



# Arm GPU Errata for Application Developers

## Software Developer Errata Notice

**Date of issue:** February 07, 2025

**Non-Confidential**

**Document version:** 2.0

Copyright © 2024-2025 Arm® Limited (or its affiliates). All rights reserved.

**Document ID:** SDEN-3735689

This document contains all known errata since the r38p0 release of the product.



This document is Non-Confidential.

Copyright © 2024-2025 Arm® Limited (or its affiliates). All rights reserved.

This document is protected by copyright and other intellectual property rights.

Arm only permits use of this document if you have reviewed and accepted Arm's Proprietary notice found at the end of this document.

This document (SDEN\_3735689\_2.0\_en) was issued on February 07, 2025.

There might be a later issue at <http://developer.arm.com/documentation/SDEN-3735689>

## 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.

If you find offensive language in this document, please email [terms@arm.com](mailto:terms@arm.com).

## Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on Arm GPU Errata for Application Developers, 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>.

# Contents

<b>Product version scope</b>	5
<b>Driver version information</b>	5
<b>Hardware version information</b>	5
<b>Introduction</b>	6
Scope	6
Categorization of errata	6
<b>Change Control</b>	7
<b>Errata summary table</b>	9
<b>Errata descriptions</b>	11
Category A	11
Category A (rare)	12
3787671 Shader scalar integer right shifts return an incorrect result	12
Category B	13
3734951 Freeing command buffers without explicit release causes memory leaks	13
3779191 Read-only storage buffer accesses in inactive control flow are speculatively executed	14
3779215 Shader clamp of conditional values with a zero limit produces incorrect code	15
3781413 Rendering large amounts of geometry causes rendering artifacts or DEVICE_LOST	16
3781554 Unreferenced vertex indices are speculatively shaded	18
3785718 Pipeline barriers are not transitive when intermediate stage is HOST	19
3786496 vkCmdBindVertexBuffers2() updates stride for an incorrect binding	20
3786517 vkCmdSetCullMode() incorrectly culls non-triangle topologies	21
3786532 Freeing simultaneous use command buffers causes memory corruption	22
3786926 vkCmdBeginRendering() causes rendering artifacts or DEVICE_LOST	23
3787459 Incorrect rendering when input gl_Position is declared outside of an input block	24
3787677 Render pass loadOp will return black for 3D images that are 32x32x1 or smaller	25
3809538 Dynamic render state incorrectly inherited on pipeline change	26
3809543 Vulkan pipeline caches incorrectly accessed when created with external synchronization	27
3809544 OpenGL ES shaders incorrectly use buffer instance name to define bind location	28
3809560 OpenGL ES surface reads have missing dependency on earlier MSAA surface writes	29
3809773 OpenGL ES deadlocks when deleting resources when many compute dispatches are outstanding	30

3809774	Incorrect rendering when dynamic state disables depth/stencil write if shader reads depth/stencil	31
3817626	Shader constant folding treats unsigned integer values as signed	32
3843108	Atomic access coordinates for images and texture buffers always truncated to 16-bits	33
3843237	Vector bitwise of swizzle/combine returns incorrect result	34
3893903	Dynamic render state incorrectly used when modified between two binds of a static pipeline	35
3893905	Dynamic render state incorrectly inherited on pipeline change (2)	36
3893908	Declaring NxM matrices as row_major reports incorrect buffer size	37
3893910	Declaring matrices in struct or array as row_major reports incorrect buffer size	38
3893911	vkQueuePresentKHR() can hang if no vkQueueSubmit() calls were made	39
Category B (rare)		39
Category C		40
3817843	Vulkan VK_ARM_scheduling_controls feature query always reports false	40
<b>Proprietary notice</b>		41
<b>Product and document information</b>		43
Product status		43
Product completeness status		43
Product revision status		43

## Product version scope

This document covers Arm GPU and driver errata that have API-visible impact on application software.

To be included in this document an erratum must meet the following criteria:

- Must impact Arm GPU hardware from the Mali-G710 series onwards.
- Must impact Arm GPU drivers from the r38p0 release onwards.
- Must have been encountered by an application developer on a shipping device.

## Driver version information

The latest Arm DDK driver release version at the time this document was generated was r52p0.

The impacted driver versions listed for each erratum are based on the version of the original Arm DDK release made by Arm. Older drivers released in OEM devices may include backported fixes from a later Arm DDK release, and therefore not be impacted by an issue listed here.

Application-visible errata caused by hardware issues that are independent of driver version will not document an impacted driver version.

## Hardware version information

Arm GPUs are released as sets of products based on the same microarchitecture. Errata will list the GPU series, instead of individual products, if the whole series is impacted.

Product series	Products
Mali-G710 series	Mali-G710, Mali-G610, Mali-G510, Mali-G310
Immortalis-G715 series	Immortalis-G715, Mali-G715, Mali-G615
Immortalis-G720 series	Immortalis-G720, Mali-G720, Mali-G620
Immortalis-G925 series	Immortalis-G925, Mali-G725, Mali-G625

Application-visible errata that are independent of the hardware product will not document an impacted hardware version.

# Introduction

## Scope

This document describes errata categorized by level of severity. Each description includes:

- The current status of the erratum.
- Where the implementation deviates from the specification and the conditions required for erroneous behavior to occur.
- The implications of the erratum with respect to typical applications.
- The application and limitations of a workaround where possible.

## Categorization of errata

Errata are split into three levels of severity and further qualified as common or rare:

<b>Category A</b>	A critical error. No workaround is available or workarounds are impactful. The error is likely to be common for many systems and applications.
<b>Category A (Rare)</b>	A critical error. No workaround is available or workarounds are impactful. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
<b>Category B</b>	A significant error or a critical error with an acceptable workaround. The error is likely to be common for many systems and applications.
<b>Category B (Rare)</b>	A significant error or a critical error with an acceptable workaround. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
<b>Category C</b>	A minor error.

# Change Control

Errata are listed in this section if they are new to the document, or marked as "updated" if there has been any change to the erratum text. Fixed errata are not shown as updated unless the erratum text has changed. The [errata summary table](#) identifies errata that have been fixed in each product revision.

## February 07, 2025: Changes in document version v2.0

ID	Status	Area	Category	Summary
<a href="#">3786926</a>	Updated	Programmer	Category B	vkCmdBeginRendering() causes rendering artifacts or DEVICE_LOST
<a href="#">3809538</a>	New	Programmer	Category B	Dynamic render state incorrectly inherited on pipeline change
<a href="#">3809543</a>	New	Programmer	Category B	Vulkan pipeline caches incorrectly accessed when created with external synchronization
<a href="#">3809544</a>	New	Programmer	Category B	OpenGL ES shaders incorrectly use buffer instance name to define bind location
<a href="#">3809560</a>	New	Programmer	Category B	OpenGL ES surface reads have missing dependency on earlier MSAA surface writes
<a href="#">3809773</a>	New	Programmer	Category B	OpenGL ES deadlocks when deleting resources when many compute dispatches are outstanding
<a href="#">3809774</a>	New	Programmer	Category B	Incorrect rendering when dynamic state disables depth/stencil write if shader reads depth/stencil
<a href="#">3817626</a>	New	Programmer	Category B	Shader constant folding treats unsigned integer values as signed
<a href="#">3843108</a>	New	Programmer	Category B	Atomic access coordinates for images and texture buffers always truncated to 16-bits
<a href="#">3843237</a>	New	Programmer	Category B	Vector bitwise of swizzle/combine returns incorrect result
<a href="#">3893903</a>	New	Programmer	Category B	Dynamic render state incorrectly used when modified between two binds of a static pipeline
<a href="#">3893905</a>	New	Programmer	Category B	Dynamic render state incorrectly inherited on pipeline change (2)
<a href="#">3893908</a>	New	Programmer	Category B	Declaring NxM matrices as row_major reports incorrect buffer size
<a href="#">3893910</a>	New	Programmer	Category B	Declaring matrices in struct or array as row_major reports incorrect buffer size
<a href="#">3893911</a>	New	Programmer	Category B	vkQueuePresentKHR() can hang if no vkQueueSubmit() calls were made
<a href="#">3785718</a>	Updated	Programmer	Category B	Pipeline barriers are not transitive when intermediate stage is HOST
<a href="#">3786517</a>	Updated	Programmer	Category B	vkCmdSetCullMode() incorrectly culls non-triangle topologies
<a href="#">3817843</a>	New	Programmer	Category C	Vulkan VK_ARM_scheduling_controls feature query always reports false

## November 14, 2024: Changes in document version v1.0

ID	Status	Area	Category	Summary
<a href="#">3787671</a>	New	Programmer	Category A (rare)	Shader scalar integer right shifts return an incorrect result
<a href="#">3734951</a>	New	Programmer	Category B	Freeing command buffers without explicit release causes memory leaks
<a href="#">3786926</a>	New	Programmer	Category B	vkCmdBeginRendering() causes rendering artifacts or DEVICE_LOST
<a href="#">3787459</a>	New	Programmer	Category B	Incorrect rendering when input gl_Position is declared outside of an input block
<a href="#">3787677</a>	New	Programmer	Category B	Render pass loadOp will return black for 3D images that are 32x32x1 or smaller
<a href="#">3779191</a>	New	Programmer	Category B	Read-only storage buffer accesses in inactive control flow are speculatively executed
<a href="#">3779215</a>	New	Programmer	Category B	Shader clamp of conditional values with a zero limit produces incorrect code
<a href="#">3781413</a>	New	Programmer	Category B	Rendering large amounts of geometry causes rendering artifacts or DEVICE_LOST
<a href="#">3781554</a>	New	Programmer	Category B	Unreferenced vertex indices are speculatively shaded
<a href="#">3785718</a>	New	Programmer	Category B	Pipeline barriers are not transitive when intermediate stage is HOST
<a href="#">3786496</a>	New	Programmer	Category B	vkCmdBindVertexBuffers2() updates dynamic strides for an incorrect binding
<a href="#">3786517</a>	New	Programmer	Category B	vkCmdSetCullMode() incorrectly culls non-triangle topologies
<a href="#">3786532</a>	New	Programmer	Category B	Freeing simultaneous use command buffers causes memory corruption



# Errata summary table

The errata associated with this product affect the product versions described in the following table.

ID	Area	Category	Summary	Found in versions	Fixed in version
<a href="#">3787671</a>	Programmer	Category A (rare)	Shader scalar integer right shifts return an incorrect result	r42p0 - r47p0	r48p0
<a href="#">3734951</a>	Programmer	Category B	Freeing command buffers without explicit release causes memory leaks	r48p0, r49p0	r49p1, r50p0
<a href="#">3779191</a>	Programmer	Category B	Read-only storage buffer accesses in inactive control flow are speculatively executed	r29p0 - ...	Open
<a href="#">3779215</a>	Programmer	Category B	Shader clamp of conditional values with a zero limit produces incorrect code	r38p1 - r49p0, r50p0 - r51p0	r49p1, r52p0
<a href="#">3781413</a>	Programmer	Category B	Rendering large amounts of geometry causes rendering artifacts or DEVICE_LOST	r0p0 - ...	Open
<a href="#">3781554</a>	Programmer	Category B	Unreferenced vertex indices are speculatively shaded	r0p0 - ...	Open
<a href="#">3785718</a>	Programmer	Category B	Pipeline barriers are not transitive when intermediate stage is HOST	r41p0 - r49p0, r50p0 - r51p0	r49p1, r52p0
<a href="#">3786496</a>	Programmer	Category B	vkCmdBindVertexBuffers2() updates dynamic strides for an incorrect binding	r42p0 - r47p0	r48p0
<a href="#">3786517</a>	Programmer	Category B	vkCmdSetCullMode() incorrectly culls non-triangle topologies	r41p0 - r49p2, r50p0 - r51p0	r49p3, r52p0
<a href="#">3786532</a>	Programmer	Category B	Freeing simultaneous use command buffers causes memory corruption	r46p0 - r49p0, r50p0	r49p1, r51p0
<a href="#">3786926</a>	Programmer	Category B	vkCmdBeginRendering() causes rendering artifacts or DEVICE_LOST	r49p1 - r49p2, r50p0 - r51p0	r49p3, r52p0
<a href="#">3787459</a>	Programmer	Category B	Incorrect rendering when input gl_Position is declared outside of an input block	r19p0 - r46p0	r47p0
<a href="#">3787677</a>	Programmer	Category B	Render pass loadOp will return black for 3D images that are 32x32x1 or smaller	r29p0 - r38p1	r39p0
<a href="#">3809538</a>	Programmer	Category B	Dynamic render state incorrectly inherited on pipeline change	r38p0 - r44p0	r44p1
<a href="#">3809543</a>	Programmer	Category B	Vulkan pipeline caches incorrectly accessed when created with external synchronization	r43p0 - r45p0	r46p0

ID	Area	Category	Summary	Found in versions	Fixed in version
<a href="#">3809544</a>	Programmer	Category B	OpenGL ES shaders incorrectly use buffer instance name to define bind location	r43p0 - r46p0	r47p0
<a href="#">3809560</a>	Programmer	Category B	OpenGL ES surface reads have missing dependency on earlier MSAA surface writes	r44p0 - r45p0	r46p0
<a href="#">3809773</a>	Programmer	Category B	OpenGL ES deadlocks when deleting resources when many compute dispatches are outstanding	r42p0 - r47p0	r48p0
<a href="#">3809774</a>	Programmer	Category B	Incorrect rendering when dynamic state disables depth/stencil write if shader reads depth/stencil	r37p0 - r44p0	r44p1
<a href="#">3817626</a>	Programmer	Category B	Shader constant folding treats unsigned integer values as signed	r0p0 - ...	Open
<a href="#">3843108</a>	Programmer	Category B	Atomic access coordinates for images and texture buffers always truncated to 16-bits	r18p0 - r53p0	r54p0
<a href="#">3843237</a>	Programmer	Category B	Vector bitwise of swizzle/combine returns incorrect result	r18p0 - r53p0	r54p0
<a href="#">3893903</a>	Programmer	Category B	Dynamic render state incorrectly used when modified between two binds of a static pipeline	r48p0 - r53p0	r54p0
<a href="#">3893905</a>	Programmer	Category B	Dynamic render state incorrectly inherited on pipeline change (2)	r48p0 - r53p0	r54p0
<a href="#">3893908</a>	Programmer	Category B	Declaring NxM matrices as row_major reports incorrect buffer size	r18p0 - r52p0	r53p0
<a href="#">3893910</a>	Programmer	Category B	Declaring matrices in struct or array as row_major reports incorrect buffer size	r18p0 - r54p0	r54p1
<a href="#">3893911</a>	Programmer	Category B	vkQueuePresentKHR() can hang if no vkQueueSubmit() calls were made	r37p0 - r42p0, r44p0 - r49p0	r43p0, r49p1
<a href="#">3817843</a>	Programmer	Category C	Vulkan VK_ARM_scheduling_controls feature query always reports false	r47p0 - r49p2, r50p0 - r52p0	r49p3, r53p0

# Errata descriptions

## Category A

There are no errata in this category.

## Category A (rare)

### 3787671

## Shader scalar integer right shifts return an incorrect result

### Status

APIs Affected: OpenGL ES, Vulkan

Impacted driver versions: r42p0 - r47p0

Fixed driver version: r48p0

### Description

A shader compiler optimization can result in integer right shifts by a constant returning the wrong value if all of the following conditions are true:

- The shifted value is a scalar, and not a vec2 16-bit pair.
- The right shifted value was multiplied by a literal constant immediately prior to the right shift, including multiplies that are generated to implement a left shift operation.
- The literal constants must be equivalent to the pair of shifts show in the expression below where *const1* shifts more bits than *const0*.

```
(X << const0) >> const1
```

This erratum affects right shifts created explicitly, using the right shift operator, and implicitly, for instance as part of a compiler address generation.

### Implications

The result of the right shift operation will be incorrect. If this causes an incorrect address to be calculated, this might result in a Vulkan *DEVICE\_LOST* if it causes a memory fault.

### Workaround

There is no workaround for this erratum for compiler-generated shifts in addressing logic.

You can avoid the issue for user-generated shifts by sourcing either of the shift constants from a uniform rather than a literal constant.

Using OpenGL ES *GL\_EXT\_robustness* or Vulkan *robustBufferAccess* can be used to reduce the number of instances of *DEVICE\_LOST* caused by memory faults. However, in these circumstances rendering will still be incorrect as the address accessed inside the buffer will be incorrect.

## Category B

### 3734951

### Freeing command buffers without explicit release causes memory leaks

#### Status

APIs affected: Vulkan

Impacted driver versions: r48p0 - r49p0

Fixed driver versions: r49p1, r50p0

#### Description

The GPU driver might leak memory when freeing a command buffer after it has been individually reset. The issue is triggered when using the following API usage sequence:

1. Create a command pool using `vkCreateCommandPool()`, with `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT` set.
2. Allocate a command buffer from the command pool using `vkAllocateCommandBuffers()`.
3. Record some commands using the command buffer to trigger memory allocation.
4. Reset the command buffer without releasing resources, either by using an implicit reset when calling `vkBeginCommandBuffer()`, or by using `vkResetCommandBuffer()` without `VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT` set.
5. Free the command buffer using `vkFreeCommandBuffers()`.

In this scenario, the memory used for command buffer recording will not be freed or reused until the command pool has been reset using `vkResetCommandPool()` with `VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT` set, or destroyed using `vkDestroyCommandPool()`.

#### Implications

An application will run out of memory when enough memory is leaked.

#### Workaround

You can explicitly reset all command buffers using `vkResetCommandBuffer()` with `VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT` set before freeing them.

Alternatively, you can regularly reset the command pool using `vkResetCommandPool()` with `VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT` set.

## 3779191

### Read-only storage buffer accesses in inactive control flow are speculatively executed

#### Status

APIs Affected: OpenGL ES, Vulkan

Impacted driver versions: r29p0 onwards

Fixed driver version: No shipping fix

#### Description

Read-only storage buffer accesses might be speculatively executed, even when located in a code path which is dynamically not taken. For example, in the code below, the *values* array may be accessed even when *index* is greater than *maxIndex*.

```
layout(set = 0, binding = 0) buffer B {  
    vec4 values[16];  
};  
  
void main()  
{  
    uint index = ...; // Calculation of accessed index  
    uint maxIndex = ...; // Calculation of dynamically last backed index  
    if (index <= maxIndex)  
    {  
        vec4 readValue = values[index];  
    }  
}
```

#### Implications

An application might encounter incorrect rendering or Vulkan *DEVICE\_LOST* if the speculative access results in a memory fault.

#### Workaround

You can ensure that the buffer is valid for the full range of potentially speculatively accessed indices. This requires that the buffer descriptor is valid and that it is backed by sufficient memory.

Alternatively, you can use OpenGL ES *GL\_EXT\_robustness* or Vulkan *robustBufferAccess* to clamp the memory range accessed. However, note that this can reduce shader performance.

## 3779215

### Shader clamp of conditional values with a zero limit produces incorrect code

#### Status

APIs affected: OpenGL ES, Vulkan, OpenCL

Impacted driver versions: r38p1 - r49p0, r50p0 - r51p0

Fixed driver versions: r49p1, r52p0

#### Description

When clamping values between two constants the compiler might generate incorrect code if all of the following conditions are true:

- The value being clamped is conditionally chosen from values known at compile-time.
- The value being clamped can be positive or negative depending on the conditional selection.
- The value of one of the clamp limits is known at compile time to be zero.

On impacted drivers the clamp is replaced by a *min()* or *max()*, limiting to only one end of the range, instead of clamping to both ends of the range.

Clamp behavior implemented manually, for example using separate *min()* and *max()* calls, is also impacted by this errata.

The following example can trigger the errata on impacted driver versions:

```
vec2 data[4] = vec2[](vec2(-1.0, -1.0), vec2(-1.0, 3.0), vec2(3.0, -1.0), vec2(3.0, -1.0));  
vec2 pos_xy = data[min(gl_VertexIndex, 3)];  
vec2 pos_xy_clamp = clamp(pos_xy, vec2(0.0, 0.0), vec2(1.0, 1.0));
```

#### Implications

The miscalculated data value can cause undefined behavior in the impacted shader program.

#### Workaround

You can use a uniform buffer to provide the constant inputs, so that they are no longer compile-time known constants. For example:

```
uniform vec2 data[4];  
vec2 pos_xy = data[min(gl_VertexIndex, 3)];  
vec2 pos_xy_clamp = clamp(pos_xy, vec2(0.0, 0.0), vec2(1.0, 1.0));
```

Alternatively, you can add a uniform-sourced zero to each constant before using it, so that the input in to the clamp is no longer a compile-time known constants. This can have more performance overhead than directly sourcing the data from a uniform buffer.

## 3781413

### Rendering large amounts of geometry causes rendering artifacts or `DEVICE_LOST`

#### Status

APIs Affected: OpenGL ES, Vulkan

Impacted hardware: All Arm GPUs

Fixed hardware: No shipping fix, but less likely on recent products

#### Description

Arm GPUs store intermediate outputs from vertex, tessellation, and geometry pipeline stages to an implementation-owned memory pool in system memory. When the intermediate memory pool is exhausted the application might experience missing geometry or a Vulkan `DEVICE_LOST` error.

The memory allocation strategy and storage requirements have been improved in recent generations of Arm hardware, making it much harder to hit the out-of-memory condition. The sections below describe the behavior in each generation. In these sections, "Advanced geometry" includes draw calls using transform feedback, tessellation shaders, and geometry shaders.

#### Arm GPUs implementing the Midgard or Bifrost architectures

This generation of hardware defaults to a 180MB memory pool per render pass that is used for intermediate memory allocations.

For all draw calls, memory is allocated to store the output data of all pipeline stages, for all vertices in the min-to-max range of spanned indices. This includes storage for indices between the min and max that are not referenced, and storage for indices that contribute only to culled primitives.

**Note:** The 180MB limit is the Arm default, but we are aware of some entry-level devices where the limit has been lowered by device manufacturers.

#### Arm GPUs implementing the Valhall architecture

This generation of hardware defaults to a 180MB memory pool per render pass that is used for intermediate memory allocations.

For draw calls that are not using Advanced geometry, memory is allocated to store output data of the vertex shader for all vertices that are used by visible primitives.

For all draw calls that are using Advanced geometry, memory is still allocated to store the output data of all pipeline stages, for all vertices in the min-to-max range of spanned indices.

**Note:** The 180MB limit is the Arm default, but we are aware of some entry-level devices where the limit has been lowered by device manufacturers.

#### Arm GPUs implementing the 5th Generation architecture



This generation of hardware uses a larger memory pool that can be dynamically shared by multiple render passes, so there is no longer a clearly defined limit.

For draw calls that are not using Advanced geometry, this generation of hardware uses Deferred Vertex Shading (DVS). When using DVS, the upfront processing only stores primitive binning metadata information to the intermediate pool. Vertex shader data outputs are no longer stored, and are recomputed during the main phase that runs the full vertex shader and fragment shading per tile.

For all draw calls that are using Advanced geometry, memory is still allocated to store the output data of all pipeline stages, for all vertices in the min-to-max range of spanned indices.

## Implications

Exhausting the intermediate memory pool can result in rendering corruption or Vulkan *DEVICE\_LOST* errors.

## Workaround

There is no complete workaround, but limits are very hard to hit if you follow the Arm best practice advice:

- You should ensure that all indices between a draw call's min and max index are actually used in the index buffer.
- You should avoid encoding metadata in the index value high bits, even if masking the value when loading data from buffers, because it can increase the spanned index range.
- You should avoid using shader pipelines that require the Advanced geometry path in the implementation.
- You should avoid using very dense geometry meshes.
- You should minimize the per-vertex storage requirements by reducing the number and data precision of vertex attributes.

## 3781554

### Unreferenced vertex indices are speculatively shaded

#### Status

APIs Affected: OpenGL ES, Vulkan

Impacted hardware: All Arm GPUs

Fixed hardware: Partial fix in Immortalis-G925 series

#### Description

Most Arm GPUs process vertices in groups of 4 sequential index locations, naturally aligned on multiple of four index boundary. Vertex indices which are not referenced by an index buffer may be speculatively shaded if they are located in a group of 4 indices where at least one of the other indices is referenced.

For example, a draw call using the index buffer [1, 2, 3, 2, 9, 3] will shade 8 vertices in two groups of 4, group 0-3 and group 8-11.

The Immortalis-G925 series hardware will no longer speculatively shade vertices when performing a standard draw call. Pipelines using the advanced geometry implementation may still make speculative accesses. Advanced geometry includes draw calls using transform feedback, tessellation shaders, and geometry shaders.

#### Implications

An application might encounter incorrect rendering or Vulkan *DEVICE\_LOST* if the speculative execution results in a memory fault on an indirect data load. Direct vertex attribute loads will not generate faults.

#### Workaround

You can avoid speculative accesses causing data faults by ensuring that buffer data is valid for all vertices in each referenced group of four indices, padding the start and end of the buffer if required.

Alternatively, you can use OpenGL ES *GL\_EXT\_robustness* or Vulkan *robustBufferAccess* to clamp the memory access to valid buffer extents. However, note that this can reduce shader performance.

## 3785718

### Pipeline barriers are not transitive when intermediate stage is HOST

#### Status

APIs Affected: Vulkan

Impacted driver versions: r41p0 - r49p0, r50p0 - r51p0

Fixed driver version: r49p1, r52p0

#### Description

Vulkan pipeline barriers define an execution dependency chain. A pipeline barrier can depend on source stages inferred by a transitive dependency on an earlier pipeline barrier, even if it does not explicitly list those stages in its own *srcStageMask*.

For example, in the code sequence below the second dispatch has a dependency on first due to the two pipeline barriers having a transitive dependency caused by the use of the *HOST* stage. The second *vkCmdDispatch()* must wait for the first to complete before it can start processing.

```
vkCmdDispatch(1)
vkCmdPipelineBarrier(srcStageMask=COMPUTE, dstStageMask=HOST)
vkCmdPipelineBarrier(srcStageMask=HOST, dstStageMask=COMPUTE)
vkCmdDispatch(2)
```

For the impacted driver versions, hardware stage dependencies inferred by a transitive dependency on the *HOST* stage are ignored. In the example above, this means that there is no enforced wait between the two compute dispatches, and they could incorrectly run out of order.

#### Implications

Loss of synchronization can cause incorrect rendering or Vulkan *DEVICE\_LOST* if a memory access results in a memory fault.

#### Workaround

Avoid transitive *HOST* dependencies by using a *srcStageMask* that explicitly lists the hardware stages that you depend on, copying the *srcStageMask* of the first pipeline barrier into the *srcStageMask* of the second.

## 3786496

### vkCmdBindVertexBuffers2() updates stride for an incorrect binding

#### Status

APIs Affected: Vulkan

Impacted driver versions: r42p0 - r47p0

Fixed driver version: r48p0

#### Description

When using `vkCmdBindVertexBuffers2()`, a dynamic stride is applied to the incorrect binding when the index of the binding in the `VkPipelineVertexInputStateCreateInfo::pVertexBindingDescriptions` array does not match the value of the binding location in the corresponding `VkVertexInputBindingDescription`.

For example:

```
VkVertexInputBindingDescription binding_desc[2] = {};  
binding_desc[0].binding = 0;  
binding_desc[1].binding = 2;  
...  
VkDeviceSize strides[3] = { stride0, stride1, stride2 };  
vkCmdBindVertexBuffers2EXT(cmd_buf, 0, 3, buffers, offsets, nullptr, strides);
```

In this example, the erratum will mean that `stride1` is used for the binding with index 2, instead of the expected `stride2`.

This erratum also impacts `vkCmdBindVertexBuffers2EXT()`.

#### Implications

When an incorrect stride is used, an incorrect image may be rendered, or a `DEVICE_LOST` might occur if incorrect data causes a memory access fault.

#### Workaround

You can add dummy entries to the `VkPipelineVertexInputStateCreateInfo pVertexBindingDescriptions` array to ensure that the array index matches the binding index.

Alternatively, you can use static vertex buffer strides.

## 3786517

### vkCmdSetCullMode() incorrectly culls non-triangle topologies

#### Status

APIs Affected: Vulkan

Impacted driver versions: r41p0 - r49p2, r50p0 - r51p0

Fixed driver versions: r49p3, r52p0

#### Description

When using *vkCmdSetCullMode()*, the dynamic cull mode will be incorrectly applied to non-triangle topologies. This can result in primitives being incorrectly culled when they should be visible.

This erratum also impacts *vkCmdSetCullModeEXT()*.

#### Implications

An incorrect image that is missing geometry may be rendered.

#### Workaround

You can use a static cull mode for pipelines rendering non-triangle topologies.

## 3786532

### Freeing simultaneous use command buffers causes memory corruption

#### Status

APIs Affected: Vulkan

Impacted driver versions: r46p0 - r49p0, r50p0

Fixed driver versions: r49p1, r51p0

#### Description

Freeing backing memory for command buffers allocated with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` might cause memory corruption.

Memory is freed when using any of the following API calls:

- `vkResetCommandBuffer()` with `VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT` set.
- `vkFreeCommandBuffers()`.
- `vkResetCommandPool()` with `VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT` set.
- `vkDestroyCommandPool()`.

#### Implications

The graphics driver may crash or behave unpredictably.

#### Workaround

You can switch to single-use command buffers, allocating command buffers without the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` set.

## 3786926

### vkCmdBeginRendering() causes rendering artifacts or DEVICE\_LOST

#### Status

APIs Affected: Vulkan

Impacted driver versions: r49p1 - r49p2, r50p0 - r51p0

Fixed driver versions: r49p3, r52p0

#### Description

When using *vkCmdBeginRendering()* the driver will reuse internal resources based on the state values passed via the *VkRenderingInfo* structure. Incorrect state hashing can cause the driver to use stale internal data.

This erratum also impacts *vkCmdBeginRenderingKHR()*.

#### Implications

This erratum may cause rendering artifacts or *DEVICE\_LOST* during execution of the render pass.

#### Workaround

You can avoid this issue by using static render passes started with *vkCmdBeginRenderPass()*.

Alternatively, you can avoid state hash collisions by not deleting any *VkImageView* handle that is referred to by a *VkRenderingInfo* structure or its children.

## 3787459

### Incorrect rendering when input `gl_Position` is declared outside of an input block

#### Status

APIs Affected: Vulkan

Impacted driver versions: r19p0 - r46p0

Fixed driver version: r47p0

#### Description

Vulkan allows input built-in variables to be specified either as a variable, or as a member of a built-in block. This built-in block is known as the `gl_PerVertex` block in GLSL. When an incoming `gl_Position` is declared as a normal variable, outside of the `gl_PerVertex` block, the position value used might be incorrect.

Vertex position inputs impact tessellation control, tessellation evaluation and geometry shaders. It is known that HLSL shaders compiled with DXC can trigger this pattern.

#### Implications

This erratum may cause rendering artifacts due to incorrect vertex positions being used.

#### Workaround

You can change the affected SPIR-V program to declare `gl_Position` inside of a built-in block.

Alternatively, it is possible that a different high-level language to SPIR-V compiler does not emit `gl_Position` as a variable outside of a built-in block.



## 3787677

### Render pass loadOp will return black for 3D images that are 32x32x1 or smaller

#### Status

APIs Affected: Vulkan

Impacted driver versions: r29p0 - r38p1

Fixed driver version: r39p0

#### Description

Vulkan render pass attachment *loadOp* will fail to load attached image slices from a 3D image if the image is less than or equal to 32x32x1 in all three dimensions. When this erratum is encountered, the *loadOp* will return black values for the impacted attachments.

#### Implications

The failing *loadOp* might result in incorrect rendering.

#### Workaround

You can workaround this issue by adding an dummy Z plane to the impacted images, increasing the Z dimension to 2.

Alternatively, you can insert a manual readback at the start of the render pass using a draw call containing a textured quad to write the data into the framebuffer.

## 3809538

### Dynamic render state incorrectly inherited on pipeline change

#### Status

APIs Affected: Vulkan

Impacted driver versions: r38p0 - r44p0

Fixed driver version: r44p1

#### Description

Draw calls using Vulkan pipelines with dynamic render states will not use the correct dynamic value when changing pipeline if the draw immediately before the pipeline change also used the same state dynamically. For example, in the following scenario:

1. Bind Pipeline1
2. Set dynamic state
3. Draw
4. Bind Pipeline2
5. Draw

... where *Pipeline1* and *Pipeline2* are both pipelines using one of the impacted dynamic states, the second draw will not use the dynamically set render state value.

This erratum impacts the following dynamic states:

- `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE`
- `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE`
- `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP`
- `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE`
- `VK_DYNAMIC_STATE_STENCIL_OP`
- `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK`
- `VK_DYNAMIC_STATE_CULL_MODE`
- `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE`

#### Implications

The application might encounter incorrect rendering.

#### Workaround

You can reset the value of all impacted dynamic states after binding a new pipeline.

Alternatively, you can use static pipeline render state.

## 3809543

### Vulkan pipeline caches incorrectly accessed when created with external synchronization

#### Status

APIs Affected: Vulkan

Impacted driver versions: r43p0 - r45p0

Fixed driver version: r46p0

#### Description

When creating a Vulkan pipeline cache using the `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, memory corruption or a driver crash may be encountered due to the driver failing to synchronize a concurrent access made by driver-owned threads.

This erratum also impacts `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT_EXT`.

#### Implications

An application might encounter incorrect rendering due to memory corruption, or software instability.

#### Workaround

You can create Vulkan pipeline caches without setting `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`.

## 3809544

# OpenGL ES shaders incorrectly use buffer instance name to define bind location

## Status

APIs Affected: OpenGL ES

Impacted driver versions: r43p0 - r46p0

Fixed driver version: r47p0

## Description

For a buffer interface block defined as:

```
storage_qualifier block_name
{
    ....
} instance_name;
```

... the compiler buffer address binding logic incorrectly uses the interface *instance\_name*, instead of the *block\_name*, to define the internal binding location used. This might cause an incorrect binding to be accessed if the same buffer is used in multiple shader stages in a program and these stages use more than one *instance\_name* to refer to it.

## Implications

An application might encounter incorrect rendering or a context lost error.

## Workaround

You can use the same *instance\_name* for buffers that are shared across multiple shader stages.

Alternatively, you can avoid specifying an *instance\_name* and use only the *block\_name* in your shaders.

## 3809560

### OpenGL ES surface reads have missing dependency on earlier MSAA surface writes

#### Status

APIs Affected: OpenGL ES

Impacted driver versions: r44p0 - r45p0

Fixed driver version: r46p0

#### Description

Multisampled OpenGL ES framebuffer attachments can be implicitly resolved to a single sample at the end of a render pass when the framebuffer is written to memory, avoiding the need to resolve manually using a separate `glBlitFramebuffer()` call. This is the default behavior for multisampled EGL window surfaces, and can be enabled for offscreen rendering by using the `GL_EXT_multisampled_render_to_texture` extension.

During multisampled rendering with an implicit resolve, the GPU will use two logical surfaces for each attachment point:

- a multisampled surface that stores all samples per pixel,
- and a resolve surface that stores a single sample per pixel.

When the multisampled surface is implicitly invalidated at the end of a render pass, in situations where it does not need to be persisted, the driver incorrectly also clears scheduling dependencies on its sibling resolve surface. This can result in later readers of the resolve surface reading from it before it has been written, causing incorrect values to be read by the GPU or the system display controller.

#### Implications

An application might encounter incorrect rendering or screen corruption.

#### Workaround

To avoid screen corruption, you can render to single-sampled EGL window surface configurations.

To avoid off-screen rendering synchronisation issues, you can render to single-sampled OpenGL ES textures or render buffers.

## 3809773

# OpenGL ES deadlocks when deleting resources when many compute dispatches are outstanding

## Status

APIs Affected: OpenGL ES

Impacted driver versions: r42p0 - r47p0

Fixed driver version: r48p0

## Description

The OpenGL ES driver might deadlock when the application calls *glDeleteBuffers()* or *glDeleteTextures()* while the driver's internal compute dispatch task queue is full.

## Implications

The application will hang.

## Workaround

You can avoid the deadlock by ensuring that all compute workloads are complete before deleting a texture or a buffer.

## 3809774

### Incorrect rendering when dynamic state disables depth/stencil write if shader reads depth/stencil

#### Status

APIs Affected: Vulkan

Impacted driver versions: r37p0 - r44p0

Fixed driver version: r44p1

#### Description

The driver will not correctly use dynamically set states that disable depth or stencil writes if the fragment shader reads depth or stencil values from the input attachment.

This erratum impacts the following dynamic states:

- `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE`
- `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK`

#### Implications

The application might encounter incorrect rendering.

#### Workaround

You can use static pipeline state to control depth and stencil writes.

## 3817626

### Shader constant folding treats unsigned integer values as signed

#### Status

APIs Affected: OpenGL ES

Impacted driver versions: rOp0 - ...

Fixed driver version: No shipping fix.

#### Description

A shader compiler optimization that evaluates constants for constant folding can treat unsigned integer values as signed values, which changes the semantics of the operation and can result in an incorrectly optimized value being used. This erratum impacts literal constant values, and literal constants assigned to a *const* local variable.

#### Implications

The result of unsigned integer constant folding might be incorrect. If this causes an incorrect address to be calculated, this might result in context loss if it causes a memory fault.

#### Workaround

You can avoid this erratum by assigning impacted constants values to a non-const local variable, and using this variable in any subsequent computations in place of the literal value.

Using OpenGL ES *GL\_EXT\_robustness* can be used to reduce the number of instances of context loss caused by memory faults. However, in these circumstances rendering will still be incorrect as the address accessed inside the buffer will be incorrect.



## 3843108

### Atomic access coordinates for images and texture buffers always truncated to 16-bits

#### Status

APIs Affected: OpenGL ES, Vulkan

Impacted driver versions: r18p0 - r53p0

Fixed driver version: r54p0

#### Description

The compiler incorrectly truncates 32-bit coordinates used to make atomic accesses to images and texture buffers to 16 bits.

#### Implications

Atomics access coordinates will wrap and only use the bottom 16 bits of each coordinate, so incorrect data values will be returned for texel indices above 65535.

#### Workaround

You can avoid the issue by manually wrapping texel coordinates at 16-bit boundaries.

For image access, you can often implement wrapping in a single axis by making use of the 2D coordinate space available. For example, an image using 1M x 1 resolution can be converted into a 2D image using 64K x 16 resolution.

For texel buffers, which are fixed to use a single coordinate, the resource must be either split across multiple texel buffer bindings or converted to a 2D image.

## 3843237

### Vector bitwise of swizzle/combine returns incorrect result

#### Status

APIs Affected: OpenGL ES, Vulkan

Impacted driver versions: r18p0 - r53p0

Fixed driver version: r54p0

#### Description

During compilation the shader compiler can apply an incorrect code transform to vector operations that combine an integer bitwise operator with an implicit or explicit lane swizzle.

#### Implications

Integer bitwise operators on vector variables can give an incorrect result.

#### Workaround

You can avoid the issue by manually scalarizing the impacted operations. For example, rewrite this:

```
uvec2 a = uvec2(input1, 2u) | 1u;
```

... as ...

```
uvec2 a = uvec2(input1, 2u);  
a.y |= 1u;
```

## 3893903

### Dynamic render state incorrectly used when modified between two binds of a static pipeline

#### Status

APIs Affected: Vulkan

Impacted driver versions: r48p0 - r53p0

Fixed driver version: r54p0

#### Description

Draw calls using a Vulkan pipeline with a static render state may incorrectly use a dynamically set state value when the dynamic state is modified between two binds of the same pipeline. For example, in the following scenario:

1. Bind pipeline1 using a static state
2. Set the equivalent dynamic state
3. Rebind pipeline1
4. Draw

This erratum impacts the following dynamic states:

- `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT`
- `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT`
- `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT`

#### Implications

The application might encounter incorrect rendering.

#### Workaround

You can set the value of the impacted dynamic states to match the required static state in the bound pipeline.

Alternatively, you can avoid using these dynamic render states.

## 3893905

### Dynamic render state incorrectly inherited on pipeline change (2)

#### Status

APIs Affected: Vulkan

Impacted driver versions: r48p0 - r53p0

Fixed driver version: r54p0

#### Description

Draw calls using Vulkan pipelines with dynamic render states will not use the correct dynamic value when changing pipeline if the draw immediately before the pipeline change also used the same state dynamically. For example, in the following scenario:

1. Bind Pipeline1
2. Set dynamic state
3. Draw
4. Bind Pipeline2
5. Draw

... where *Pipeline1* and *Pipeline2* are both pipelines using one of the impacted dynamic states, the second draw will not use the correct dynamically set render state value.

This erratum impacts the following dynamic states:

- `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT`
- `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT`

#### Implications

The application might encounter incorrect rendering.

#### Workaround

You can reset the value of all impacted dynamic states after binding a new pipeline.

Alternatively, you can use static pipeline render state.

## 3893908

### Declaring NxM matrices as row\_major reports incorrect buffer size

#### Status

APIs Affected: OpenGL ES

Impacted driver versions: r18p0 - r52p0

Fixed driver version: r53p0

#### Description

Buffer sizes will be reported incorrectly when specifying *layout(row\_major)* on a non-square matrix variable in a buffer.

For example, in the code below, the *GL\_UNIFORM\_BLOCK\_DATA\_SIZE* will be reported as 32 bytes when it should be 48.

```
layout(std140) uniform Block {  
    layout(row_major) mediump mat2x3 var;  
};
```

#### Implications

The application might encounter incorrect rendering if the application uses the buffer size returned by the query to size the allocated buffer. The actual buffer layout used by the shader compiler is correct, and the shader will function correctly if the allocated buffer size matches the specification requirements.

#### Workaround

You can specify the layout qualifier on the interface block instead of the member variables. For example:

```
layout(std140, row_major) uniform Block {  
    mediump mat2x3 var;  
};
```

Alternatively, you can use *layout(column\_major)* matrices.

## 3893910

### Declaring matrices in struct or array as *row\_major* reports incorrect buffer size

#### Status

APIs Affected: OpenGL ES

Impacted driver versions: r18p0 - r54p0

Fixed driver version: r54p1

#### Description

Buffer sizes will be reported incorrectly when specifying *layout(row\_major)* on a variable in a buffer when the member is an array of matrices, or the member is nested inside a structure.

For example, in the code below, the *GL\_UNIFORM\_BLOCK\_DATA\_SIZE* will be reported as 96 bytes when it should be 144.

```
layout(std140) uniform Block {  
    layout(row_major) highp mat2x3 var[3];  
};
```

#### Implications

The application might encounter incorrect rendering if the application uses the buffer size returned by the query to size the allocated buffer. The actual buffer layout used by the shader compiler is correct, and the shader will function correctly if the allocated buffer size matches the specification requirements.

#### Workaround

You can specify the layout qualifier on the interface block instead of the member variables. For example:

```
layout(std140, row_major) uniform Block {  
    highp mat2x3 var[3];  
};
```

Alternatively, you can use *layout(column\_major)* matrices.

## 3893911

### vkQueuePresentKHR() can hang if no vkQueueSubmit() calls were made

#### Status

APIs Affected: Vulkan

Impacted driver versions: r37p0 - r42p0, r44 - r49p0

Fixed driver version: r43p0, r49p1

#### Description

If an application repeatedly calls *vkQueuePresentKHR()* on a queue without any *vkQueueSubmit()* calls on the same queue, the application will hang.

#### Implications

The call to *vkQueuePresentKHR()* will hang.

#### Workaround

You can change the call to *vkQueuePresentKHR()* to use the queue that is used for rendering workloads and therefore also uses *vkQueueSubmit()* as part of normal operation.

Alternatively, if you have a dedicated present queue, you can add an empty *vkQueueSubmit()* to the queue before calling *vkQueuePresentKHR()*.

### Category B (rare)

There are no errata in this category.

## Category C

3817843

### Vulkan VK\_ARM\_scheduling\_controls feature query always reports false

#### Status

APIs Affected: Vulkan

Impacted driver versions: r47p0 - r49p2, r50p0 - r52p0

Fixed driver versions: r49p3, r53p0

#### Description

The *VK\_ARM\_scheduling\_controls* extension feature availability check is incorrectly implemented in *vkGetPhysicalDeviceProperties2()* instead of *vkGetPhysicalDeviceFeatures2()*. When applications use *vkGetPhysicalDeviceFeatures2()* the extension availability is always reported as *VK\_FALSE*.

#### Implications

The application feature query will incorrectly indicate that *VK\_ARM\_scheduling\_controls* is not available. This loss of functionality should have no impact on production applications.

#### Workaround

If the extension is supported in the list of available extensions you can assume that the feature is available, and attempt to use it. If it is disabled you may not see the shader core scheduling configuration take impact.

Alternatively, on impacted driver versions, you can use the *vkGetPhysicalDeviceProperties2()* function to query extension availability.



# Proprietary notice

This document is protected by copyright and other related rights and the use 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 Limited ("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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

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

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(PRE-1121-V1.0)

# Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in the Arm documents.

## Product status

All products and Services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

### Product completeness status

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

### Product revision status

The rxpy identifier indicates the revision status of the product described in this manual, where:

#### **rx**

Identifies the major revision of the product.

#### **py**

Identifies the minor revision or modification status of the product.