Arm GPU Errata for Application Developers

Software Developer Errata Notice

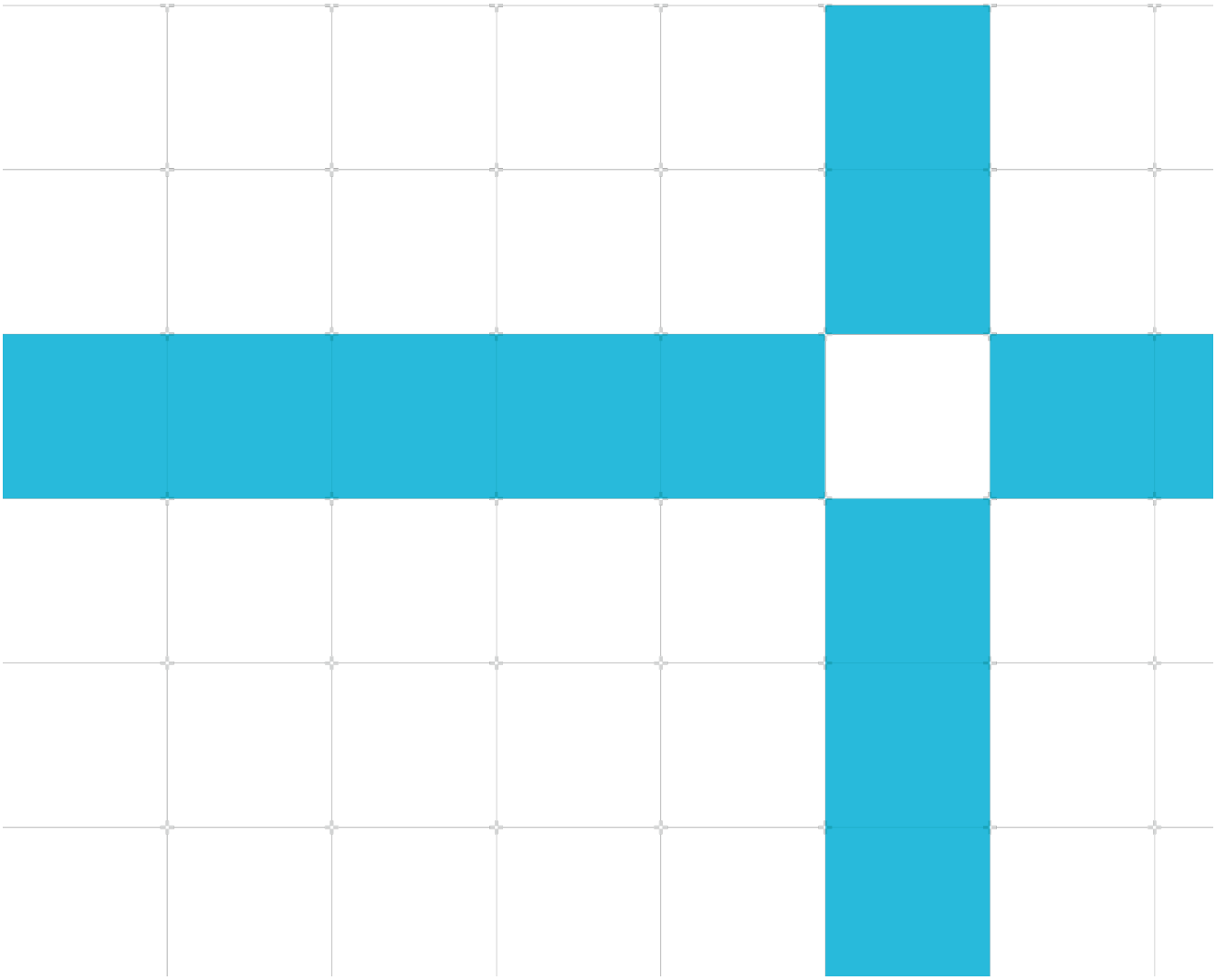
Date of issue: November 14, 2024

Non-Confidential

Document version: 1.0

Document ID: SDEN-3735689

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



This document (SDEN_3735689_1.0_en) was issued on November 14, 2024.

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**.

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

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:

Category A	A critical error. No workaround is available or workarounds are impactful. The error is likely to be common for many systems and applications.
Category A (Rare)	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.
Category B	A significant error or a critical error with an acceptable workaround. The error is likely to be common for many systems and applications.
Category B (Rare)	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.
Category C	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.

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

ID	Status	Area	Category	Summary
3787671	New	Programmer	Category A (rare)	Shader scalar integer right shifts return an incorrect result
3734951	New	Programmer	Category B	Freeing command buffers without explicit release causes memory leaks
3786926	New	Programmer	Category B	vkCmdBeginRendering() causes rendering artifacts or DEVICE_LOST
3787459	New	Programmer	Category B	Incorrect rendering when input gl_Position is declared outside of an input block
3787677	New	Programmer	Category B	Render pass loadOp will return black for 3D images that are 32x32x1 or smaller
3779191	New	Programmer	Category B	Read-only storage buffer accesses in inactive control flow are speculatively executed
3779215	New	Programmer	Category B	Shader clamp of conditional values with a zero limit produces incorrect code
3781413	New	Programmer	Category B	Rendering large amounts of geometry causes rendering artifacts or DEVICE_LOST
3781554	New	Programmer	Category B	Unreferenced vertex indices are speculatively shaded
3785718	New	Programmer	Category B	Pipeline barriers are not transitive when intermediate stage is HOST
3786496	New	Programmer	Category B	vkCmdBindVertexBuffers2() updates dynamic strides for an incorrect binding
3786517	New	Programmer	Category B	vkCmdSetCullMode() incorrectly culls non-triangle topologies
3786532	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
3787671	Programmer	Category A (rare)	Shader scalar integer right shifts return an incorrect result	r42p0 - r47p0	r48p0
3734951	Programmer	Category B	Freeing command buffers without explicit release causes memory leaks	r48p0, r49p0	r49p1, r50p0
3779191	Programmer	Category B	Read-only storage buffer accesses in inactive control flow are speculatively executed	r29p0 - ...	Open
3779215	Programmer	Category B	Shader clamp of conditional values with a zero limit produces incorrect code	r38p1 - r49p0, r50p0 - r51p0	r49p1, r52p0
3781413	Programmer	Category B	Rendering large amounts of geometry causes rendering artifacts or DEVICE_LOST	r0p0 - ...	Open
3781554	Programmer	Category B	Unreferenced vertex indices are speculatively shaded	r0p0 - ...	Open
3785718	Programmer	Category B	Pipeline barriers are not transitive when intermediate stage is HOST	r41p0 - ...	Open
3786496	Programmer	Category B	vkCmdBindVertexBuffers2() updates dynamic strides for an incorrect binding	r42p0 - r47p0	r48p0
3786517	Programmer	Category B	vkCmdSetCullMode() incorrectly culls non-triangle topologies	r41p0 - r51p0	r52p0
3786532	Programmer	Category B	Freeing simultaneous use command buffers causes memory corruption	r46p0 - r50p0	r49p1, r51p0
3786926	Programmer	Category B	vkCmdBeginRendering() causes rendering artifacts or DEVICE_LOST	r49p1 - r51p0	r52p0
3787459	Programmer	Category B	Incorrect rendering when input gl_Position is declared outside of an input block	r19p0 - r46p0	r47p0
3787677	Programmer	Category B	Render pass loadOp will return black for 3D images that are 32x32x1 or smaller	r29p0 - r38p1	r39p0

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 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 version: 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 onwards
Fixed driver version: No shipping fix

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 - r51p0
Fixed driver version: 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 - r51p0
Fixed driver version: 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 *vkCmdBeginRenderingEXT()*.

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.

Category B (rare)

There are no errata in this category.

Category C

There are no errata in this category.

Proprietary notice

(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.