Category:GPU

Contents

Stable: 26.02.2021 - 14:29 / Revision: 11.01.2021 - 17:21

Stable: 26.02.2021 - 14:29 | Revision: 11.01.2021 - 17:21

A quality version of this page, approved on *26 February 2021*, was based off this revision.

This category groups together all articles related to the Linux®**GPU** software frameworks.

Graphics Processing Units

Pages in category "GPU"

The following 7 pages are in this category, out of 7 total.

- GPU device tree configuration
- GPU troubleshooting grid
- How to launch glmark2 benchmark
- How to launch Khronos OpenGLES conformance tests
- How to monitor the GCNANO GPU load
- How to test and benchmark OpenGLES
- OpenVG overview

Stable: 26.02.2021 - 16:54 / Revision: 26.02.2021 - 16:53

A quality version of this page, approved on *26 February 2021*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the **GPU** internal peripheral when it is assigned to the Linux®OS.

The configuration is performed using the device tree mechanism that provides a hardware description of the GPU internal peripheral, used by the STM32 GPU Linux driver.

2 DT bindings documentation

The GPU is represented by the STM32 GPU device tree bindings [1].

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

The GPU device tree node is declared in stm32mp157.dtsi [2]. The declaration (shown below) defines the hardware registers base address, the interrupt, the clocks and the reset.

```
gpu: gpu@59000000 {
        compatible = "vivante,gc";
        reg = <0x59000000 0x800>;
        interrupts = <GIC_SPI 109 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&rcc GPU>, <&rcc GPU_K>;
        clock-names = "bus" ,"core";
        resets = <&rcc GPU_R>;
        status = "disabled";
};
```

Warning

This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

- Linux kernel bindings for "vivante,gc"
- Linux kernel STM32MP157 device tree (stm32mp157.dtsi)

Operating System

Graphics Processing Units

Device Tree

Generic Interrupt Controller

Serial Peripheral Interface

Stable: 04.10.2019 - 15:50 / Revision: 04.10.2019 - 15:49

A quality version of this page, approved on *4 October 2019*, was based off this revision.

Some typical issues related to the **GPU** framework are listed below. Solutions or debugging methods are proposed for these issues.

If your issue is not listed, try also looking in the articles in the GPU or troubleshooting grids categories.

Symptom	Resolution

`[ 45.679109] Unhandled fault: imprecise external abort (0xc06) at 0xae1a0000`
`[ 45.684628] pgd = d1e98000`
`[ 45.687334] [ae1a0000] *pgd=d1e9f835, *pte=f8000747, *ppte=f8000c37`
`[ 45.693842] Unhandled fault: imprecise external abort (0xc06) at 0xae1a0000`
`[ 45.700480] pgd = d1dc0000`
`[ 45.703156] [ae1a0000] *pgd=00000000`
	Maybe the **GPU reserved memory** area in the kernel device tree is not *aligned* with the board DDR memory size, please have a look at the "gpu_reserved" entry in the kernel device tree, and update it if necessary.

Graphics Processing Units

Doubledata rate (memory domain)

Stable: 03.10.2019 - 13:22 / Revision: 26.09.2019 - 12:54

A quality version of this page, approved on *3 October 2019*, was based off this revision.

Contents

1 Introduction

The purpose of this article is to describe how to build and execute the **Khronos OpenGL ES 2.0 Conformance Tests**.

These tests are provided by the Khronos Group [1]. They are available as open source from the **Khronos CTS GitHub source repo** [2].

For a detailed description of the Khronos OpenGL ES2.0 CTS, please refer to the **openglcts README.md** [3] file.

2 How to install khronos-cts

2.1 Building khronos-cts

Execute the following command in the OpenSTLinux build environment:

```
PC $> bitbake khronos-cts
```

2.2 Installing kronos-cts on the target board

Execute the following command in the OpenSTLinux build environment:

```
PC $> scp tmp*/deploy/deb/*neon*/khronos-cts_opengl-cts-<CTS Release>.deb root@<IPBOARD>:
<SomewhereInTheBoard>
```

Information

But default dpkg will extract within /home/root/.

This debian package is rather huge. For instance opengl-cts-4.6.0 is around 115MB. If rootfs space is at sake this is better to unpack the debian somewhere else by appending instdir=<SomewhereElse> to the above command

3 How to execute khronos-cts

3.1 Running test suite

Execute the following commands on the target board:

```
Board $> cd /home/root
Board $> ./cts-runner --type=es2 1>results.txt
[     1] EGL: enable default configs for conformance test
...
```

Information

The test execution can take several hours.

Warning

The *free* system RAM amount required to complete this test suite in around 500MB. Under out of memory occurs.

3.2 Running a single/group of test(s)

```
Board $> cd /home/root
Board $> glcts.exe --deqp-case=dEQP-EGL.functional.*
```

4 Testing verdict example

```
Board $>
...
Test run totals:
  Passed:         13786/13957 (98.8%)
  Failed:         11/13957 (0.1%)
  Not supported: 131/13957 (0.9%)
  Warnings:       29/13957 (0.2%)
219/220 sessions passed, conformance test FAILED
```

In order to interpret result, please look at "Understanding the result"[4]

Even more there are several tools available to process those test logs[5]

5 References

- https://www.khronos.org/
- https://github.com/KhronosGroup/VK-GL-CTS
- https://github.com/KhronosGroup/VK-GL-CTS/blob/master/external/openglcts/README.md#introduction
- https://github.com/KhronosGroup/VK-GL-CTS/tree/master/external/openglcts#understanding-the-results
- https://github.com/KhronosGroup/VK-GL-CTS/tree/master/external/openglcts#test-logs

Open Graphics Library (See http://www.opengl.org/ for more details)

Compatibility Test Suite (Android specific) or Clear to send (in UART context)

Khronos Native Platform Graphics Interface (See http://www.khronos.org/egl/ for more details)

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Stable: 26.10.2020 - 12:47 / Revision: 23.10.2020 - 12:51

A quality version of this page, approved on *26 October 2020*, was based off this revision.

Contents

1 Introduction

glmark2 *is an OpenGL 2.0 and ES 2.0 benchmark, developed by Alexandros Frantzis and Jesse Barker. It is based on the original glmark benchmark by Ben Smith.* *extracted from* ***The glmark2 official web site*** [1]

2 Using glmark2

Building glmark2 generates 2 binaries:

- **glmark2-es2-wayland**, to launch glmark2 benchmark as a wayland client.
- **glmark2-es2-drm**, to launch glmark2 benchmark as a native DRM/GBM openGLES application.

2.1 glmark2-es2-wayland

- Start Weston (if not already started)

```
Board $> systemctl start weston@root.service
```

- Launch glmark2

```
Board $> glmark2-es2-wayland
```

2.2 glmark2-es2-drm

- Stop Weston

```
Board $> systemctl stop weston@root.service
```

- The DRM display mode may need to be set and the DRM master token released so that glmark2 can use the DRM interfaces. This depends on the glmark2 version. For instance:

```
Board $> modetest -s 27:720x1280 -d &
```

Note: The connector and the mode are given as an example.

- Launch glmark2

```
Board $> glmark2-es2-drm
```

3 Source code location

- Official source code: https://github.com/glmark2/glmark2

4 To go further

You can find the full documentation of glmark2 in the related Ubuntu man page http://manpages.ubuntu.com/manpages/cosmic/man1/glmark2.1.html.

5 References

- https://github.com/glmark2/glmark2

Open Graphics Library (See http://www.opengl.org/ for more details)

Direct Rendering Manager (kernel module that gives direct hardware access to DRI clients, find more information on official DRI web site http://dri.freedesktop.org/wiki/DRM)

Stable: 26.02.2021 - 14:28 / Revision: 11.01.2021 - 17:25

A quality version of this page, approved on *26 February 2021*, was based off this revision.

When a GPU animation is running on the display, the related GCNANO estimated GPU load can be monitored from the GCNANO driver level, by using the following command:

```
Board $> (while true; do
gpu1=$(cat /sys/kernel/debug/gc/idle); \
sleep 4; \
gpu2=$(cat /sys/kernel/debug/gc/idle); \
echo $gpu1 $gpu2 | tr -d '\n' | tr -d ',' | tr -d 'ns' | awk -F" " '{printf("gpu load %.
0f%%\n", ($10-$2)*100/($10+$12+$14+$16-$2-$4-$6-$8))}'; \
done) &
```

The GCNANO estimated GPU load is then periodically output in the user console as a percentage "%":

```
gpu load 75%
gpu load 75%
gpu load 75%
```

Notes:

- Stop monitoring the GPU load with the command "**kill -9 `ps -o ppid= -C sleep`**".
- Adjust the GPU load update period by modifying the "sleep" value (4 seconds in the example).
- Use the command "dmesg -n8" to mix both user and kernel console outputs.
- Debugfs configuration needs to be enabled.
- The detailed calculation is: GPU load = (On-previous_On) / (Total-previous_Total) with Total=On+Off+Idle+Suspend, all variables coming from the command:

```
Board $> cat /sys/kernel/debug/gc/idle
On:                2,071,009,284,477 ns
Off:              11,480,071,864,263 ns
Idle:                              0 ns
Suspend:           1,242,043,838,898 ns
...
```

Graphics Processing Units

Stable: 13.11.2020 - 16:53 / Revision: 13.11.2020 - 14:45

Stable: 13:11:2020 - 10:00 | Revision: 13:11:2020 - 14:48

A quality version of this page, approved on *13 November 2020*, was based off this revision.

Contents

1 Tests

1.1 Conformance tests

- How to launch Khronos OpenGLES conformance tests

2 Benchmarks

2.1 Linux Benchmarks

- How to launch glmark2 benchmark

Stable: 24.09.2019 - 09:35 / Revision: 24.09.2019 - 09:34

A quality version of this page, approved on *24 September 2019*, was based off this revision.

OpenVG™ *is a royalty-free, cross-platform API that provides a low-level hardware acceleration interface for vector graphics libraries such as Flash and SVG. OpenVG is targeted primarily at handheld devices that require portable acceleration of high-quality vector graphics for compelling user interfaces and text on small screen devices, while enabling hardware acceleration to provide fluidly interactive performance at very low power levels.* (extracted from the official web site)

- **OpenVG Khronos official** [1]
- **OpenVG in wikipedia** [2]

References

- http://www.khronos.org/openvg/
- https://en.wikipedia.org/wiki/OpenVG

Open Vector Graphics (See http://www.khronos.org/openvg/ for more details)

Application programming interface