is not required to have an impact on the order of observation of memory accesses to SDRAM. For this reason, there is no effect of the non-Reordering attribute on the ordering relations between accesses to different locations described in Ordering relations on page B2-165 as part of the formal definition of the memory model.

is updated to read:

> The non-Reordering property is only required by the architecture to apply the order of arrival of accesses to a single memory-mapped peripheral of an **IMPLEMENTATION DEFINED** size, and is not required to have an impact on the order of observation of memory accesses to SDRAM. For this reason, there is no effect of the non-Reordering attribute on the ordering relations between accesses to different locations described in B2.3.3 Ordering relations on page B2-165 as part of the formal definition of the memory model. It does have an effect on the Peripheral Coherence Order described in section B2.3.7 (Completion and endpoint ordering).

## 2.36 C19749

In section D17.2.51 (HDFGWTR_EL2, Hypervisor Debug Fine-Grained Write Trap Register), in field PMCCNTR_EL0, bit [15], the following text is added:

> PMCCNTR_EL0 is indirectly accessed when PMCR_EL0.C is set to `0b1`.
>
> Setting this field to 1 has no effect on accesses to PMCCNTR_EL0 using PMCR_EL0.

and in field PMEVCNTRn_EL0, bit [12], the following text is added:

> PMEVCNTR<n>_EL0 is indirectly accessed when PMCR_EL0.P is set to `0b1`.
>
> Setting this field to 1 has no effect on accesses to PMEVCNTR<n>_EL0 using PMCR_EL0.

The same changes are made in the corresponding fields in section D17.2.50 (HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register).

## 2.37 D19753

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

```
Halt(bits(6) reason, boolean is_async)

   CTI_SignalEvent(CrossTriggerIn_CrossHalt);  // Trigger other cores to halt
   ...
```

Is updated to read:

```
Halt(bits(6) reason, boolean is_async)

    if HaveTME() && TSTATE.depth > 0 then
        FailTransaction(TMFailure_DBG, FALSE);

    CTI_SignalEvent(CrossTriggerIn_CrossHalt);  // Trigger other cores to halt
    ...
```

## 2.38 C19772

In section C5.5.10 (TLBI ASIDE1, TLBI ASIDE1NXS, TLB Invalidate by ASID, EL1), in the subsection 'Executing TLBI ASIDE1, TLBI ASIDE1NXS instruction', the EL1 accessibility pseudocode that reads:

```
elsif EL2Enabled() && HCR_EL2.FB == '1' then
    if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) &&
 HCRX_EL2.FnXS == '1' then
       AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Shareability_ISH, TLBI_ExcludeXS, X[t, 64]);
```

is updated to read:

```
elsif EL2Enabled() && HCR_EL2.FB == '1' then
    if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) &&
 IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
       AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Shareability_ISH, TLBI_ExcludeXS, X[t, 64]);
```

The same edits are made in the following sections:

- C5.5.29 (TLBI RVAAE1, TLBI RVAAE1NXS).
- C5.5.32 (TLBI RVAALE1, TLBI RAAVLE1NXS).

- C5.5.35 (TLBI RVAE1, TLBI RVAE1NXS).
- C5.5.44 (TLBI RVALE1, TLBI RAVLE1NXS).
- C5.5.53 (TLBI VAAE1, TLBI VAAE1NXS).
- C5.5.56 (TLBI VAALE1, TLBI VAALE1NXS).
- C5.5.59 (TLBI VAE1, TLBI VAE1NXS).
- C5.5.68 (TLBI VALE1, TLBI VALE1NXS).
- C5.5.77 (TLBI VMALLE1, TLBI VMALLE1NXS).
- G8.2.136 (TLBIALL, TLB Invalidate All).
- G8.2.142 (TLBIASID, TLB Invalidate by ASID match).
- G8.2.148 (TLBIMVA, TLB Invalidate by VA).
- G8.2.149 (TLBIMVAA, TLB Invalidate by VA, All ASID).
- G8.2.151 (TLBIMVAAL, TLB Invalidate by VA, All ASID, Last level).
- G8.2.156 (TLBIMVAL, TLB Invalidate by VA, Last level).

## 2.39 D19800

In section J1.1.3 (aarch64/function), the function IsHCRXEL2Enabled(), that reads as:

```
boolean IsHCRXEL2Enabled()
    assert(HaveFeatHCX());
   ...
```

Is updated to read:

```
boolean IsHCRXEL2Enabled()
    if !HaveFeatHCX() then return FALSE;
    ...
```

## 2.40 D19817

In section G8.3.33 (PMMIR, Performance Monitors Machine Identification Register), in the 'BUS_SLOTS, bits [15:8]' field, the text that reads:

> Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.
>
> When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.
>
> This field has an **IMPLEMENTATION DEFINED** value.

> Access to this field is RO.

is corrected to read:

> Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.
>
> When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.
>
> If the information is not available, this field will read as zero.
>
> This field has an **IMPLEMENTATION DEFINED** value.
>
> Access to this field is RO.

The equivalent changes are made in section D17.5.12 (PMMIR_EL1, Performance Monitors Machine Identification Register) and I5.3.30 (PMMIR, Performance Monitors Machine Identification Register).

## 2.41 D19829

In section D17.2.63 (ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2) in the 'RPRES, bits [7:4]' field, the following text is removed:

> From Armv8.7, if Advanced SIMD and floating-point is implemented, the only permitted value is `0b0001`.

## 2.42 E19831

In section K7.2 (Gray-count scheme for timer distribution scheme), the following pseudocode for Gray code encoding and decoding:

```
Gray[N] = Count[N]
Gray[i] = (XOR(Gray[N:i+1])) XOR Count[i] for N-1 >= i >= 0
Count[i] = XOR(Gray[N:i]) for N >= i >= 0
```

is updated to read:

```
Gray = Count EOR ('0':Count<N:1>)

Count<N> = Gray<N>
for i = N-1 downto 0
   Count<i> = Gray<i> EOR Count<i+1>
```

## 2.43 D19833

In section K7.2 (Gray-count scheme for timer distribution scheme) the following Note is removed:

> This scheme has the advantage of being relatively simple to switch, in either direction, between operating with low-frequency and low-precision, and operating with high-frequency and high-precision. To achieve this, the ratio of the frequencies must be $2^n$, where n is an integer. A switch-over can occur only on the 2 n+1 boundary to avoid losing the Gray-coding property on a switch-over.

## 2.44 C19835

In section B2.3.12 (Limited ordering regions), after the following text:

> A memory location lies within the LORegion identified by the LORegion Number if the PA lies between the Start Address and the End Address, inclusive. The Start Address must be defined to be aligned to 64KB and the End Address must be defined as the top byte of a 64KB block of memory.

the following statement is added:

> It is permitted for multiple LORegion descriptors with non-overlapping address ranges to be configured with the same LORegion Number.

## 2.45 C215: SVE

Arm is making a retrospective change to the SVE architecture to remove the capability of selecting a non-power-of-two vector length in non-Streaming SVE as well as in Streaming SVE mode. Specific updates as a result of this change will be communicated in due course.

## 2.46 D1461: Armv9 Debug

In section D4.6.12 (External Outputs), the statement $I_{BZHDF}$ that reads:

> The ETE architecture supports between one and four External Outputs. The number of outputs that a trace unit has is **IMPLEMENTATION DEFINED**, but at least one output is always implemented.

is updated to read:

> The ETE architecture supports between zero and four External Outputs. The number of outputs that a trace unit has is **IMPLEMENTATION DEFINED**, and Arm recommends that at least one output is implemented.

## 2.47 D1466: Armv9 Debug

In section D11.11.3 (Common event numbers), in the subsection 'Common microarchitectural events', the description for each CTI_TRIGOUT<n> event, where <n> is in the range 4 to 7, that reads:

> This event must be implemented if FEAT_ETE is implemented.

is updated to read:

> This event must be implemented if FEAT_ETE is implemented and TRCIDR5.NUMEXTINSEL > (n - 4).

## 2.48 D1493: Armv9 Debug

In section D4.5.3 (Trace unit behavior while the PE is in Debug state), rule $R_{DPKSC}$ that reads:

> While the PE is in Debug state, the trace unit does not trace instructions that are executed.

is updated to read:

> While the PE is in Debug state, the trace unit:
> - Does not trace instructions that are executed.
> - Does not trace the effects of instructions that are executed.
> - Does not trace Exceptional occurrences.

Additionally, in section D4.5.8 (Filtering trace generation), in the subsection 'Rules for tracing Exceptional occurrences', rule $R_{DPMBQ}$ that reads:

> When an Exceptional occurrence occurs and TRCRSR.TA is `0b1`, the Exceptional occurrence is traced.

is updated to read:

> When an Exceptional occurrence occurs and the PE is not in Debug state and TRCRSR.TA is `0b1`, the Exceptional occurrence is traced.