Mali-400 MP Symbian OpenGL ES DDK (DX920) **Errata Notice**

This document contains all errata known at the date of issue in supported releases up to and including revision r2p0 of Mali-400 MP Symbian OpenGL ES DDK

Proprietary notice

Document confidentiality status

This document is Non Confidential.

Web address

http://www.arm.com/

Feedback on the product

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

- The product name
- A concise explanation of your comments.

Feedback on this document

If you have any comments on about this document, please send email to mailto:errata@arm.com giving:

- The document title
- The documents number
- The page number(s) to which your comments refer
- A concise explanation of your comments

General suggestion for additions and improvements are also welcome.

Contents

Introduction

Scope

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

- a unique defect tracking identifier
- the current status of the defect
- where the implementation deviates from the specification and the conditions under which erroneous behavior occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a 'work-around' where possible

Categorisation of Errata

Errata recorded in this document are split into three levels of severity:

Category 1 Behavior that is impossible to work around and that severely restricts the use of the product in all, or the majority of applications, rendering the device unusable.

Category 2 Behavior that contravenes the specified behavior and that might limit or severely impair the intended use of specified features, but does not render the product unusable in all or the majority of applications.

Category 3 Behavior that was not the originally intended behavior but should not cause any problems in applications.

Change Control

14 Apr 2010: Changes in Document v2

Page	Status	ID		Cat	Summary
11	New	724848	EGL	Cat 1	Mapped framebuffer is not completely unmapped
12	New	724859	EGL	Cat 1	EGL window surface resize OOM issue
13	New	724862	EGL	Cat 1	EGL pixmap readback OOM issue
14	New	724919	EGL	Cat 1	EGL_BUFFER_PRESERVED causing semaphore lock-up
15	New	724920	Shared	Cat 1	Potential deadlock between direct and deferred surface writes
17	New	724933	Shared	Cat 1	Potential deadlock between multiple framebuilder flush
18	New	725065	Base	Cat 1	Driver does not handle reset of the cores while the MMU is in Pagefault mode
20	Updated	716303	OpenGLES	Cat 2	Missing SW workaround for HW issue 711869
21	New	724508	OpenGLES	Cat 2	glCopyTexSubImage does not support 4444 and 5551 formats
22	New	724514	OpenGLES	Cat 2	GLES 1: Compressed textures do not disable texture unit channels
23	New	724595	ESSL	Cat 2	GLES 2: Internal compiler error on certain combinations of vector constructors and type conversions
24	New	724627	OpenGLES	Cat 2	Missing SW workaround for issue 725058
25	New	725067	Base	Cat 2	Wait after core reset in device driver may not be long enough
26	New	725069	Base	Cat 2	Mali memory allocation might fail if a specified OS memory region is not a multiple of 256KB
27	New	725070	Base	Cat 2	The Mali job dump system will program the incorrect MMU for pixel processor jobs
28	New	725071	Base	Cat 2	Heap growth not supported by Mali job dump system
32	Updated	716774	OpenGLES	Cat 3	Modifying an EGLimage created from a pixmap within a frame can give wrong result
37	Updated	717175	ESSL	Cat 3	GLES 2: Shader compilation may fail when the client application sets the locale to non-US
41	New	724530	OpenGLES	Cat 3	Vertex attribute stride queries return actual stride, not specified stride
42	New	724849	EGL	Cat 3	eglGetProcAddress can return wrong client API function pointer
43	New	724850	EGL	Cat 3	bind-to-texture flags wrong on configs
44	New	724906	OpenGLES	Cat 3	Rebinding of texture objects not handled properly
45	New	724937	OpenGLES	Cat 3	glBindFramebuffer should not early out
46	New	724939	OpenGLES	Cat 3	glBindBuffer should not early out
47	New	725062	Base	Cat 3	Crash when handling spurious MMU page fault interrupts
48	New	725076	OpenGLES	Cat 3	GLES2: Insufficient SW workaround for issue 725058

15 Jun 2009: Changes in Document v1

Page	Status	ID		Cat	Summary
20	New	716303	OpenGLES	Cat 2	Missing hardware workaround: Points drawn after a zero-size triangle may disappear
30	New	716148	OpenGLES	Cat 3	Retrieving large integer uniform values may lose precision
31	New	716740	EGL	Cat 3	EGLImage: Missing check for whether pbuffer is bound through eglBindTexImage
32	New	716774	OpenGLES	Cat 3	Modifying an EGLimage created from a pixmap within a frame can give wrong result
33	New	716781	OpenGLES	Cat 3	Creation of EGL Images from Mipmap level 10 and above disallowed
34	New	716872	OpenGLES	Cat 3	glDrawElements is limited to 24bit index count
35	New	716873	OpenGLES	Cat 3	Previously issued draw-calls may be lost when running out of memory
36	New	716874	Base	Cat 3	Too many jobs are dumped when dumping is combined with instrumentation
37	New	717175	ESSL	Cat 3	GLES 2: Shader compilation may fail when the client application sets the locale to non-US
38	New	717176	ESSL	Cat 3	GLES 2: Shader compiler preprocessor accepts some illegal combinations of preprocessor directives
39	New	717177	ESSL	Cat 3	GLES 2: Compilation of a vertex shader may fail with the error "Register allocation failed for vertex shader"
40	New	717178	OpenGLES	Cat 3	GLES2: Shader linker does not do invariance checks for built-in varyings

Errata Summary Table

The errata associated with this product affect product versions as below.

A cell shown thus **X** indicates that the defect affects the revision shown at the top of that column.

ID		Cat	Summary of Erratum	r1p0	r1p1
725065	Base	Cat 1	Driver does not handle reset of the cores while the MMU is in Pagefault mode	X	
724933	Shared	Cat 1	Potential deadlock between multiple framebuilder flush	X	
724920	Shared	Cat 1	Potential deadlock between direct and deferred surface writes	X	
724919	EGL	Cat 1	EGL_BUFFER_PRESERVED causing semaphore lock-up	X	
724862	EGL	Cat 1	EGL pixmap readback OOM issue	X	
724859	EGL	Cat 1	EGL window surface resize OOM issue	X	
724848	EGL	Cat 1	Mapped framebuffer is not completely unmapped	X	
725071	Base	Cat 2	Heap growth not supported by Mali job dump system	X	X
725070	Base	Cat 2	The Mali job dump system will program the incorrect MMU for pixel processor jobs	X	
725069	Base	Cat 2	Mali memory allocation might fail if a specified OS memory region is not a multiple of 256KB	X	
725067	Base	Cat 2	Wait after core reset in device driver may not be long enough	X	
724627	OpenGLES	Cat 2	Missing SW workaround for issue 725058	X	
724595	ESSL	Cat 2	GLES 2: Internal compiler error on certain combinations of vector constructors and type conversions	X	
724514	OpenGLES	Cat 2	GLES 1: Compressed textures do not disable texture unit channels	X	
724508	OpenGLES	Cat 2	glCopyTexSubImage does not support 4444 and 5551 formats	X	
716303	OpenGLES	Cat 2	Missing SW workaround for HW issue 711869	X	
725076	OpenGLES	Cat 3	GLES2: Insufficient SW workaround for issue 725058		X
725062	Base	Cat 3	Crash when handling spurious MMU page fault interrupts	X	
724939	OpenGLES	Cat 3	glBindBuffer should not early out	X	X
724937	OpenGLES	Cat 3	glBindFramebuffer should not early out	X	X
724906	OpenGLES	Cat 3	Rebinding of texture objects not handled properly	X	
724850	EGL	Cat 3	bind-to-texture flags wrong on configs	X	
724849	EGL	Cat 3	eglGetProcAddress can return wrong client API function pointer	X	
724530	OpenGLES	Cat 3	Vertex attribute stride queries return actual stride, not specified stride	X	X
717178	OpenGLES	Cat 3	GLES2: Shader linker does not do invariance checks for built-in varyings	X	X

ID		Cat	Summary of Erratum	r1p0	r1p1
717177	ESSL	Cat 3	GLES 2: Compilation of a vertex shader may fail with the error "Register allocation failed for vertex shader"	X	X
717176	ESSL	Cat 3	GLES 2: Shader compiler preprocessor accepts some illegal combinations of preprocessor directives	X	X
717175	ESSL	Cat 3	GLES 2: Shader compilation may fail when the client application sets the locale to non-US	X	X
716874	Base	Cat 3	Too many jobs are dumped when dumping is combined with instrumentation	X	
716873	OpenGLES	Cat 3	Previously issued draw-calls may be lost when running out of memory	X	X
716872	OpenGLES	Cat 3	glDrawElements is limited to 24bit index count	X	X
716781	OpenGLES	Cat 3	Creation of EGL Images from Mipmap level 10 and above disallowed	X	X
716774	OpenGLES	Cat 3	Modifying an EGLimage created from a pixmap within a frame can give wrong result	X	X
716740	EGL	Cat 3	EGLImage: Missing check for whether pbuffer is bound through eglBindTexImage	X	
716148	OpenGLES	Cat 3	Retrieving large integer uniform values may lose precision	X	X

Errata - Category 1

724848: Mapped framebuffer is not completely unmapped

Status

Affects: EGL

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

The framebuffer device is memory mapped upon initialization. The size of memory unmapped at termination time does not correspond to the memory mapped size.

Implications

You can run out of virtual memory address space when initializing and terminating the framebuffer device a large number of times.

Workaround

Close the display at application termination time.

724859: EGL window surface resize OOM issue

Status

Affects: EGL

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

An out of memory situation when resizing a window surface can leave the driver in a state where a crash is possible.

Implications

When resizing a windowed surface, the old resources should be kept until new resources has been successfully allocated. However, in some situations, the old resources are released before new resources have been successfully allocated.

This can lead to a driver crash later on, when operations are performed on released resources.

Workaround

Other than disabling window surface resize, none.

724862: EGL pixmap readback OOM issue

Status

Affects: EGL

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

When rendering to a native pixmap, EGL needs to read in the pixmap data. An out of memory situation during this read back process can potentially lead to a memory leak or driver crash.

Implications

Depending on where the actual out of memory situation occurs, you might have a memory leak or a driver crash. The memory leak is caused by missing memory cleanup, while the driver crash is caused by operations on released memory after the read back process has finished.

Workaround

None.

724919: EGL_BUFFER_PRESERVED causing semaphore lock-up

Status

Affects: EGL

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

Enabling preserved swap buffer behavior can cause a driver lock.

Implications

A deadlock can occur when using preserved swap behavior in combination with direct rendering. The deadlock can occur after calling eglSwapBuffers on such a surface.

Workaround

Use other means of preserving the color buffer. This can be accomplished by client API readback functions, or by rendering to textures.

724920: Potential deadlock between direct and deferred surface writes

Status

Affects: Shared

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

A deferred drawcall is a write operation happening through the Mali core. Any draw operation done through glDrawArrays and glDrawElements qualifies as a deferred write to the output surface.

A direct write is any surface draw operation happening immediately (and thus without the aid of Mali). A good example is a call to glTexSubImage2D, but any operation directly modifying the surface pixel values qualifies.

When drawing to a surface, the driver locks the surface - either to modify the pixel data (direct), or to notify the surface that there is an outstanding write in the pipeline (deferred).

A deferred draw will lock down all the surfaces the drawcall in question is rendering to. If you are writing to an FBO with a depth and color buffer, then both the depth and the color surface will be locked in a predetermined order. Attempts to flush the FBO will also lock down the same two buffers in the same predetermined order to ensure that nothing else has written to the surfaces in the meantime.

A direct draw will lock down the surface being written to. If there is an outstanding deferred write to that surface, the deferred write operation will be flushed/finished prior to the direct write taking place.

This is a potential deadlock situation, as direct writes first locks the direct surface, then flush outstanding writes (which may lock other surfaces). The combined set of surface locks in this operation do not happen in the safe predetermined order, and is as such potentially unsafe. The gles driver design prevents most any case from actually taking place, but the EGL Image extension omits many of the safety guards. The following example is still possible:

```
1: Thread/Context A creates a depth renderbuffer (surface 1) 2: Thread/Context A
also creates a color texture/renderbuffer (surface 2)

3: Thread/Context A sets up an FBO to render to these two. 4: The color surface
(2) is shared through an EGL Image

5: Thread/Context B binds up the EGL image (2) as a new texture in its context

6: Thread/Context A adds a deferred write job the FBO (glDraw)

  - Both surfaces are marked with an outstanding write. 7: Thread/Context B
attempts a direct write to the color surface (1).   - This will lock surface (2)

  - Surface 1 has an outstanding write, which will be attempted flushed

  - This will lock the surfaces in the FBO (1, 2), except for 2 which is already
locked.   - The flush will thus only lock surface (1).

  - Locking order of the entire operation is: (2, 1)

8: At the same time, Thread/Context A adds another deferred write job to both
surfaces.   - This will lock both surfaces in a predetermined order (1, 2)
```

Implications

It is possible through the use of the EGL Image extension, to lock a set of surfaces in a different order in two threads at the same time. This will deadlock the driver.

Workaround

Instead of sharing the surface through an EGL image, share the surface through a GLES context sharing. Another option is to stick to only deferred write operations.

724933: Potential deadlock between multiple framebuilder flush

Status

Affects: Shared

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

A flush may happen as a result of a call to glFlush/glFinish/eglSwapBuffers, or as an implicit action when needing the content of a surface. The flush will lock down the framebuilder, then lock all surfaces the framebuilder is rendering to. In that order.

A surface write will lock down the surfaces, then lock down the framebuilder. Notably, in the opposite order. This is a potential deadlock which can be triggered by the following chain of commands:

```
1: Thread/Context A draws to a texture through an FBO. 2: An EGL Image is made
from the texture.

3: Thread/Context B creates a new texture from the EGL Image. 4: Thread/Context B
does a direct write to the texture.  - This locks the surface (1)

 - The surface has an outstanding draw operation on it

 - The draw operation is flushed by flushing the FBO in Thread/Context A

 - This flush attempts to lock the frame (2)

5. At the same time, Thread/Context A does a glFlush.  - This will attemt to lock
the frame (1) then the surface (2)
```

Implications

Different types of flushes of the same framebuilder at the same time may deadlock the driver.

Workaround

By not sharing surfaces through EGL images (use the gles context sharing) or at least not putting deferred drawcalls on EGL Image surfaces will eliminate the possibility of this deadlock.

725065: Driver does not handle reset of the cores while the MMU is in Pagefault mode

Status

Affects: Base

Fault status: Cat 1, Present in: r1p0, Fixed in r1p1.

Description

The Mali device driver is not able to handle both a Mali MMU page fault and core reset at the same time if that process has a Mali job running on a core. This can typically happen if a process is forcefully terminated by the operating system, like when pressing CTRL-C. The reason for this is that during program termination, the operating system removes the memory from the Mali MMU while a job might still be running, and then reset the cores. The errata will trigger if the Mali MMU page fault is present but not handled before core reset is completed.

Implications

The affected Mali core(s) will not be working, a system reboot is needed to resolve the issue.

Workaround

None.

Errata - Category 2

716303: Missing SW workaround for HW issue 711869

Status

Affects: OpenGLES

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

Hardware issue 711869 identifies an issue with Mali400 r0p0 and r0p1, where after issuing a drawcall with one or more zero-sized triangles, subsequent drawcalls with points may disappear.

There is no automatic workaround in the driver for this issue, which means the applications have to work around it manually.

Implications

Points drawn after a drawcall which draws zero-sized triangles may disappear, causing rendering errors.

Workaround

Whenever a drawcall using triangles at the risk of being zero-sized is followed by a drawcall using points, separate the two calls by a fullscreen transparent quad.

724508: glCopyTexSubImage does not support 4444 and 5551 formats

Status

Affects: OpenGLES

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

Doing a glCopyTexSubImage into a texture of format RGBA4444 or RGBA5551 will fail. For debug builds, this will trigger an assert crash. Release build drivers will ignore the glCopyTexSubImage command and return a texture filled with garbage.

Implications

Using glCopyTexSubImage2D on textures with these formats is not possible.

Workaround

Use a different texture format, render to the texture using an FBO (GLES2 only), or call glReadpixels to retrieve the current framebuffer, convert the retrieved buffer to your required format, and re-upload the data with glTexImage2D.

724514: GLES 1: Compressed textures do not disable texture unit channels

Status

Affects: OpenGLES

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

The OpenGLES 1.1 specification states that textures units should only do its operation on channels present in the texture bound to the texture unit in question. Channels in this context refer to "RGB" or "Alpha".

For example, a texture unit set to a GL_REPLACE operation should replace the texture unit input with the texture bound. But if that bound texture doesn't have an alpha channel, the alpha channel is not replaced. Similarly, if the texture doesn't have an RGB channel, the RGB input of the texture unit is not replaced. This is how texture units operate on all uncompressed texture formats.

However, all compressed texture formats are being treated as having all channels present, regardless of texture format. For example, The ETC1_RGB_888 format would for example be treated as a texture with an alpha channel, despite that not being the case. As a result, if this texture was bound to a texture unit, it would also replace the alpha channel.

Implications

OpenGL ES 1.1 applications depending on the texture unit channel disabling working correctly (when using compressed textures) will see erroneous output.

Workaround

Instead of using the default combiner modes, use the GL_COMBINE combiner mode. This mode allows the application developer to directly control the operation of each channel in the texture unit, including the ability to disable them.

Example:

Instead of setting up a texture unit with mode=GL_REPLACE (like this):

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

Set it up with a mode=GL_COMBINE (like this):

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_REPLACE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_ALPHA, GL_REPLACE);
glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND0_RGB, GL_SRC_COLOR);
glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND0_ALPHA, GL_SRC_ALPHA);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB, GL_TEXTURE);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_ALPHA, GL_TEXTURE);
```

If it is required to disable a specific channel, do so explicitly by replacing the relevant channel source GL_TEXTURE with GL_PREVIOUS.

724595: GLES 2: Internal compiler error on certain combinations of vector constructors and type conversions

Status

Affects: ESSL

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

In fragment shaders, certain combinations of vector constructors and type conversions where a value is used more than once and some components of the result are never used, may cause the ESSL compiler to report an internal compiler error.

For instance, the following fragment shader triggers the error:

```
precision mediump float;
varying vec2 v;
uniform sampler2D sam;
void main()
{
        ivec3 a = ivec3(texture2D(sam, v));
        a.xz.y = 2;
        gl_FragColor = vec4(bvec4(true, false, a).b);
}
```

The ESSL compiler will exit gracefully and will not crash or silently generate invalid code.

Implications

Certain rare combinations of ESSL language constructs may cause the compiler to report an error for valid code.

Workaround

A safe way to avoid triggering the internal compiler error is to never construct vectors where only some of the components are ever used. Any shader which violates this principle can easily be changed to adhere to it with no change in functionality.

724627: Missing SW workaround for issue 725058

Status

Affects: OpenGLES

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

This errata tracks that hardware issue 725058 is missing a workaround. Hardware issue 725058 may cause the last attribute stream sent to the hardware to be corrupted, if its data cross a 16 byte boundary. This issue only affects Mali400-MP revision R0P0 and R0P1.

The compiler and linker may layout GLES attributes on the hardware as it sees fit based on datatypes, alignment and aliasing requirements as given by glBindAttribLocation. It is not straighforward to determine which GLES attribute that corresponds to a given hardware attribute.

Implications

The last vertex attribute sent to the hardware, should its data cross a 16byte boundary, will contain garbage data when read by the vertex shader. This will lead to visual rendering errors.

Workaround

By not using 3 component streams, this issue will not trigger. Use vec4 attribute streams instead and ignore the fourth component, and do not use non-default attribute strides. This will ensure that all attribute streams contain data that do not cross a 16 byte boundary.

725067: Wait after core reset in device driver may not be long enough

Status

Affects: Base

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

The device driver does a fixed number of Mali register reads in order to wait for the a Mali core to be ready after a reset. In some systems, this might not be long enough, and the device driver will start using the core before it has completed the reset.

Implications

The setup of the Mali core after reset will not be correctly executed, and unexpected behavior might happen. The core will most likely be rendered unusable, and applications trying to use Mali will hang.

Workaround

This errata can be fixed by replacing the fixed number of dummy register reads with a small for loop after issuing the MALIGP2_REG_VAL_CMD_RESET to the Mali GP management command register and after issuing MALI200_REG_VAL_CTRL_MGMT_FORCE_RESET to the Mali PP management command register.

For the Mali geometry processor, do the following changes:

1) Write the value 0xC0FFE000 to the register MALIGP2_REG_ADDR_MGMT_WRITE_BOUND_LOW right before issuing the MALIGP2_REG_VAL_CMD_RESET command.

2) Replace the dummy register reads after issuing the MALIGP2_REG_VAL_CMD_RESET with a for loop doing a maximum of 15 iterations.

3) In the for loop; write the value 0xC01A0000 to the MALIGP2_REG_ADDR_MGMT_WRITE_BOUND_LOW register, and read the same register afterward. If you get the same value (0xC01A0000), then you can exit the for loop and continue.

4) After the for loop, write the value 0x00000000 to the MALIGP2_REG_ADDR_MGMT_WRITE_BOUND_LOW register.

Do the same for the Mali pixel processor code, except use the MALI200_REG_ADDR_MGMT_WRITE_BOUNDARY_LOW register instead of the MALIGP2_REG_ADDR_MGMT_WRITE_BOUND_LOW register.

725069: Mali memory allocation might fail if a specified OS memory region is not a multiple of 256KB

Status

Affects: Base

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

The documentation fails to mention a limitation for the OS_MEMORY type that can be specified in the config.h file used to build the Mali device driver. The size field must be a multiple of 256KB.

The support for OS_MEMORY has been dropped for the r1p1 release.

Implications

If the OS_MEMORY section size isn't a multiple of 256KB and there is a MEMORY section defined after it and there is an allocation request that cannot be fully fulfilled by the incorrectly configured OS_MEMORY section, then that allocation will fail, even though there are free memory to fulfill the request.

Workaround

Make sure any OS_MEMORY section size is a multiple of 256KB and/or make sure the OS_MEMORY section is defined last in the config.h file.

725070: The Mali job dump system will program the incorrect MMU for pixel processor jobs

Status

Affects: Base

Fault status: Cat 2, Present in: r1p0, Fixed in r1p1.

Description

When the MMU programming steps for a Mali pixel processor job are written into the config.txt file, then the addresses of the MMU for the Mali geometry processor is used, instead of the addresses of the MMU for the Mali pixel processor.

Implications

Playback of dump cannot be done with MMU support.

Workaround

Manually edit the config.txt file and adjust the MMU register addresses to match the one of the MMU for the pixel processor desired, or don't use MMU during playback.

725071: Heap growth not supported by Mali job dump system

Status

Affects: Base

Fault status: Cat 2, Present in: r1p0,r1p1, Open.

Description

New MMU tables is not dumped when dumping a Mali geometry processor which is resumed after being given more heap memory.

Implications

The dump might contain an incomplete MMU table, and thus cause page faults during playback.

Workaround

Don't use MMU support during dumping or playback.

Errata - Category 3

716148: Retrieving large integer uniform values may lose precision

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

The GLES driver stores all uniforms as FP32 floats. This includes integer uniforms, which are transformed to FP32 floating point values when specified by glUniform. This conversion is within legal bounds as far as GLES is concerned, and Mali requires all uniforms to be converted to floating point. But this scheme has the side effect that glGetUniform will access your uniforms with a lower precision than the one originally set.

Implications

Integer uniforms retrieved with glGetUniform will differ from the uniform value set with glUniform.

Workaround

Track integer uniforms on the application side instead of retrieving them with glGetUniform

716740: EGLImage: Missing check for whether pbuffer is bound through eglBindTexImage

Status

Affects: EGL

Fault status: Cat 3, Present in: r1p0, Fixed in r1p1.

Description

eglCreateImageKHR is missing a check for whether a pbuffer is bound through eglBindTexImage, and will therefore succeed even if the pbuffer is bound.

Implications

Creation of EGLImage will succeed, even if the supplied pbuffer is bound through eglBindTexImage. No other impact.

Workaround

Unbind the pbuffer using eglReleaseTexImage before creating an EGLImage.

716774: Modifying an EGLimage created from a pixmap within a frame can give wrong result

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

If a texture is created from an EGLimage that was created from a pixmap and that texture is used for rendering and then modified in the same frame, the rendering will use the modified texture instead of the original.

Implications

The modified texture is used for draw-calls that happened before the modification took place.

Workaround

716781: Creation of EGL Images from Mipmap level 10 and above disallowed

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

Mali requires the driver to pack the data in mipmap level 10 and above together into one memory allocation. The driver does this for normal textures as required, but this repacking scheme collides with the EGL requirement that EGLImages cannot be re-located in memory.

Implications

There is no way of currently creating an EGLImage from a GLES texture, mipmap level 10 and above.

Workaround

none

716872: glDrawElements is limited to 24bit index count

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

glDrawElements does not support an index count larger that 2^24-1.

Implications

The index count in glDrawElements will be limited to 2^24-1, meaning that indices will be discarded if the count is above this value.

Workaround

If an index count of 2^24 or more is needed, the call to glDrawElements should be split into several calls with index count less than 2^24.

716873: Previously issued draw-calls may be lost when running out of memory

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

If more than 100 GL calls that flush the pipeline (glReadPixels, glFlush and glFinish) are interleaved with drawing calls (this can be any combination of glDrawElements, glDrawArrays, and glClear without all buffers bits or with scissoring enabled) are issued without any calls to glClear (with all buffer bits set and scissoring disabled) or eglSwapBuffers in between, and the driver runs out of memory, the result of previously issued drawing calls can be lost.

Implications

Previously rendered content is lost.

Workaround

Avoid excessive calls to glReadPixels, and avoid all calls to glFlush. glFlush is called implicitly at eglSwapBuffers.

716874: Too many jobs are dumped when dumping is combined with instrumentation

Status

Affects: Base

Fault status: Cat 3, Present in: r1p0, Fixed in r1p1.

Description

Enabling dumping while more than two pixel processor performance counters are enabled causes the same job to be dumped multiple times (once for every two enabled counters).

Implications

The same pixel processor job will be dumped several times.

Workaround

Don't enable dumping and instrumentation of pixel processor at the same time.

717175: GLES 2: Shader compilation may fail when the client application sets the locale to non-US

Status

Affects: ESSL

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

The ESSL compiler uses strtod() for parsing floating point literals. Therefore, if the online compiler is used with an OpenGL ES application and the application either sets the locale explicitly to a locale with different numerical conventions than the US with e.g.

setlocale(LC_NUMERIC, "nb_NO");

or picks up the current locale from environment variables with e.g.

setlocale(LC_NUMERIC, "");

and the environment variables are set in a way that indicates a locale with adifferent numerical convention than the US, the parser will fail with errors such as

0:28: L0001: Error while parsing floating point literal '0.0'

This is because the strtod() function now is looking for literals with a different decimal separator.

Implications

Online compilation of valid shaders using floating-point literals may fail.

Workaround

Either use the off-line compiler and load shader binaries, or surround each call to glCompileShader with setlocale calls, e.g.

char *prev_locale = setlocale(LC_NUMERIC, "C"); /* reset numeric locale to default,

save the old locale */

glCompileShader(...);

setlocale(LC_NUMERIC, prev_locale); /* restore the old locale */

717176: GLES 2: Shader compiler preprocessor accepts some illegal combinations of preprocessor directives

Status

Affects: ESSL

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

The ESSL compiler may in some cases accept shaders containing illegal combinations of #if / #ifdef / #ifndef / #elif / #else / #endif as valid.

Implications

This defect has no implications for valid programs.

Workaround

No workaround needed.

717177: GLES 2: Compilation of a vertex shader may fail with the error "Register allocation failed for vertex shader"

Status

Affects: ESSL

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

MaliGP2 has only limited internal bandwidth between its registers and execution units. In some rare cases, the register allocator for MaliGP2 in the shader compiler runs into a situation where there are more operations executed in one cycle than can be fed from the registers simultaneously. The compiler aborts the compilation in this case.

Implications

Some very big or very complex vertex shaders fail to compile.

Workaround

Changing the shader code slightly will often eliminate the problem. Try to rewrite some of the shader to do the calculations differently.

717178: GLES2: Shader linker does not do invariance checks for built-in varyings

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

Section 4.6.4 of the OpenGL ES 2.0 specification, Invariance and Linkage, requires invariance checking between the built-in shader varyings gl_FragCoord, gl_Position, gl_PointCoord, gl_PointSize, and gl_FrontFacing. This is not being done by the linker. Instead, all varyings are being treated as if they were invariant.

Implications

This defect has no implications for valid programs.

Workaround

Do not declare any of the built-in varyings invariant.

724530: Vertex attribute stride queries return actual stride, not specified stride

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

When specifying an attribute stride as 0, the GLES driver will calculate the actual minimum stride based on datatype and component count. Retrieving the stride through glGet should however return the number specified when setting up the stream, not the calculated stride.

The gles driver erroneously returns the actual stride, and not the specified stride.

Implications

The following sequence should set "value" to 0, but is setting it to 16 instead.

glVertexPointer(4, GL_FLOAT, 0, vertices);

glGetIntegerv(GL_VERTEX_ARRAY_STRIDE, &value);

Workaround

Don't query the driver for the attribute stride, unless the actual stride is an acceptable return value for your application. Applications truly needing to retrieve the number 0 in this case should track this value manually.

724849: eglGetProcAddress can return wrong client API function pointer

Status

Affects: EGL

Fault status: Cat 3, Present in: r1p0, Fixed in r1p1.

Description

eglGetProcAddress is used to retrieve client API extension function pointers. In some cases, EGL can return a function pointer which is invalid for the client API version to be used.

The following conditions has to be met in order for this to happen:

* Both OpenGL ES 1.x and OpenGL ES 2.x has to be available from within EGL

* eglGetProcAddress has to be queried for the glEGLImageTargetTexture2DOES extension pointer

If the following two conditions are met, then the returned extension pointer will be not be usable for OpenGL ES 2.x, since the OpenGL ES 1.x extension entrypoint is returned. glEGLImageTargetTexture2DOES is the only extension that is affected by this.

Implications

The returned extension pointer will not be valid for OpenGL ES 2.x. It will result in a NOP, since the two libraries can't be active at the same time from within EGL.

Workaround

One option is to have separate libraries for EGL/OpenGL ES 1.x and EGL/OpenGL ES 2.x. Another is to load the symbol of the extension manually by using for example libdl.

724850: bind-to-texture flags wrong on configs

Status

Affects: EGL

Fault status: Cat 3, Present in: r1p0, Fixed in r1p1.

Description

In the list of configs in EGL, all the GLES configs have both EGL_BIND_TO_TEXTURE_RGB and EGL_BIND_TO_TEXTURE_RGBA set to EGL_TRUE. However, an RGB config can not support binding to an RGBA texture, because there is no alpha channel available.

Implications

EGL claims support for binding an RGB surface to RGBA texture, which is not allowed.

Workaround

Do not bind an RGB surface to an RGBA texture. Use a surface that supports alpha.

724906: Rebinding of texture objects not handled properly

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0, Fixed in r1p1.

Description

When binding a texture target to an object already bound, the driver early out of the bind operation as it technically doesn't do anything. This is however not correct. When objects are deleted in a scenario with multiple contexts, the object is only deleted in the context where the delete operation took place. Other contexts may still have the texture object bound, but the object is no longer present in the object list.

A bind operation will need to check the object list, and should there be no object present, create a new one. In the scenario with multiple contexts, "rebinding" an object may thus result in a new object being created, as illustrated by this example:

1. Context A does BindTexture(GL_TEXTURE_2D, 1);
2. Context A does TexImage2D(GL_TEXTURE_2D, ...)
3. Context B does DeleteTextures(1, {1});
4. Context A does BindTexture(GL_TEXTURE_2D, 1);

The GLES spec will require step 4 to check the object list, notice that that the object list is empty and create a new texture object. The driver is earlying out of that operation, resulting in the old deleted texture object still being bound.

Implications

Bound texture objects may have a life-time longer than what is dictated by the GLES specification. Applications relying on the texture being a new empty texture after step 4 in the example above will work on false assumptions, which might in the worst case lead to the wrong (old) texture being used.

Workaround

Bind to texture object 0 before binding to the required texture object if this specific behavior is required. The "early out" will then not take place.

724937: glBindFramebuffer should not early out

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

When binding the framebuffer target to an object already bound, the driver early out of the bind operation as it technically doesn't do anything. This is however not correct. When objects are deleted in a scenario with multiple contexts, the object is only deleted in the context where the delete operation took place. Other contexts may still have the framebuffer object bound, but the object is no longer present in the object list.

A bind operation will need to check the object list, and should there be no object present, create a new one. In the scenario with multiple contexts, "rebinding" an object may thus result in a new object being created, as illustrated by this example:

1. Context A does BindFramebuffer(GL_FRAMEBUFFER, 1);
2. Context A does FramebufferTexture2D(GL_FRAMEBUFFER, ..., <sometexture>)
3. Context B does DeleteFramebuffer(1, {1});
4. Context A does BindFramebuffer(GL_FRAMEBUFFER, 1);

The GLES spec will require step 4 to check the object list, notice that that the object list is empty and create a new framebuffer object. The driver is earlying out of that operation, resulting in the old deleted framebuffer object still being bound.

Implications

Bound framebuffer objects may have a life-time longer than what is dictated by the GLES specification. Applications relying on the framebuffer object being a new empty object after step 4 in the example above will work on false assumptions, which might in the worst case lead to the wrong (old) set of attachments being rendered to.

Workaround

Bind to framebuffer object 0 before binding to the required framebuffer object if this specific behavior is required. The "early out" will then not take place.

724939: glBindBuffer should not early out

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p0,r1p1, Open.

Description

When binding a vertexbuffer target to an object already bound, the driver early out of the bind operation as it technically doesn't do anything. This is however not correct. When objects are deleted in a scenario with multiple contexts, the object is only deleted in the context where the delete operation took place. Other contexts may still have the framebuffer object bound, but the object is no longer present in the object list.

A bind operation will need to check the object list, and should there be no object present, create a new one. In the scenario with multiple contexts, "rebinding" an object may thus result in a new object being created, as illustrated by this example:

1. Context A does BindBuffer(GL_ARRAY_BUFFER, 1);
2. Context A does BufferData(GL_ARRAY_BUFFER, ...)
3. Context B does DeleteBuffer(1, {1});
4. Context A does BindBuffer(GL_ARRAY_BUFFER, 1);

The GLES spec will require step 4 to check the object list, notice that that the object list is empty and create a new vertexbuffer object. The driver is earlying out of that operation, resulting in the old deleted vertexbuffer object still being bound.

Implications

Bound vertexbuffer objects may have a life-time longer than what is dictated by the GLES specification. Applications relying on the vertexbuffer object being a new empty object after step 4 in the example above will work on false assumptions, which might in the worst case lead to the wrong (old) set of bufferdata being used.

Workaround

Bind to vertexbuffer object 0 before binding to the required vertexbuffer object if this specific behavior is required. The "early out" will then not take place.

725062: Crash when handling spurious MMU page fault interrupts

Status

Affects: Base

Fault status: Cat 3, Present in: r1p0, Fixed in r1p1.

Description

The Mali MMU page fault handling will try dereference a NULL pointer in order to access a mutex if a spurious page fault interrupt is received. That is, a page fault when there are no active cores behind the Mali MMU. This should not happen unless there is some HW malfunction.

Implications

A NULL pointer will be dereferenced in kernel space, and we will get a system crash.

Workaround

None.

725076: GLES2: Insufficient SW workaround for issue 725058

Status

Affects: OpenGLES

Fault status: Cat 3, Present in: r1p1, Open.

Description

The fix for issue 724626, which in turn works around HW issue 725058, is not flawless. This errata entry tracks the problem with the SW workaround for this HW issue.

HW issue 725058 (present on Mali400-MP revision R0P0 and R0P1) may corrupt the last attribute stream in situations where its data cross a 16 byte boundaries. Attribute streams using non-standard strides or a 3 component data type may result in this hardware issue triggering.

The workaround implemented is simply enough, just to add a dummy stream to the end of the attribtue stream table. If the HW issue triggers, only this dummy stream will be corrupted. But this workaround is not possible in situations where all attribute streams are already in use.

OpenGL ES 1.1 applications will at most use 14 streams, and can thus never end up in this situation. OpenGL ES 2.0 applications using all 16 streams will however be left without an automatic workaround for this problem.

Implications

OpenGL ES 2.0 applications triggering the defect described in HW issue 725058, and using all 16 streams, will get the last attribute stream corrupted. This will lead to visual rendering errors.

Workaround

OpenGL ES 2.0 applications need to limit itself to a maximum of 15 attribute streams, or avoid using attribute streams with non-default stride or datatypes of three components (like vec3 or mat3).

