The Virtual Video Test Driver (vivid)

This driver emulates video4linux hardware of various types: video capture, video output, vbi capture and output, metadata capture and output, radio receivers and transmitters, touch capture and a software defined radio receiver. In addition a simple framebuffer device is available for testing capture and output overlays.

Up to 64 vivid instances can be created, each with up to 16 inputs and 16 outputs.

Each input can be a webcam, TV capture device, S-Video capture device or an HDMI capture device. Each output can be an S-Video output device or an HDMI output device.

These inputs and outputs act exactly as a real hardware device would behave. This allows you to use this driver as a test input for application development, since you can test the various features without requiring special hardware.

This document describes the features implemented by this driver:

- Support for read()/write(), MMAP, USERPTR and DMABUF streaming I/O.
- A large list of test patterns and variations thereof
- Working brightness, contrast, saturation and hue controls
- Support for the alpha color component
- Full colorspace support, including limited/full RGB range
- All possible control types are present
- Support for various pixel aspect ratios and video aspect ratios
- Error injection to test what happens if errors occur
- Supports crop/compose/scale in any combination for both input and output
- Can emulate up to 4K resolutions
- All Field settings are supported for testing interlaced capturing
- Supports all standard YUV and RGB formats, including two multiplanar YUV formats
- Raw and Sliced VBI capture and output support
- Radio receiver and transmitter support, including RDS support
- Software defined radio (SDR) support
- Capture and output overlay support
- Metadata capture and output support
- Touch capture support

These features will be described in more detail below.

Configuring the driver

By default the driver will create a single instance that has a video capture device with webcam, TV, S-Video and HDMI inputs, a video output device with S-Video and HDMI outputs, one vbi capture device, one vbi output device, one radio receiver device, one radio transmitter device and one SDR device.

The number of instances, devices, video inputs and outputs and their types are all configurable using the following module options:

- n_devs:

  number of driver instances to create. By default set to 1. Up to 64 instances can be created.

- node_types:

  which devices should each driver instance create. An array of hexadecimal values, one for each instance. The default is 0x1d3d. Each value is a bitmask with the following meaning:

  - bit 0: Video Capture node
  - bit 2-3: VBI Capture node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
  - bit 4: Radio Receiver node
  - bit 5: Software Defined Radio Receiver node
  - bit 8: Video Output node
  - bit 10-11: VBI Output node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
  - bit 12: Radio Transmitter node
  - bit 16: Framebuffer for testing overlays
  - bit 17: Metadata Capture node
  - bit 18: Metadata Output node
  - bit 19: Touch Capture node

  So to create four instances, the first two with just one video capture device, the second two with just one video output device you would pass these module options to vivid:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 82)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        n_devs=4 node_types=0x1,0x1,0x100,0x100
```

- num_inputs:

  the number of inputs, one for each instance. By default 4 inputs are created for each video capture device. At most 16 inputs can be created, and there must be at least one.

- input_types:

  the input types for each instance, the default is 0xe4. This defines what the type of each input is when the inputs are created for each driver instance. This is a hexadecimal value with up to 16 pairs of bits, each pair gives the type and bits 0-1 map to input 0, bits 2-3 map to input 1, 30-31 map to input 15. Each pair of bits has the following meaning:

  - 00: this is a webcam input
  - 01: this is a TV tuner input
  - 10: this is an S-Video input
  - 11: this is an HDMI input

  So to create a video capture device with 8 inputs where input 0 is a TV tuner, inputs 1-3 are S-Video inputs and inputs 4-7 are HDMI inputs you would use the following module options:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 109)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        num_inputs=8 input_types=0xffa9
```

- num_outputs:

  the number of outputs, one for each instance. By default 2 outputs are created for each video output device. At most 16 outputs can be created, and there must be at least one.

- output_types:

  the output types for each instance, the default is 0x02. This defines what the type of each output is when the outputs are created for each driver instance. This is a hexadecimal value with up to 16 bits, each bit gives the type and bit 0 maps to output 0, bit 1 maps to output 1, bit 15 maps to output 15. The meaning of each bit is as follows:

  - 0: this is an S-Video output
  - 1: this is an HDMI output

  So to create a video output device with 8 outputs where outputs 0-3 are S-Video outputs and outputs 4-7 are HDMI outputs you would use the following module options:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 134)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        num_outputs=8 output_types=0xf0
```

- vid_cap_nr:

  give the desired videoX start number for each video capture device. The default is -1 which will just take the first free number. This allows you to map capture video nodes to specific videoX device nodes. Example:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 144)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        n_devs=4 vid_cap_nr=2,4,6,8
```

This will attempt to assign /dev/video2 for the video capture device of the first vivid instance, video4 for the next up to video8 for the last instance. If it can't succeed, then it will just take the next free number.

- vid_out_nr:

  give the desired videoX start number for each video output device. The default is -1 which will just take the first free number.

- vbi_cap_nr:

  give the desired vbiX start number for each vbi capture device. The default is -1 which will just take the first free number.

- vbi_out_nr:

  give the desired vbiX start number for each vbi output device. The default is -1 which will just take the first free number.

- radio_rx_nr:

  give the desired radioX start number for each radio receiver device. The default is -1 which will just take the first free number.

- radio_tx_nr:

  give the desired radioX start number for each radio transmitter device. The default is -1 which will just take the first free number.

- sdr_cap_nr:

  give the desired swradioX start number for each SDR capture device. The default is -1 which will just take the first free number.

- meta_cap_nr:

  give the desired videoX start number for each metadata capture device. The default is -1 which will just take the first free number.

- meta_out_nr:

  give the desired videoX start number for each metadata output device. The default is -1 which will just take the first free number.

- touch_cap_nr:

  give the desired v4l-touchX start number for each touch capture device. The default is -1 which will just take the first free number.

- ccs_cap_mode:

  specify the allowed video capture crop/compose/scaling combination for each driver instance. Video capture devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those is should emulate. By default the user can select this through controls.

  The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

  - bit 0:

Enable crop support. Cropping will take only part of the incoming picture.

- bit 1:

  Enable compose support. Composing will copy the incoming picture into a larger buffer.

- bit 2:

  Enable scaling support. Scaling can scale the incoming picture. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

Note that this value is ignored by webcam inputs: those enumerate discrete framesizes and that is incompatible with cropping, composing or scaling.

- ccs_out_mode:

  specify the allowed video output crop/compose/scaling combination for each driver instance. Video output devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those is should emulate. By default the user can select this through controls.

  The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

  - bit 0:

    Enable crop support. Cropping will take only part of the outgoing buffer.

  - bit 1:

    Enable compose support. Composing will copy the incoming buffer into a larger picture frame.

  - bit 2:

    Enable scaling support. Scaling can scale the incoming buffer. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

- multiplanar:

  select whether each device instance supports multi-planar formats, and thus the V4L2 multi-planar API. By default device instances are single-planar.

  This module option can override that for each instance. Values are:

  - 1: this is a single-planar instance.
  - 2: this is a multi-planar instance.

- vivid_debug:

  enable driver debugging info

- no_error_inj:

  if set disable the error injecting controls. This option is needed in order to run a tool like v4l2-compliance. Tools like that exercise all controls including a control like 'Disconnect' which emulates a USB disconnect, making the device inaccessible and so all tests that v4l2-compliance is doing will fail afterwards.

  There may be other situations as well where you want to disable the error injection support of vivid. When this option is set, then the controls that select crop, compose and scale behavior are also removed. Unless overridden by ccs_cap_mode and/or ccs_out_mode the will default to enabling crop, compose and scaling.

- allocators:

  memory allocator selection, default is 0. It specifies the way buffers will be allocated.

  - 0: vmalloc
  - 1: dma-contig

- cache_hints:

  specifies if the device should set queues' user-space cache and memory consistency hint capability

(V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS). The hints are valid only when using MMAP streaming I/O. Default is 0.

- 0: forbid hints
- 1: allow hints

Taken together, all these module options allow you to precisely customize the driver behavior and test your application with all sorts of permutations. It is also very suitable to emulate hardware that is not yet available, e.g. when developing software for a new upcoming device.

Video Capture

This is probably the most frequently used feature. The video capture device can be configured by using the module options num_inputs, input_types and ccs_cap_mode (see section 1 for more detailed information), but by default four inputs are configured: a webcam, a TV tuner, an S-Video and an HDMI input, one input for each input type. Those are described in more detail below.

Special attention has been given to the rate at which new frames become available. The jitter will be around 1 jiffie (that depends on the HZ configuration of your kernel, so usually 1/100, 1/250 or 1/1000 of a second), but the long-term behavior is exactly following the framerate. So a framerate of 59.94 Hz is really different from 60 Hz. If the framerate exceeds your kernel's HZ value, then you will get dropped frames, but the frame/field sequence counting will keep track of that so the sequence count will skip whenever frames are dropped.

Webcam Input

The webcam input supports three framesizes: 320x180, 640x360 and 1280x720. It supports frames per second settings of 10, 15, 25, 30, 50 and 60 fps. Which ones are available depends on the chosen framesize: the larger the framesize, the lower the maximum frames per second.

The initially selected colorspace when you switch to the webcam input will be sRGB.

TV and S-Video Inputs

The only difference between the TV and S-Video input is that the TV has a tuner. Otherwise they behave identically.

These inputs support audio inputs as well: one TV and one Line-In. They both support all TV standards. If the standard is queried, then the Vivid controls 'Standard Signal Mode' and 'Standard' determine what the result will be.

These inputs support all combinations of the field setting. Special care has been taken to faithfully reproduce how fields are handled for the different TV standards. This is particularly noticeable when generating a horizontally moving image so the temporal effect of using interlaced formats becomes clearly visible. For 50 Hz standards the top field is the oldest and the bottom field is the newest in time. For 60 Hz standards that is reversed: the bottom field is the oldest and the top field is the newest in time.

When you start capturing in V4L2_FIELD_ALTERNATE mode the first buffer will contain the top field for 50 Hz standards and the bottom field for 60 Hz standards. This is what capture hardware does as well.

Finally, for PAL/SECAM standards the first half of the top line contains noise. This simulates the Wide Screen Signal that is commonly placed there.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

The pixel aspect ratio will depend on the TV standard. The video aspect ratio can be selected through the 'Standard Aspect Ratio' Vivid control. Choices are '4x3', '16x9' which will give letterboxed widescreen video and '16x9 Anamorphic' which will give full screen squashed anamorphic widescreen video that will need to be scaled accordingly.

The TV 'tuner' supports a frequency range of 44-958 MHz. Channels are available every 6 MHz, starting from 49.25 MHz. For each channel the generated image will be in color for the +/- 0.25 MHz around it, and in grayscale for +/- 1 MHz around the channel. Beyond that it is just noise. The VIDIOC_G_TUNER ioctl will return 100% signal strength for +/- 0.25 MHz and 50% for +/- 1 MHz. It will also return correct afc values to show whether the frequency is too low or too high.

The audio subchannels that are returned are MONO for the +/- 1 MHz range around a valid channel frequency. When the frequency is within +/- 0.25 MHz of the channel it will return either MONO, STEREO, either MONO | SAP (for NTSC) or LANG1 | LANG2 (for others), or STEREO | SAP.

Which one is returned depends on the chosen channel, each next valid channel will cycle through the possible audio subchannel combinations. This allows you to test the various combinations by just switching channels..

Finally, for these inputs the v4l2_timecode struct is filled in in the dequeued v4l2_buffer struct.

HDMI Input

The HDMI inputs supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always V4L2_FIELD_ALTERNATE. For HDMI the field order is always top field first, and when you start capturing an interlaced format you will receive the top field first.

The initially selected colorspace when you switch to the HDMI input or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

The video aspect ratio can be selected through the 'DV Timings Aspect Ratio' Vivid control. Choices are 'Source Width x Height' (just use the same ratio as the chosen format), '4x3' or '16x9', either of which can result in pillarboxed or letterboxed video.

For HDMI inputs it is possible to set the EDID. By default a simple EDID is provided. You can only set the EDID for HDMI inputs. Internally, however, the EDID is shared between all HDMI inputs.

No interpretation is done of the EDID data with the exception of the physical address. See the CEC section for more details.

There is a maximum of 15 HDMI inputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address.

Video Output

The video output device can be configured by using the module options num_outputs, output_types and ccs_out_mode (see section 1 for more detailed information), but by default two outputs are configured: an S-Video and an HDMI input, one output for each output type. Those are described in more detail below.

Like with video capture the framerate is also exact in the long term.

S-Video Output

This output supports audio outputs as well: "Line-Out 1" and "Line-Out 2". The S-Video output supports all TV standards.

This output supports all combinations of the field setting.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

HDMI Output

The HDMI output supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always V4L2_FIELD_ALTERNATE.

The initially selected colorspace when you switch to the HDMI output or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

An HDMI output has a valid EDID which can be obtained through VIDIOC_G_EDID.

There is a maximum of 15 HDMI outputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address. See also the CEC section for more details.

VBI Capture

There are three types of VBI capture devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the node_types module option. In all cases the driver will generate valid VBI data: for 60 Hz standards it will generate Closed Caption and XDS data. The closed caption stream will alternate between "Hello world!" and "Closed captions test" every second. The XDS stream will give the current time once a minute. For 50 Hz standards it will generate the Wide Screen Signal which is based on the actual Video Aspect Ratio control setting and teletext pages 100-159, one page per frame.

The VBI device will only work for the S-Video and TV inputs, it will give back an error if the current input is a webcam or HDMI.

VBI Output

There are three types of VBI output devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the node_types module option.

The sliced VBI output supports the Wide Screen Signal and the teletext signal for 50 Hz standards and Closed Captioning + XDS for 60 Hz standards.

The VBI device will only work for the S-Video output, it will give back an error if the current output is HDMI.

Radio Receiver

The radio receiver emulates an FM/AM/SW receiver. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

Valid channels are emulated every 1 MHz for FM and every 100 kHz for AM and SW. The signal strength decreases the further the frequency is from the valid frequency until it becomes 0% at +/- 50 kHz (FM) or 5 kHz (AM/SW) from the ideal frequency. The initial frequency when the driver is loaded is set to 95 MHz.

The FM receiver supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the RDS information is stored in read-only controls. These controls are updated every time the frequency is changed, or when the tuner status is requested. The Block I/O method uses the read() interface to pass the RDS blocks on to the application for decoding.

The RDS signal is 'detected' for +/- 12.5 kHz around the channel frequency, and the further the frequency is away from the valid frequency the more RDS errors are randomly introduced into the block I/O stream, up to 50% of all blocks if you are +/- 12.5 kHz from the channel frequency. All four errors can occur in equal proportions: blocks marked 'CORRECTED', blocks marked 'ERROR', blocks marked 'INVALID' and dropped blocks.

The generated RDS stream contains all the standard fields contained in a 0B group, and also radio text and the current time.

The receiver supports HW frequency seek, either in Bounded mode, Wrap Around mode or both, which is configurable with the "Radio HW Seek Mode" control.

Radio Transmitter

The radio transmitter emulates an FM/AM/SW transmitter. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

The initial frequency when the driver is loaded is 95.5 MHz.

The FM transmitter supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the transmitted RDS information is configured using controls, and in 'Block I/O' mode the blocks are passed to the driver using write().

Software Defined Radio Receiver

The SDR receiver has three frequency bands for the ADC tuner:

- 300 kHz
- 900 kHz - 2800 kHz
- 3200 kHz

The RF tuner supports 50 MHz - 2000 MHz.

The generated data contains the In-phase and Quadrature components of a 1 kHz tone that has an amplitude of sqrt(2).

Metadata Capture

The Metadata capture generates UVC format metadata. The PTS and SCR are transmitted based on the values set in vivid contols.

The Metadata device will only work for the Webcam input, it will give back an error for all other inputs.

Metadata Output

The Metadata output can be used to set brightness, contrast, saturation and hue.

The Metadata device will only work for the Webcam output, it will give back an error for all other outputs.

Touch Capture

The Touch capture generates touch patterns simulating single tap, double tap, triple tap, move from left to right, zoom in, zoom out, palm press (simulating a large area being pressed on a touchpad), and simulating 16 simultaneous touch points.

Controls

Different devices support different controls. The sections below will describe each control and which devices support them.

User Controls - Test Controls

The Button, Boolean, Integer 32 Bits, Integer 64 Bits, Menu, String, Bitmask and Integer Menu are controls that represent all possible control types. The Menu control and the Integer Menu control both have 'holes' in their menu list, meaning that one or more menu items return EINVAL when VIDIOC_QUERYMENU is called. Both menu controls also have a non-zero minimum control value. These features allow you to check if your application can handle such things correctly. These controls are supported for every device type.

User Controls - Video Capture

The following controls are specific to video capture.

The Brightness, Contrast, Saturation and Hue controls actually work and are standard. There is one special feature with the Brightness control: each video input has its own brightness value, so changing input will restore the brightness for that input. In addition, each video input uses a different brightness range (minimum and maximum control values). Switching inputs will cause a control event to be sent with the V4L2_EVENT_CTRL_CH_RANGE flag set. This allows you to test controls that can change their range.

The 'Gain, Automatic' and Gain controls can be used to test volatile controls: if 'Gain, Automatic' is set, then the Gain control is volatile and changes constantly. If 'Gain, Automatic' is cleared, then the Gain control is a normal control.

The 'Horizontal Flip' and 'Vertical Flip' controls can be used to flip the image. These combine with the 'Sensor Flipped Horizontally/Vertically' Vivid controls.

The 'Alpha Component' control can be used to set the alpha component for formats containing an alpha channel.

User Controls - Audio

The following controls are specific to video capture and output and radio receivers and transmitters.

The 'Volume' and 'Mute' audio controls are typical for such devices to control the volume and mute the audio. They don't actually do anything in the vivid driver.

Vivid Controls

These vivid custom controls control the image generation, error injection, etc.

Test Pattern Controls

The Test Pattern Controls are all specific to video capture.

- Test Pattern:

  selects which test pattern to use. Use the CSC Colorbar for testing colorspace conversions: the colors used in that test pattern map to valid colors in all colorspaces. The colorspace conversion is disabled for the other test patterns.

- OSD Text Mode:

  selects whether the text superimposed on the test pattern should be shown, and if so, whether only counters should be displayed or the full text.

- Horizontal Movement:

  selects whether the test pattern should move to the left or right and at what speed.

- Vertical Movement:

  does the same for the vertical direction.

- Show Border:

  show a two-pixel wide border at the edge of the actual image, excluding letter or pillarboxing.

- Show Square:

  show a square in the middle of the image. If the image is displayed with the correct pixel and image aspect ratio corrections, then the width and height of the square on the monitor should be the same.

- Insert SAV Code in Image:

  adds a SAV (Start of Active Video) code to the image. This can be used to check if such codes in the image are inadvertently interpreted instead of being ignored.

- Insert EAV Code in Image:

does the same for the EAV (End of Active Video) code.

Capture Feature Selection Controls

These controls are all specific to video capture.

- Sensor Flipped Horizontally:

  the image is flipped horizontally and the V4L2_IN_ST_HFLIP input status flag is set. This emulates the case where a sensor is for example mounted upside down.

- Sensor Flipped Vertically:

  the image is flipped vertically and the V4L2_IN_ST_VFLIP input status flag is set. This emulates the case where a sensor is for example mounted upside down.

- Standard Aspect Ratio:

  selects if the image aspect ratio as used for the TV or S-Video input should be 4x3, 16x9 or anamorphic widescreen. This may introduce letterboxing.

- DV Timings Aspect Ratio:

  selects if the image aspect ratio as used for the HDMI input should be the same as the source width and height ratio, or if it should be 4x3 or 16x9. This may introduce letter or pillarboxing.

- Timestamp Source:

  selects when the timestamp for each buffer is taken.

- Colorspace:

  selects which colorspace should be used when generating the image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

  Changing the colorspace will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Transfer Function:

  selects which colorspace transfer function should be used when generating an image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

  Changing the transfer function will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Y'CbCr Encoding:

  selects which Y'CbCr encoding should be used when generating a Y'CbCr image. This only applies if the format is set to a Y'CbCr format as opposed to an RGB format.

  Changing the Y'CbCr encoding will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Quantization:

  selects which quantization should be used for the RGB or Y'CbCr encoding when generating the test pattern.

  Changing the quantization will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Limited RGB Range (16-235):

  selects if the RGB range of the HDMI source should be limited or full range. This combines with the Digital Video 'Rx RGB Quantization Range' control and can be used to test what happens if a source provides you with the wrong quantization range information. See the description of that control for more details.

- Apply Alpha To Red Only:

apply the alpha channel as set by the 'Alpha Component' user control to the red color of the test pattern only.

- Enable Capture Cropping:

  enables crop support. This control is only present if the ccs_cap_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Capture Composing:

  enables composing support. This control is only present if the ccs_cap_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Capture Scaler:

  enables support for a scaler (maximum 4 times upscaling and downscaling). This control is only present if the ccs_cap_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Maximum EDID Blocks:

  determines how many EDID blocks the driver supports. Note that the vivid driver does not actually interpret new EDID data, it just stores it. It allows for up to 256 EDID blocks which is the maximum supported by the standard.

- Fill Percentage of Frame:

  can be used to draw only the top X percent of the image. Since each frame has to be drawn by the driver, this demands a lot of the CPU. For large resolutions this becomes problematic. By drawing only part of the image this CPU load can be reduced.

Output Feature Selection Controls

These controls are all specific to video output.

- Enable Output Cropping:

  enables crop support. This control is only present if the ccs_out_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Output Composing:

  enables composing support. This control is only present if the ccs_out_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Output Scaler:

  enables support for a scaler (maximum 4 times upscaling and downscaling). This control is only present if the ccs_out_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

Error Injection Controls

The following two controls are only valid for video and vbi capture.

- Standard Signal Mode:

  selects the behavior of VIDIOC_QUERYSTD: what should it return?

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).

- Standard:

  selects the standard that VIDIOC_QUERYSTD should return if the previous control is set to "Selected Standard".

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a changed input standard.

The following two controls are only valid for video capture.

- DV Timings Signal Mode:

  selects the behavior of VIDIOC_QUERY_DV_TIMINGS: what should it return?

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).

- DV Timings:

  selects the timings the VIDIOC_QUERY_DV_TIMINGS should return if the previous control is set to "Selected DV Timings".

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates changed input timings.

The following controls are only present if the no_error_inj module option is set to 0 (the default). These controls are valid for video and vbi capture and output streams and for the SDR capture device except for the Disconnect control which is valid for all devices.

- Wrap Sequence Number:

  test what happens when you wrap the sequence number in struct v4l2_buffer around.

- Wrap Timestamp:

  test what happens when you wrap the timestamp in struct v4l2_buffer around.

- Percentage of Dropped Buffers:

  sets the percentage of buffers that are never returned by the driver (i.e., they are dropped).

- Disconnect:

  emulates a USB disconnect. The device will act as if it has been disconnected. Only after all open filehandles to the device node have been closed will the device become 'connected' again.

- Inject V4L2_BUF_FLAG_ERROR:

  when pressed, the next frame returned by the driver will have the error flag set (i.e. the frame is marked corrupt).

- Inject VIDIOC_REQBUFS Error:

  when pressed, the next REQBUFS or CREATE_BUFS ioctl call will fail with an error. To be precise: the videobuf2 queue_setup() op will return -EINVAL.

- Inject VIDIOC_QBUF Error:

  when pressed, the next VIDIOC_QBUF or VIDIOC_PREPARE_BUFFER ioctl call will fail with an error. To be precise: the videobuf2 buf_prepare() op will return -EINVAL.

- Inject VIDIOC_STREAMON Error:

  when pressed, the next VIDIOC_STREAMON ioctl call will fail with an error. To be precise: the videobuf2 start_streaming() op will return -EINVAL.

- Inject Fatal Streaming Error:

  when pressed, the streaming core will be marked as having suffered a fatal error, the only way to recover from that is to stop streaming. To be precise: the videobuf2 vb2_queue_error() function is called.

VBI Raw Capture Controls

- Interlaced VBI Format:

  if set, then the raw VBI data will be interlaced instead of providing it grouped by field.

Digital Video Controls

- Rx RGB Quantization Range:

  sets the RGB quantization detection of the HDMI input. This combines with the Vivid 'Limited RGB Range (16-235)' control and can be used to test what happens if a source provides you with the wrong quantization range information. This can be tested by selecting an HDMI input, setting this control to Full or Limited range and selecting the opposite in the 'Limited RGB Range (16-235)' control. The effect is easy to see if the 'Gray Ramp' test pattern is selected.

- Tx RGB Quantization Range:

sets the RGB quantization detection of the HDMI output. It is currently not used for anything in vivid, but most HDMI transmitters would typically have this control.

- Transmit Mode:

  sets the transmit mode of the HDMI output to HDMI or DVI-D. This affects the reported colorspace since DVI_D outputs will always use sRGB.

- Display Present:

  sets the presence of a "display" on the HDMI output. This affects the tx_edid_present, tx_hotplug and tx_rxsense controls.

FM Radio Receiver Controls

- RDS Reception:

  set if the RDS receiver should be enabled.

- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

  these are all read-only controls. If RDS Rx I/O Mode is set to "Block I/O", then they are inactive as well. If RDS Rx I/O Mode is set to "Controls", then these controls report the received RDS data.

> **Note**
>
> The vivid implementation of this is pretty basic: they are only updated when you set a new frequency or when you get the tuner status (VIDIOC_G_TUNER).

- Radio HW Seek Mode:

  can be one of "Bounded", "Wrap Around" or "Both". This determines if VIDIOC_S_HW_FREQ_SEEK will be bounded by the frequency range or wrap-around or if it is selectable by the user.

- Radio Programmable HW Seek:

  if set, then the user can provide the lower and upper bound of the HW Seek. Otherwise the frequency range boundaries will be used.

- Generate RBDS Instead of RDS:

  if set, then generate RBDS (the US variant of RDS) data instead of RDS (European-style RDS). This affects only the PICODE and PTY codes.

- RDS Rx I/O Mode:

  this can be "Block I/O" where the RDS blocks have to be read() by the application, or "Controls" where the RDS data is provided by the RDS controls mentioned above.

FM Radio Modulator Controls

- RDS Program ID:
- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Stereo:
- RDS Artificial Head:
- RDS Compressed:
- RDS Dynamic PTY:

- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

  these are all controls that set the RDS data that is transmitted by the FM modulator.

- RDS Tx I/O Mode:

  this can be "Block I/O" where the application has to use write() to pass the RDS blocks to the driver, or "Controls" where the RDS data is Provided by the RDS controls mentioned above.

Metadata Capture Controls

- Generate PTS

  if set, then the generated metadata stream contains Presentation timestamp.

- Generate SCR

  if set, then the generated metadata stream contains Source Clock information.

Video, VBI and RDS Looping

The vivid driver supports looping of video output to video input, VBI output to VBI input and RDS output to RDS input. For video/VBI looping this emulates as if a cable was hooked up between the output and input connector. So video and VBI looping is only supported between S-Video and HDMI inputs and outputs. VBI is only valid for S-Video as it makes no sense for HDMI.

Since radio is wireless this looping always happens if the radio receiver frequency is close to the radio transmitter frequency. In that case the radio transmitter will 'override' the emulated radio stations.

Looping is currently supported only between devices created by the same vivid driver instance.

Video and Sliced VBI looping

The way to enable video/VBI looping is currently fairly crude. A 'Loop Video' control is available in the "Vivid" control class of the video capture and VBI capture devices. When checked the video looping will be enabled. Once enabled any video S-Video or HDMI input will show a static test pattern until the video output has started. At that time the video output will be looped to the video input provided that:

- the input type matches the output type. So the HDMI input cannot receive video from the S-Video output.
- the video resolution of the video input must match that of the video output. So it is not possible to loop a 50 Hz (720x576) S-Video output to a 60 Hz (720x480) S-Video input, or a 720p60 HDMI output to a 1080p30 input.
- the pixel formats must be identical on both sides. Otherwise the driver would have to do pixel format conversion as well, and that's taking things too far.
- the field settings must be identical on both sides. Same reason as above: requiring the driver to convert from one field format to another complicated matters too much. This also prohibits capturing with 'Field Top' or 'Field Bottom' when the output video is set to 'Field Alternate'. This combination, while legal, became too complicated to support. Both sides have to be 'Field Alternate' for this to work. Also note that for this specific case the sequence and field counting in struct v4l2_buffer on the capture side may not be 100% accurate.
- field settings V4L2_FIELD_SEQ_TB/BT are not supported. While it is possible to implement this, it would mean a lot of work to get this right. Since these field values are rarely used the decision was made not to implement this for now.
- on the input side the "Standard Signal Mode" for the S-Video input or the "DV Timings Signal Mode" for the HDMI input should be configured so that a valid signal is passed to the video input.

The framerates do not have to match, although this might change in the future.

By default you will see the OSD text superimposed on top of the looped video. This can be turned off by changing the "OSD Text Mode" control of the video capture device.

For VBI looping to work all of the above must be valid and in addition the vbi output must be configured for sliced VBI. The VBI capture side can be configured for either raw or sliced VBI. Note that at the moment only CC/XDS (60 Hz formats) and WSS (50 Hz formats) VBI data is looped. Teletext VBI data is not looped.

Radio & RDS Looping

As mentioned in section 6 the radio receiver emulates stations are regular frequency intervals. Depending on the frequency of the radio receiver a signal strength value is calculated (this is returned by VIDIOC_G_TUNER). However, it will also look at the frequency set by the radio transmitter and if that results in a higher signal strength than the settings of the radio transmitter will be used as if it was a valid station. This also includes the RDS data (if any) that the transmitter 'transmits'. This is received faithfully on the receiver side. Note that when the driver is loaded the frequencies of the radio receiver and transmitter are not identical, so initially no looping takes place.

Cropping, Composing, Scaling

This driver supports cropping, composing and scaling in any combination. Normally which features are supported can be selected through the Vivid controls, but it is also possible to hardcode it when the module is loaded through the ccs_cap_mode and ccs_out_mode module options. See section 1 on the details of these module options.

This allows you to test your application for all these variations.

Note that the webcam input never supports cropping, composing or scaling. That only applies to the TV/S-Video/HDMI inputs and outputs. The reason is that webcams, including this virtual implementation, normally use VIDIOC_ENUM_FRAMESIZES to list a set of discrete framesizes that it supports. And that does not combine with cropping, composing or scaling. This is primarily a limitation of the V4L2 API which is carefully reproduced here.

The minimum and maximum resolutions that the scaler can achieve are 16x16 and (4096 * 4) x (2160 x 4), but it can only scale up or down by a factor of 4 or less. So for a source resolution of 1280x720 the minimum the scaler can do is 320x180 and the maximum is 5120x2880. You can play around with this using the qv4l2 test tool and you will see these dependencies.

This driver also supports larger 'bytesperline' settings, something that VIDIOC_S_FMT allows but that few drivers implement.

The scaler is a simple scaler that uses the Coarse Bresenham algorithm. It's designed for speed and simplicity, not quality.

If the combination of crop, compose and scaling allows it, then it is possible to change crop and compose rectangles on the fly.

Formats

The driver supports all the regular packed and planar 4:4:4, 4:2:2 and 4:2:0 YUYV formats, 8, 16, 24 and 32 RGB packed formats and various multiplanar formats.

The alpha component can be set through the 'Alpha Component' User control for those formats that support it. If the 'Apply Alpha To Red Only' control is set, then the alpha component is only used for the color red and set to 0 otherwise.

The driver has to be configured to support the multiplanar formats. By default the driver instances are single-planar. This can be changed by setting the multiplanar module option, see section 1 for more details on that option.

If the driver instance is using the multiplanar formats/API, then the first single planar format (YUYV) and the multiplanar NV16M and NV61M formats the will have a plane that has a non-zero data_offset of 128 bytes. It is rare for data_offset to be non-zero, so this is a useful feature for testing applications.

Video output will also honor any data_offset that the application set.

Capture Overlay

Note: capture overlay support is implemented primarily to test the existing V4L2 capture overlay API. In practice few if any GPUs support such overlays anymore, and neither are they generally needed anymore since modern hardware is so much more capable. By setting flag 0x10000 in the node_types module option the vivid driver will create a simple framebuffer device that can be used for testing this API. Whether this API should be used for new drivers is questionable.

This driver has support for a destructive capture overlay with bitmap clipping and list clipping (up to 16 rectangles) capabilities. Overlays are not supported for multiplanar formats. It also honors the struct v4l2_window field setting: if it is set to FIELD_TOP or FIELD_BOTTOM and the capture setting is FIELD_ALTERNATE, then only the top or bottom fields will be copied to the overlay.

The overlay only works if you are also capturing at that same time. This is a vivid limitation since it copies from a buffer to the overlay instead of filling the overlay directly. And if you are not capturing, then no buffers are available to fill.

In addition, the pixelformat of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise VIDIOC_OVERLAY will return an error.

In order to really see what it going on you will need to create two vivid instances: the first with a framebuffer enabled. You configure the capture overlay of the second instance to use the framebuffer of the first, then you start capturing in the second instance. For the first instance you setup the output overlay for the video output, turn on video looping and capture to see the blended framebuffer overlay that's being written to by the second instance. This setup would require the following commands:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 1293)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        $ sudo modprobe vivid n_devs=2 node_types=0x10101,0x1
        $ v4l2-ctl -d1 --find-fb
        /dev/fb1 is the framebuffer associated with base address 0x12800000
        $ sudo v4l2-ctl -d2 --set-fbuf fb=1
        $ v4l2-ctl -d1 --set-fbuf fb=1
        $ v4l2-ctl -d0 --set-fmt-video=pixelformat='AR15'
        $ v4l2-ctl -d1 --set-fmt-video-out=pixelformat='AR15'
        $ v4l2-ctl -d2 --set-fmt-video=pixelformat='AR15'
        $ v4l2-ctl -d0 -i2
        $ v4l2-ctl -d2 -i2
        $ v4l2-ctl -d2 -c horizontal_movement=4
        $ v4l2-ctl -d1 --overlay=1
        $ v4l2-ctl -d1 -c loop_video=1
        $ v4l2-ctl -d2 --stream-mmap --overlay=1
```

And from another console:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 1312)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        $ v4l2-ctl -d1 --stream-out-mmap
```

And yet another console:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\media\[linux-master][Documentation][admin-guide][media]vivid.rst`, line 1318)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        $ qv4l2
```

and start streaming.

As you can see, this is not for the faint of heart...

Output Overlay

Note: output overlays are primarily implemented in order to test the existing V4L2 output overlay API. Whether this API should be used for new drivers is questionable.

This driver has support for an output overlay and is capable of:

- bitmap clipping,
- list clipping (up to 16 rectangles)
- chromakey
- source chromakey
- global alpha
- local alpha
- local inverse alpha

Output overlays are not supported for multiplanar formats. In addition, the pixelformat of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise VIDIOC_OVERLAY will return an error.

Output overlays only work if the driver has been configured to create a framebuffer by setting flag 0x10000 in the node_types module option. The created framebuffer has a size of 720x576 and supports ARGB 1:5:5:5 and RGB 5:6:5.

In order to see the effects of the various clipping, chromakeying or alpha processing capabilities you need to turn on video looping and see the results on the capture side. The use of the clipping, chromakeying or alpha processing capabilities will slow down the video loop considerably as a lot of checks have to be done per pixel.

CEC (Consumer Electronics Control)

If there are HDMI inputs then a CEC adapter will be created that has the same number of input ports. This is the equivalent of e.g. a TV that has that number of inputs. Each HDMI output will also create a CEC adapter that is hooked up to the corresponding input port, or (if there are more outputs than inputs) is not hooked up at all. In other words, this is the equivalent of hooking up each output device to an input port of the TV. Any remaining output devices remain unconnected.

The EDID that each output reads reports a unique CEC physical address that is based on the physical address of the EDID of the input. So if the EDID of the receiver has physical address A.B.0.0, then each output will see an EDID containing physical address A.B.C.0 where C is 1 to the number of inputs. If there are more outputs than inputs then the remaining outputs have a CEC adapter that is disabled and reports an invalid physical address.

Some Future Improvements

Just as a reminder and in no particular order:

- Add a virtual alsa driver to test audio
- Add virtual sub-devices and media controller support
- Some support for testing compressed video
- Add support to loop raw VBI output to raw VBI input
- Add support to loop teletext sliced VBI output to VBI input
- Fix sequence/field numbering when looping of video with alternate fields
- Add support for V4L2_CID_BG_COLOR for video outputs
- Add ARGB888 overlay support: better testing of the alpha channel
- Improve pixel aspect support in the tpg code by passing a real v4l2_fract
- Use per-queue locks and/or per-device locks to improve throughput
- Add support to loop from a specific output to a specific input across vivid instances
- The SDR radio should use the same 'frequencies' for stations as the normal radio receiver, and give back noise if the frequency doesn't match up with a station frequency
- Make a thread for the RDS generation, that would help in particular for the "Controls" RDS Rx I/O Mode as the read-only RDS controls could be updated in real-time.
- Changing the EDID should cause hotplug detect emulation to happen.