drm/tegra NVIDIA Tegra GPU and display driver

NVIDIA Tegra SoCs support a set of display, graphics and video functions via the host1x controller. host1x supplies command streams, gathered from a push buffer provided directly by the CPU, to its clients via channels. Software, or blocks amongst themselves, can use syncpoints for synchronization.

Up until, but not including, Tegra124 (aka Tegra K1) the drm/tegra driver supports the built-in GPU, comprised of the gr2d and gr3d engines. Starting with Tegra124 the GPU is based on the NVIDIA desktop GPU architecture and supported by the drm/nouveau driver.

The drm/tegra driver supports NVIDIA Tegra SoC generations since Tegra20. It has three parts:

- A host1x driver that provides infrastructure and access to the host1x services.
- A KMS driver that supports the display controllers as well as a number of outputs, such as RGB, HDMI, DSI, and DisplayPort.
- A set of custom userspace IOCTLs that can be used to submit jobs to the GPU and video engines via host1x.

Driver Infrastructure

The various host1x clients need to be bound together into a logical device in order to expose their functionality to users. The infrastructure that supports this is implemented in the host1x driver. When a driver is registered with the infrastructure it provides a list of compatible strings specifying the devices that it needs. The infrastructure creates a logical device and scan the device tree for matching device nodes, adding the required clients to a list. Drivers for individual clients register with the infrastructure as well and are added to the logical host1x device.

Once all clients are available, the infrastructure will initialize the logical device using a driver-provided function which will set up the bits specific to the subsystem and in turn initialize each of its clients.

Similarly, when one of the clients is unregistered, the infrastructure will destroy the logical device by calling back into the driver, which ensures that the subsystem specific bits are torn down and the clients destroyed in turn.

Host1x Infrastructure Reference

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\[linux-master][Documentation][gpu]tegra.rst, line 50)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/host1x.h
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\[linux-master][Documentation][gpu]tegra.rst, line 52)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/host1x/bus.c
   :export:
```

Host1x Syncpoint Reference

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\[linux-master][Documentation][gpu]tegra.rst, line 58)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/host1x/syncpt.c
   :export:
```

KMS driver

The display hardware has remained mostly backwards compatible over the various Tegra SoC generations, up until Tegra186 which introduces several changes that make it difficult to support with a parameterized driver.

Display Controllers

Tegra SoCs have two display controllers, each of which can be associated with zero or more outputs. Outputs can also share a single display controller, but only if they run with compatible display timings. Two display controllers can also share a single framebuffer, allowing cloned configurations even if modes on two outputs don't match. A display controller is modelled as a CRTC in KMS terms.

On Tegra186, the number of display controllers has been increased to three. A display controller can no longer drive all of the outputs. While two of these controllers can drive both DSI outputs and both SOR outputs, the third cannot drive any DSI.

Windows

A display controller controls a set of windows that can be used to composite multiple buffers onto the screen. While it is possible to assign arbitrary Z ordering to individual windows (by programming the corresponding blending registers), this is currently not supported by the driver. Instead, it will assume a fixed Z ordering of the windows (window A is the root window, that is, the lowest, while windows B and C are overlaid on top of window A). The overlay windows support multiple pixel formats and can automatically convert from YUV to RGB at scanout time. This makes them useful for displaying video content. In KMS, each window is modelled as a plane. Each display controller has a hardware cursor that is exposed as a cursor plane.

Outputs

The type and number of supported outputs varies between Tegra SoC generations. All generations support at least HDMI. While earlier generations supported the very simple RGB interfaces (one per display controller), recent generations no longer do and instead provide standard interfaces such as DSI and eDP/DP.

Outputs are modelled as a composite encoder/connector pair.

RGB/LVDS

This interface is no longer available since Tegra124. It has been replaced by the more standard DSI and eDP interfaces.

HDMI

HDMI is supported on all Tegra SoCs. Starting with Tegra210, HDMI is provided by the versatile SOR output, which supports eDP, DP and HDMI. The SOR is able to support HDMI 2.0, though support for this is currently not merged.

DSI

Although Tegra has supported DSI since Tegra30, the controller has changed in several ways in Tegra114. Since none of the publicly available development boards prior to Dalmore (Tegra114) have made use of DSI, only Tegra114 and later are supported by the drm/tegra driver.

eDP/DP

eDP was first introduced in Tegra124 where it was used to drive the display panel for notebook form factors. Tegra210 added support for full DisplayPort support, though this is currently not implemented in the drm/tegra driver.

Userspace Interface

The userspace interface provided by drm/tegra allows applications to create GEM buffers, access and control syncpoints as well as submit command streams to host1x.

GEM Buffers

The `DRM_IOCTL_TEGRA_GEM_CREATE` IOCTL is used to create a GEM buffer object with Tegra-specific flags. This is useful for buffers that should be tiled, or that are to be scanned out upside down (useful for 3D content).

After a GEM buffer object has been created, its memory can be mapped by an application using the mmap offset returned by the `DRM_IOCTL_TEGRA_GEM_MMAP` IOCTL.

Syncpoints

The current value of a syncpoint can be obtained by executing the `DRM_IOCTL_TEGRA_SYNCPT_READ` IOCTL. Incrementing the syncpoint is achieved using the `DRM_IOCTL_TEGRA_SYNCPT_INCR` IOCTL.

Userspace can also request blocking on a syncpoint. To do so, it needs to execute the `DRM_IOCTL_TEGRA_SYNCPT_WAIT` IOCTL, specifying the value of the syncpoint to wait for. The kernel will release the application when the syncpoint reaches that value or after a specified timeout.

Command Stream Submission

Before an application can submit command streams to host1x it needs to open a channel to an engine using the `DRM_IOCTL_TEGRA_OPEN_CHANNEL` IOCTL. Client IDs are used to identify the target of the channel. When a channel is no longer needed, it can be closed using the `DRM_IOCTL_TEGRA_CLOSE_CHANNEL` IOCTL. To retrieve the syncpoint associated with a channel, an application can use the `DRM_IOCTL_TEGRA_GET_SYNCPT`.

After opening a channel, submitting command streams is easy. The application writes commands into the memory backing a GEM buffer object and passes these to the `DRM_IOCTL_TEGRA_SUBMIT` IOCTL along with various other parameters, such as the syncpoints or relocations used in the job submission.