drm/vkms Virtual Kernel Modesetting

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\[linux-master][Documentation][gpu]vkms.rst`, line 7)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/vkms/vkms_drv.c
   :doc: vkms (Virtual Kernel Modesetting)
```

Setup

The VKMS driver can be setup with the following steps:

To check if VKMS is loaded, run:

```
lsmod | grep vkms
```

This should list the VKMS driver. If no output is obtained, then you need to enable and/or load the VKMS driver. Ensure that the VKMS driver has been set as a loadable module in your kernel config file. Do:

```
make nconfig

Go to `Device Drivers> Graphics support`

Enable `Virtual KMS (EXPERIMENTAL)`
```

Compile and build the kernel for the changes to get reflected. Now, to load the driver, use:

```
sudo modprobe vkms
```

On running the lsmod command now, the VKMS driver will appear listed. You can also observe the driver being loaded in the dmesg logs.

The VKMS driver has optional features to simulate different kinds of hardware, which are exposed as module options. You can use the *modinfo* command to see the module options for vkms:

```
modinfo vkms
```

Module options are helpful when testing, and enabling modules can be done while loading vkms. For example, to load vkms with cursor enabled, use:

```
sudo modprobe vkms enable_cursor=1
```

To disable the driver, use

```
sudo modprobe -r vkms
```

Testing With IGT

The IGT GPU Tools is a test suite used specifically for debugging and development of the DRM drivers. The IGT Tools can be installed from here .

The tests need to be run without a compositor, so you need to switch to text only mode. You can do this by:

```
sudo systemctl isolate multi-user.target
```

To return to graphical mode, do:

```
sudo systemctl isolate graphical.target
```

Once you are in text only mode, you can run tests using the --device switch or IGT_DEVICE variable to specify the device filter for the driver we want to test. IGT_DEVICE can also be used with the run-test.sh script to run the tests for a specific driver:

```
sudo ./build/tests/<name of test> --device "sys:/sys/devices/platform/vkms"
sudo IGT_DEVICE="sys:/sys/devices/platform/vkms" ./build/tests/<name of test>
sudo IGT_DEVICE="sys:/sys/devices/platform/vkms" ./scripts/run-tests.sh -t <name of test>
```

For example, to test the functionality of the writeback library, we can run the kms_writeback test:

```
sudo ./build/tests/kms_writeback --device "sys:/sys/devices/platform/vkms"
sudo IGT_DEVICE="sys:/sys/devices/platform/vkms" ./build/tests/kms_writeback
sudo IGT_DEVICE="sys:/sys/devices/platform/vkms" ./scripts/run-tests.sh -t kms_writeback
```

You can also run subtests if you do not want to run the entire test:

```
sudo ./build/tests/kms_flip --run-subtest basic-plain-flip --device "sys:/sys/devices/platform/vkms"
sudo IGT_DEVICE="sys:/sys/devices/platform/vkms" ./build/tests/kms_flip --run-subtest basic-plain-flip
```

TODO

If you want to do any of the items listed below, please share your interest with VKMS maintainers.

IGT better support

Debugging:

- kms_plane: some test cases are failing due to timeout on capturing CRC;
- kms_flip: when running test cases in sequence, some successful individual test cases are failing randomly; when individually, some successful test cases display in the log the following error:

  ```
  [drm:vkms_prepare_fb [vkms]] ERROR vmap failed: -4
  ```

Virtual hardware (vblank-less) mode:

- VKMS already has support for vblanks simulated via hrtimers, which can be tested with kms_flip test; in some way, we can say that VKMS already mimics the real hardware vblank. However, we also have virtual hardware that does not support vblank interrupt and completes page_flip events right away; in this case, compositor developers may end up creating a busy loop on virtual hardware. It would be useful to support Virtual Hardware behavior in VKMS because this can help compositor developers to test their features in multiple scenarios.

Add Plane Features

There's lots of plane features we could add support for:

- Clearing primary plane: clear primary plane before plane composition (at the start) for correctness of pixel blend ops. It also guarantees alpha channel is cleared in the target buffer for stable crc. [Good to get started]
- ARGB format on primary plane: blend the primary plane into background with translucent alpha.
- Support when the primary plane isn't exactly matching the output size: blend the primary plane into the black background.
- Full alpha blending on all planes.
- Rotation, scaling.
- Additional buffer formats, especially YUV formats for video like NV12. Low/high bpp RGB formats would also be interesting.
- Async updates (currently only possible on cursor plane using the legacy cursor api).

For all of these, we also want to review the igt test coverage and make sure all relevant igt testcases work on vkms. They are good options for internship project.

Runtime Configuration

We want to be able to reconfigure vkms instance without having to reload the module. Use/Test-cases:

- Hotplug/hotremove connectors on the fly (to be able to test DP MST handling of compositors).
- Configure planes/crtcs/connectors (we'd need some code to have more than 1 of them first).
- Change output configuration: Plug/unplug screens, change EDID, allow changing the refresh rate.

The currently proposed solution is to expose vkms configuration through configfs. All existing module options should be supported through configfs too.

Writeback support

- The writeback and CRC capture operations share the use of composer_enabled boolean to ensure vblanks. Probably, when these operations work together, composer_enabled needs to refcounting the composer state to proper work. [Good to get started]
- Add support for cloned writeback outputs and related test cases using a cloned output in the IGT kms_writeback.
- As a v4l device. This is useful for debugging compositors on special vkms configurations, so that developers see what's really going on.

Output Features

- Variable refresh rate/freesync support. This probably needs prime buffer sharing support, so that we can use vgem fences to simulate rendering in testing. Also needs support to specify the EDID.
- Add support for link status, so that compositors can validate their runtime fallbacks when e.g. a Display Port link goes bad.

CRC API Improvements

- Optimize CRC computation `compute_crc()` and plane blending `blend()`

Atomic Check using eBPF

Atomic drivers have lots of restrictions which are not exposed to userspace in any explicit form through e.g. possible property values. Userspace can only inquiry about these limits through the atomic IOCTL, possibly using the TEST_ONLY flag. Trying to add configurable code for all these limits, to allow compositors to be tested against them, would be rather futile exercise. Instead we could add support for eBPF to validate any kind of atomic state, and implement a library of different restrictions.

This needs a bunch of features (plane compositing, multiple outputs, ...) enabled already to make sense.