STM32MP15x Series interfacing with a MIPI® CSI-2 camera

Introduction

This application note provides information on how to interface the STM32MP15x Series with a MIPI CSI-2 camera. The STM32MP15x Series, like the STM32 high-performance MCUs, can address CMOS camera sensors through its DCMI (digital camera module interface) parallel port. However, it is possible to extend the range of addressable camera sensors for instance MIPI® CSI-2 cameras (camera serial interface), thanks to the STMIPID02 MIPI CSI-2 deserializer discrete component.

The MIPI CSI-2 interface protocol has become for many years the standard technology in today's embedded sensors, mostly driven by the mobile market and it is also widely used in the industrial market where MIPI CSI-2 brings decisive advantages, offering a lower pin count and cost versus the conventional parallel interface or MIPI CPI.

The STMIPID02 MIPI CSI-2 deserializer can address a broad range of MIPI CSI-2 camera sensors used in mobile devices and automotive applications. This direct interface removes the requirement for associated software overhead linked to frame decoding (for camera over USB or Ethernet for instance).

The purpose of this application note is to demonstrate, using the DH96 Avenger board, the STM32MP15x Series capability to address the five Mpixel OV5640 MIPI CSI-2 camera sensor through the STMIPID02 MIPI CSI-2 deserializer. Both drivers are available and included in the STMicroelectronics OpenSTLinux software distribution package. For the purpose of this application note and according to the STMIPID02 deserializer specifications, the focus is made exclusively on the MIPI CSI-2.1 protocol over a D-PHY.

1 General information

This document applies to STM32MP15x Series Arm® Cortex® core-based microprocessors.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

Table 1. **List of acronyms**

Acronym	Description
CSI	Camera serial interface
CPI	Camera parallel interface
MIPI	Mobile industry processor interface
DCMI	Digital camera interface
PMIC	Power management integrated circuit
LDO	Low-dropout regulator

2 Reference documents

Below resources are public and available either on STMicroelectronics web site at or on third parties websites.

Table 2. **Reference documents**

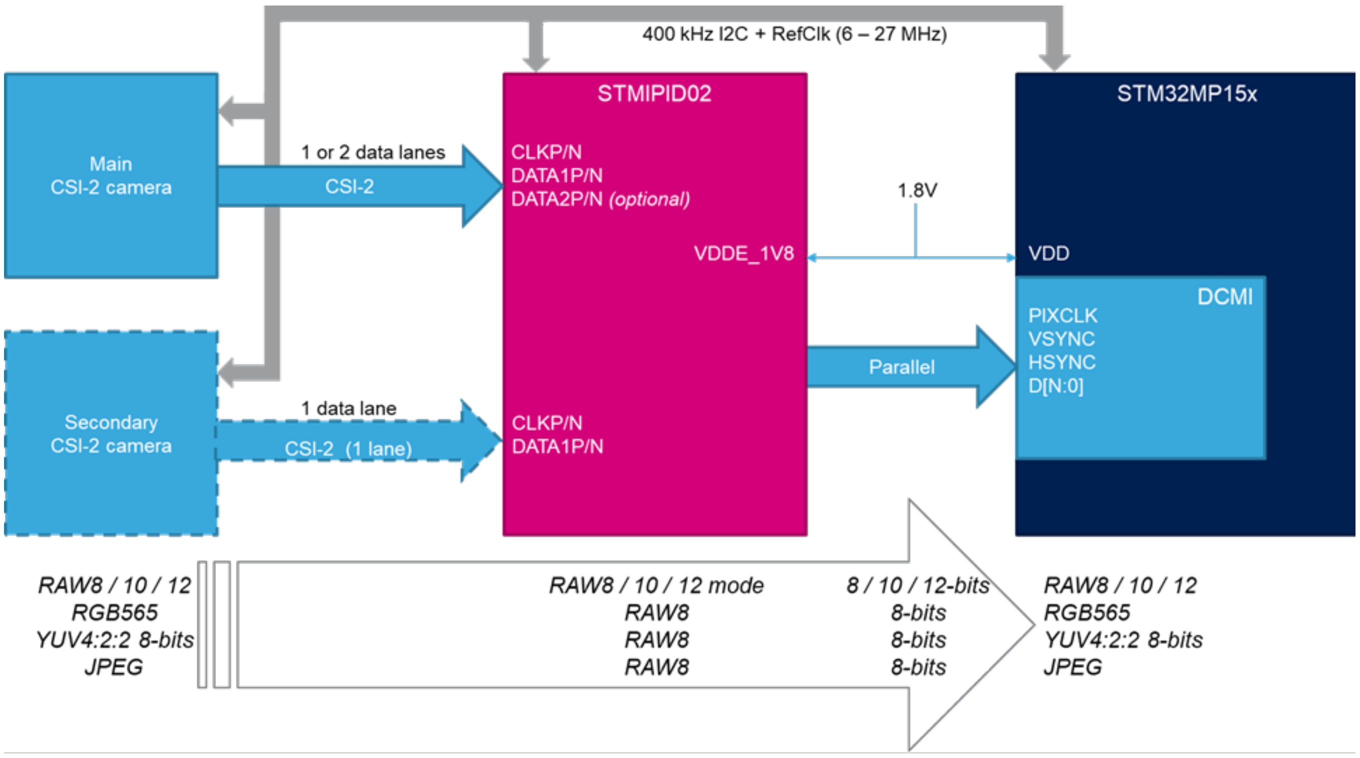
Reference number	Document title
	STMicroelectronics documents(1)
[R1]	STM32MP15x datasheets (DS12500, DS12501, DS12502, DS12503, DS12504, DS12505)
[R2]	STM32MP151x/3x/7x device errata (ES0438)
[R3]	Dual mode MIPI CSI-2 / SMIA CCP2 de-serializer (DS12803)
[R4]	Getting started with STM32MP151, STM32MP153 and STM32MP157 line hardware development (AN5031)
[R5]	STM32 Arm® Cortex®-based MPUs user guide: www.wiki.st.com/stm32mpu
[R6]	Linux® V4L2 camera framework: www.wiki.st.com/stm32mpu/wiki/V4L2_camera_overview
[R7]	STCubeProgrammer (flash programming tool): www.wiki.st.com/stm32mpu/wiki/STM32CubeProgrammer
[R8]	Device tree configuration: www.wiki.st.com/stm32mpu/wiki/DCMI_device_tree_configuration
[R9]	OpenST Linux distribution: www.wiki.st.com/stm32mpu/wiki/OpenSTLinux_distribution#Reference_source_code
[R10]	Directory structure for the OpenSTLinux Distribution package: www.wiki.st.com/stm32mpu/wiki/Example_of_directory_structure_for_Packages
[R11]	STM32CubeProgrammer software tool: www.st.com/en/development-tools/stm32cubeprog.html
[R12]	www.wiki.st.com/stm32mpu/wiki/How_to_populate_the_SD_card_with_dd_command
[R13]	wiki.st.com/stm32mpu/wiki/How_to_use_USB_mass_storage_in_U-Boot
[R14]	wiki.st.com/stm32mpu/wiki/Yavta
[R15]	wiki.st.com/stm32mpu/wiki/I2C_i2c-tools
[R16]	wiki.st.com/stm32mpu/wiki/GStreamer_overview
	Open source software resources(2)
[R17]	www.arrow.com (d3cameramezzov5640/d3-engineering)
[R18]	DH96 Board information: www.dh-electronics.com (/index.php/Avenger96)
[R19]	www.github.com
[R20]	www.github.com (dh-electronics/manifest-av96)
[R21]	www.dh-electronics.com (/index.php/Avenger96_Image_Programming)

1. *Available at www.st.com. Contact STMicroelectronics when more information is needed.*
2. *This URL belongs to a third party. It is active at document publication, however STMicroelectronics shall not be liable for any change, move or inactivation of the URL or the referenced material.*

3 STM32MP15x Series interfacing with the STMIPID02 MIPI CSI-2 deserializer

The STM32MP15x Series MPU family products do not natively implement a MIPI CSI-2 interface but embed a DCMI parallel port based on a MIPI CPI interface. It can be connected through the STMIPID02 MIPI CSI-2 deserializer to address any compatible MIPI CSI-2 camera sensor device. The STMIPID02 MIPI CSI-2 deserializer is connected to a MIPI CSI-2 camera on one side, and to the STM32MP15x Series DCMI12-bit data parallel interface on the other side. An overview of the block diagram is shown in the figure below.

Figure 1. Block diagram overview



3.1 MIPI CSI-2 versus MIPI CPI interface

The MIPI CSI-2 pinout saving is interesting when compared to a MIPI CPI interface. A MIPI CPI data port requires a minimum of eight data lines (of a maximum of 12 data lines), one clock, two synchronization lines, where a MIPI CSI-2 data port requires 2-wire differential pair per lane, and the clock lane.

3.2 Power supply considerations

Considering that the STMIPID02 deserializer bridge external power supply voltage pins are limited to 1.8 V, the STM32MP15x Series must be supplied at V_{DD} = 1.8 V (instead of nominal 3.3 V) to avoid the presence of level shifters on the DCMI interface clocks and I^2C signals. All the different power voltages to the STM32MP15x Series are supplied through the external PMIC module (power management integrated circuit).

For detailed information about the overall schematics, refer to the DH96 board information [R18]. In order to configure the STM32MP15x Series in V_{DD} = 1.8 V supply voltage, refer to [R4].

Regarding the OV5640 camera sensor, the V_{DD} power supply for the I/Os and the LDO (low-dropout regulator) external source power source is set to 1.8 V. For the analog logic, a 2.8 V must also be supplied as well as the external source.

3.3 STM32MP15x Series video throughput performance through DCMI

The MIPI CSI-2.1 interface can theoretically achieve data throughput rates up to 2.5 Gbyte/s per lane with a D-PHY. This is hardly achievable on a parallel port interface due firstly to I/O slew rate pin constraints on a general-purpose device such as the STM32MP15x Series, which only have a MIPI CPI interface. Secondly because the MPU requires to process this large amount of data fast enough to sustain a continuous frame rate from the camera.

For instance, a 5 Mpixel sensor with 16-bit per pixel, and a frame rate of 30 frames/s, gives a data throughput of 300 Mbyte/s processed continuously. This target is hardly achievable on a parallel port interface. Therefore, the sensor image data throughput must be reduced by adjusting the image frame rate, the resolution, and the pixel depth, or a combination of these.

From the OV5640 sensor to the STM32MP15x Series MPU through the STMIPID02 deserializer bridge it is possible to continuously acquire images with the following resolutions and frames below.

- 720 p 1280 × 720 RGB 565 27 fps
- 720 p 1280 × 720 YUYV 27 fps
- 720 p 1280 × 720 JPEG 27 fps
- HD 1920 × 1080 RGB 565 13 fps
- HD 1920 ×1080 YUYV 13 fps
- HD 1920 ×1080 JPEG 6 fps
- 5 Mpixel 2592 × 1944 RGB565 3 fps
- 5 Mpixel 2592 × 1944 YUYV 3 fps
- 5 Mpixel 2592 ×1944 JPEG 3 fps

The maximum performance achieved is 24 Mpixel/s or the equivalent of 1.3 Mpixel at 18.5 fps. This limitation is mostly due to the impact of the DCMI internal latency constraints as previously described in this section.

3.4 STMIPID02 Linux driver

The STMIPID02 MIPI CSI-2 deserializer bridge is designed to address a broad range of MIPI CSI-2 sensors targeting the consumer market and specifically mobile phone applications. To satisfy the growing demand to address this type of sensors from the industrial to IoT (Internet of Things) market through the artificial intelligence domain, the STMIPID02 driver has been upstreamed to the Linux community. It is freely available in Linux-based applications. The STMIPID02 bridge driver is included in the STMicroelectronics OpenSTLinux delivery package, starting from version 1.1.0 and above.

4 Overall application

The camera demonstrator is part of the GTK demo launcher application, based on the OpenSTLinux software distribution package. It is ported on DH96 boards featuring the STM32MP157A and the STMIPID02 on the DH96 Avenger board and the camera sensor OV5640 on the D3 Engineering DesignCore® camera mezzanine board.

4.1 DH Avenger96 board overview

The DH96 Avenger mother board integrates:

- ADH Core SOM module composed of the STM32MP157AAC microprocessor
- STPMIC1A power module
- 2 Mbytes × 512 Mbytes of DDR3L RAM
- STMIPID02 deserializer bridge
- 2-Mbytes SPIboot flash memory
- Connectivity and expansion connectors to receive the D3 DesignCore camera mezzanine board OV5640.

For more information, refer to [R18].

4.2 DH D3 Engineering DesignCore board overview

This mezzanine board is designed to fit the Avenger96 board through both high and low-speed expansion connectors for the STM32MP157 line. It allows connecting the OV5640 module camera over MIPI CSI-2 for evaluation purposes.

This board can connect a serial console, for instance GPIO PD1 and PB2 corresponding to UART4_TX and UART4_RX. It can be used to display the Linux kernel and boot stages. An optional USB/UART bridge can be used to connect to a console to the host PC through these pins.

4.3 Building the board image

The OpenSTLinux software distribution package targets STMicroelectronics application boards (STM32MP157C-DK2 and STM32MP15x Series-EVAL boards) and provides drivers, library, tools and examples to exercise STM32MP157 line embedded peripherals.

The software package for the third party DH96 Avenger and D3 Engineering camera boards is based on OpenSTLinux Distribution package. However it needs to be patched for the STM32MP157 line DCMI to natively address MIPI CSI-2 camera sensors through the STMIPID02 MIPI CSI-2 deserializer bridge.

Several options are listed below to build the board image, depending whether OpenSTLinux is already downloaded or not.

For image programming, many options are found at *www.st.com* from the related wiki page. The main options are detailed in this application note.

4.3.1 Fetching the manifest-av96 from the DH96 github repository

Downloading the latest git repository manifest.xml file that describes the directory structure and the links to the source files is probably the most straightforward option if the OpenSTLinux distribution package is not installed on the computer host. The patched distribution software is available on the github website for download, see [R20].

The Yocto manifest includes the DH Avenger96 board meta-layer “meta-av96”, in addition to the OpenSTLinux distribution package sources. A full build source environment is ready to be installed on the host PC to produce the built image targeting the DH96 Avenger board and the OV5640 camera sensor through the STMIPID02 deserializer bridge.

Refer to [R10] for guidelines to build a directory structure for the OpenSTLinux distribution package. Apply the following commands as mentioned in [R20] to build the board image.

```
PC $> cd <Distribution Package directory>/<distribution version>/
PC $> repo init -u https://github.com/dh-electronics/manifest-av96 -b thud
PC $> repo sync
PC $> source layers/meta-arrow/scripts/init-build-env.sh
PC $> bitbake av96-weston
```

4.3.2 Adding the meta-av96 layer on the top of the OpenSTLinux distribution package release

If the OpenSTLinux distribution package is previously installed on the PC host, another option is to add the Yocto board meta-layer « meta-av96 » on top of the OpenSTLinux distribution package.

To implement this, the DH Avenger96 board meta-layer must first be cloned locally before building the new image.

4.3.2.1 Clone of the git AV96 layer repository

```
PC $> cd <Distribution Package directory>/<OpenSTLinux distribution>/layers
PC $> git clone https://github.com/dh-electronics/meta-av96 -b thud
```

4.3.2.2 Setup the Yocto Bitbake environment for the new machine

```
PC $> cd ../
PC $> META_LAYER_ROOT=layers DISTRO=openstlinux-weston MACHINE=stm32mp1-av96
PC $> source layers/meta-st/scripts/envsetup.sh
```

4.3.2.3 Add the board meta-layer and run Bitbake

```
PC $> bitbake-layers add-layer ../layers/meta-av96
PC $> bitbake stm32mp1-av96
```

The process to create the board image can take several hours and approximatively 20 Gbytes of disk space. The image directory is created is the following directory:

`<Distribution-Package build directory>/tmp-glibc/deploy/images/stm32mp1-av96`

4.3.3 STM32CubeProgrammer software tool

This all-in-one software tool is an easy option to populate the binary image onto any flash device. The tool can be downloaded from [R11].

It is recommended to select the board H/W boot switches to DFU (or USB boot mode) for a faster programming as shown in the table below.

Table 3. **STM32CubeProgrammer software**

Boot mode	Comments	Boot 2 (switch 3)	Boot 1 (switch 2)	Boot 0 (switch 1)
UART and USB	USB OTG	1	1	0

Once the binary image is programmed:

1. Select the H/W boot switches to SDCard boot mode.
   a. Boot2 = 1
   b. Boot1 = 0
   c. Boot0 = 1

4.3.4 Other programming tools

These are the other programming tools available.

- To use the conventional “dd” command to program the binary image onto an SDcard, refer to the [R12] guidelines.
- It is recommended to program the whole binary image into an SDcard, however if the binary image partition must be split between the NOR-Flash (TF-A, U-boot) and e-mmc (Linux). For guidelines, refer to [R20].
- Use of U-boot to open a USB mass storage device, refer to [R13].

This method only applies if U-boot is available, therefore, if the two initial flash board partitions are already present into the SDCard or the board non-volatile memory.

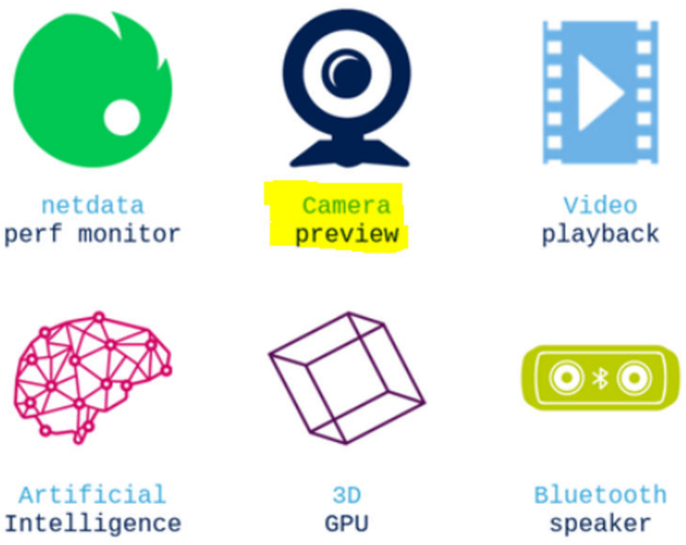
4.4 Booting the board image and camera preview display

Once the boot pin is changed to SDcard (or the end-user default image settings):

1. Connect the HDMI® to display monitor or TV.
2. The STMicroelectronics demonstrator must pop-up.
3. Plug the USB mouse.
4. Select the camera logo to stream live video from the OV5640 MIPI CSI-2 camera module.

The figure below shows the interface with the camera logo highlited in yellow.

Figure 2. Camera logo



4.4.1 V4L2 commands

There is a set of built-in V4L2 commands to communicate with the camera sensor. The most used command is `v4l2-ctrl`. It can retrieve all the camera sensor properties, review the V4L2 device node setup, and send commands to the sensor. The commands are detailed below.

To list the camera control menu

```
root@stm32mp1-av96:~# v4l2-ctl -L

User Controls

                       contrast 0x00980901 (int)     : min=0 max=255 step=1 default=0 value=0
flags=slider
                     saturation 0x00980902 (int)     : min=0 max=255 step=1 default=64 value=6
4 flags=slider
                            hue 0x00980903 (int)     : min=0 max=359 step=1 default=0 value=0
flags=slider
        white_balance_automatic 0x0098090c (bool)    : default=1 value=1 flags=update
                    red_balance 0x0098090e (int)     : min=0 max=4095 step=1 default=0 value=0
 flags=inactive, slider
                   blue_balance 0x0098090f (int)     : min=0 max=4095 step=1 default=0 value=0
 flags=inactive, slider
                       exposure 0x00980911 (int)     : min=0 max=65535 step=1 default=0 value=
972 flags=inactive, volatile
                 gain_automatic 0x00980912 (bool)    : default=1 value=1 flags=update
                           gain 0x00980913 (int)     : min=0 max=1023 step=1 default=0 value=1
9 flags=inactive, volatile
                horizontal_flip 0x00980914 (bool)    : default=0 value=0
                  vertical_flip 0x00980915 (bool)    : default=0 value=0
           power_line_frequency 0x00980918 (menu)    : min=0 max=3 default=1 value=1
                                0: Disabled
                                1: 50 Hz
                                2: 60 Hz
                                3: Auto

Camera Controls

                  auto_exposure 0x009a0901 (menu)    : min=0 max=1 default=0 value=0 flags=upd
ate
                                0: Auto Mode
                                1: Manual Mode

Image Processing Controls

                 link_frequency 0x009f0901 (intmenu): min=0 max=0 default=0 value=0 flags=rea
d-only
                                0: 384000000 (0x16e36000)
                   test_pattern 0x009f0903 (menu)    : min=0 max=4 default=0 value=0
                                0: Disabled
                                1: Color bars
                                2: Color bars w/ rolling bar
                                3: Color squares
                                4: Color squares w/ rolling bar
```

To show DCMI driver information

```
root@stm32mp1-av96:~# v4l2-ctl -d /dev/video0 -D
Driver Info:
        Driver name      : stm32-dcmi
        Card type        : STM32 Camera Memory Interface
        Bus info         : platform:dcmi
        Driver version   : X.Y.Z
        Capabilities     : 0x85200001
                Video Capture
                Read/Write
                Streaming
                Extended Pix Format
                Device Capabilities
        Device Caps      : 0x05200001
                Video Capture
                Read/Write
                Streaming
                Extended Pix Format
Media Driver Info:
        Driver name      : stm32-dcmi
        Model            : stm32-dcmi
        Serial           :
        Bus info         : platform:stm32-dcmi
        Media version    : X.Y.Z
        Hardware revision: 0x00000000 (0)
        Driver version   : X.Y.Z
Interface Info:
        ID               : 0x03000003
        Type             : V4L Video
Entity Info:
        ID               : 0x00000001 (1)
        Name             : stm32_dcmi
        Function         : V4L2 I/O
        Flags         : default
        Pad 0x01000002   : 0: Sink
          Link 0x02000009: from remote pad 0x1000008 of entity 'st-mipid02 1-0014': Data, Ena
bled, Immutable
```

To stream capture in RGB format and store it into a file

```
root@stm32mp1-av96:~# v4l2-ctl --set-fmt-video=width=1280,height=720,pixelformat=RGBP --strea
m-mmap --stream-count=1 --stream-to=file.raw
<
root@stm32mp1-av96:~# ls -ltr file.raw
-rw-r--r-- 1 root root 1843200 Jun 17 17:18 file.raw
```

To perform a full screen preview

```
root@stm32mp1-av96:~# gst-launch-1.0 v4l2src ! "video/x-raw, width=1280, Height=720, framerat
e=(fraction)15/1" ! queue ! autovideosink -e
```

For a detailed overview of the V4L2 Linux framework and camera driver, refer to [R6].

4.4.2 Application commands

The application commands control the camera resolution, frame rate and format in order to display a stream of images over an HDMI® or LCD screen based on the V4L2 framework, GStreamer or Yavta. They are provided by Linux and available in the OpenSTLinux distribution package.

4.4.2.1 GStreamer

GStreamer is an open source multimedia framework to handle video and sound over a variety of operating systems, such as GNU/Linux or Windows. It is based on concatenated pipeline commands to handle a video stream at the same time.

If GStreamer is not installed on the board image, use `apt-get` command from the board connected to internet.

```
root@stm32mp1-av96:~# apt-get update
…
Reading package lists... Done

root@stm32mp1-av96:~# apt-get install GStreamer
…
Preparing to unpack .../GStreamer_0.10.36-r2_armhf.deb ...
Unpacking GStreamer (0.10.36-r2) ...
Setting up GStreamer (0.10.36-r2) ...
```

This command previews in 640 × 480 a stream of images at 30 fps, connecting a display monitor to the Avenger96 board using an HDMI cable.

```
root@stm32mp1-av96:~# gst-launch-1.0 v4l2src device=/dev/video0 ! "video/x-raw, width=640,hei
ght=480,framerate=30/1” ! waylandsink &
```

This command takes 3 VGA JPEG in 640 x 480 and store them to files.

```
root@stm32mp1-av96:~# gst-launch-1.0 v4l2src num-buffers=3 ! "image/jpeg, width=640, height=4
80" ! queue ! multifilesink location=pic%05d.jpeg
```

This command checks that the produced output file has the right format.

```
root@stm32mp1-av96:~# gst-typefind-1.0 pic00000.jpeg
pic00000.jpeg - image/jpeg, width=(int)640, height=(int)480, sof-marker=(int)0
```

For more information refer to [R16].

4.4.2.2 Yavta

Based on the V4L2 framework, the Yavta test application allows the test, control, and debug the camera sensor. The commands are detailed below.

This command list all the sensor setup, the supported video formats and the frame rates.

```
root@stm32mp1-av96:~# yavta -l --enum-formats --enum-inputs /dev/video0
Device /dev/video0 opened.
Device `STM32 Camera Memory Interface' on `platform:dcmi' is a video output (without mplanes)
 device.
--- User Controls (class 0x00980001) ---
control 0x00980901 `Contrast' min 0 max 255 step 1 default 0 current 0.
control 0x00980902 `Saturation' min 0 max 255 step 1 default 64 current 64.
control 0x00980903 `Hue' min 0 max 359 step 1 default 0 current 0.
control 0x0098090c `White Balance, Automatic' min 0 max 1 step 1 default 1 current 1.
control 0x0098090e `Red Balance' min 0 max 4095 step 1 default 0 current 0.
control 0x0098090f `Blue Balance' min 0 max 4095 step 1 default 0 current 0.
control 0x00980911 `Exposure' min 0 max 65535 step 1 default 0 current 885.
control 0x00980912 `Gain, Automatic' min 0 max 1 step 1 default 1 current 1.
control 0x00980913 `Gain' min 0 max 1023 step 1 default 0 current 248.
control 0x00980914 `Horizontal Flip' min 0 max 1 step 1 default 0 current 0.
control 0x00980915 `Vertical Flip' min 0 max 1 step 1 default 0 current 0.
control 0x00980918 `Power Line Frequency' min 0 max 3 step 1 default 1 current 1.
  0: Disabled
  1: 50 Hz (*)
  2: 60 Hz
  3: Auto
--- Camera Controls (class 0x009a0001) ---
control 0x009a0901 `Auto Exposure' min 0 max 1 step 1 default 0 current 0.
  0: Auto Mode (*)
  1: Manual Mode
--- Image Processing Controls (class 0x009f0001) ---
control 0x009f0901 `Link Frequency' min 0 max 0 step 1 default 0 current 0.
  0: 384000000 (*)
control 0x009f0903 `Test Pattern' min 0 max 4 step 1 default 0 current 0.
  0: Disabled (*)
  1: Color bars
  2: Color bars w/ rolling bar
  3: Color squares
  4: Color squares w/ rolling bar
15 controls found.
- Available formats:
        Format 0: JPEG (4745504a)
        Type: Video capture (1)
        Name: JFIF JPEG
        Frame size: 176x144 (1/15, 1/30)
        Frame size: 320x240 (1/15, 1/30)
        Frame size: 640x480 (1/15, 1/30, 1/60)
        Frame size: 720x480 (1/15, 1/30)
        Frame size: 720x576 (1/15, 1/30)
        Frame size: 1024x768 (1/15, 1/30)
        Frame size: 1280x720 (1/15, 1/30)
        Frame size: 1920x1080 (1/15, 1/30)
        Frame size: 2592x1944 (1/15, 1/30)

        Format 1: UYVY (59565955)
        Type: Video capture (1)
        Name: UYVY 4:2:2
        Frame size: 176x144 (1/15, 1/30)
        Frame size: 320x240 (1/15, 1/30)
        Frame size: 640x480 (1/15, 1/30, 1/60)
        Frame size: 720x480 (1/15, 1/30)
        Frame size: 720x576 (1/15, 1/30)
        Frame size: 1024x768 (1/15, 1/30)
        Frame size: 1280x720 (1/15, 1/30)
        Frame size: 1920x1080 (1/15, 1/30)
        Frame size: 2592x1944 (1/15, 1/30)

        Format 2: YUYV (56595559)
        Type: Video capture (1)
        Name: YUYV 4:2:2
        Frame size: 176x144 (1/15, 1/30)
        Frame size: 320x240 (1/15, 1/30)
        Frame size: 640x480 (1/15, 1/30, 1/60)
        Frame size: 720x480 (1/15, 1/30)
        Frame size: 720x576 (1/15, 1/30)
        Frame size: 1024x768 (1/15, 1/30)
        Frame size: 1280x720 (1/15, 1/30)
        Frame size: 1920x1080 (1/15, 1/30)
        Frame size: 2592x1944 (1/15, 1/30)

        Format 3: RGB565 (50424752)
        Type: Video capture (1)
        Name: 16-bit RGB 5-6-5
        Frame size: 176x144 (1/15, 1/30)
        Frame size: 320x240 (1/15, 1/30)
        Frame size: 640x480 (1/15, 1/30, 1/60)
        Frame size: 720x480 (1/15, 1/30)
        Frame size: 720x576 (1/15, 1/30)
        Frame size: 1024x768 (1/15, 1/30)
        Frame size: 1280x720 (1/15, 1/30)
        Frame size: 1920x1080 (1/15, 1/30)
        Frame size: 2592x1944 (1/15, 1/30)

- Available inputs:
        Input 0: Camera.

Video format: JPEG (4745504a) 320x240 (stride 320) field none buffer size 76800
```

This command writes 10 frames on the disk at default format resolution.

```
root@stm32mp1-av96:~# yavta -F /dev/video0 --capture=10
Device /dev/video0 opened.
Device `STM32 Camera Memory Interface' on `platform:dcmi' is a video output (without mplanes)
 device.
Video format: JPEG (4745504a) 320x240 (stride 320) field none buffer size 76800
8 buffers requested.
length: 76800 offset: 0 timestamp type/source: mono/EoF
Buffer 0/0 mapped at address 0xb6e2e000.
length: 76800 offset: 77824 timestamp type/source: mono/EoF
Buffer 1/0 mapped at address 0xb6e1b000.
length: 76800 offset: 155648 timestamp type/source: mono/EoF
Buffer 2/0 mapped at address 0xb6e08000.
length: 76800 offset: 233472 timestamp type/source: mono/EoF
Buffer 3/0 mapped at address 0xb6df5000.
length: 76800 offset: 311296 timestamp type/source: mono/EoF
Buffer 4/0 mapped at address 0xb6de2000.
length: 76800 offset: 389120 timestamp type/source: mono/EoF
Buffer 5/0 mapped at address 0xb6dcf000.
length: 76800 offset: 466944 timestamp type/source: mono/EoF
Buffer 6/0 mapped at address 0xb6dbc000.
length: 76800 offset: 544768 timestamp type/source: mono/EoF
Buffer 7/0 mapped at address 0xb6da9000.
Warning: bytes used 6144 != image size 76800 for plane 0
0 (0) [-] none 0 6144 B 398.623775 398.623910 24.408 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
1 (1) [-] none 1 6144 B 398.657082 398.657199 30.024 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
2 (2) [-] none 2 6144 B 398.690402 398.690512 30.012 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
3 (3) [-] none 3 6144 B 398.723708 398.723812 30.025 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
4 (4) [-] none 4 6144 B 398.757021 398.757134 30.018 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
5 (5) [-] none 5 6144 B 398.790334 398.790439 30.018 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
6 (6) [-] none 6 6144 B 398.823651 398.823753 30.015 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
7 (7) [-] none 7 6144 B 398.856968 398.857067 30.015 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
8 (0) [-] none 8 6144 B 398.890280 398.890381 30.019 fps ts mono/EoF
Warning: bytes used 6144 != image size 76800 for plane 0
9 (1) [-] none 9 6144 B 398.923596 398.923693 30.016 fps ts mono/EoF
Captured 10 frames in 0.340887 seconds (29.335169 fps, 180235.280085 B/s).
8 buffers released.
```

For more information on Yavta refer to [R14].

5 Guidelines to interface with another MIPI CSI-2 camera sensor

This section provides guidelines to allow the end-user to build its own application as well as guidelines to replace the OV5640 camera sensor by another MIPI CSI-2 camera sensor model.

When choosing the camera sensor, it is important to note that neither the STMIPID02 nor the STM32MP15x Series DCMI can decode video frames from the camera sensor. To avoid the CPU overhead due to frame decoding, the new camera sensor must either embeds an image sensor processor (ISP) as in the OV5640 camera sensor, or the processing requires to be deported in a discrete component in the application board.

The STMIPID02 bridge and the OmniVision® OV5640 camera sensor drivers are provided in theOpenSTLinux Expansion Package delivery. By default, the Linux driver may not be included for the new camera sensor.

This section describes the necessary steps to replace the OV5640 camera sensor by another camera sensor in the end-user application. All the following steps related to the OV5640 camera sensor must be transposed with the new camera sensor.

5.1 I^{2}C probing

Once the new camera is plugged onto the board, the initial step is to enable its access through the I^{2}C port. The camera sensor driver is based on this, to allow the access to the sensor device registers.

Prior to this step, reset and power-down signals must be driven through GPIOs. This is performed either directly, or through a board GPIO expander controlled by I^{2}C with the correct drive polarity as per the sensor specifications.

5.1.1 Patching the camera sensor driver for debug

Since the power and clock are dynamically controlled by the camera sensor driver, whenever the module is probed or not, the driver must be temporarily patched to maintain the power link when the driver module is probed. It is possible to read or write the register values by using i2ctools or a debugging tool while the camera is up and running. For the OV5640 camera sensor, the driver can be found at this location:

```
<distri_pack_dir>/build-openstlinuxweston-stm32mp1-av96/tmp-glibc/work-shared/stm32
mp1-av96/kernel-source/drivers/media/i2c/ov5640.c
```

This patch disables the power-off capability when the OV5640 camera module is probed, this can be reproduced also on the new camera driver.

```
-     ov5640_power(sensor, false);
-     regulator_bulk_disable(OV5640_NUM_SUPPLIES, sensor->supplies);

 xclk_off:
     clk_disable_unprepare(sensor->xclk);
     return ret;
@@ -1878,9 +1878,9 @@ static int ov5640_set_power_on(struct ov5640_dev *sensor)

 static void ov5640_set_power_off(struct ov5640_dev *sensor)
 {
-     ov5640_power(sensor, false);
-     regulator_bulk_disable(OV5640_NUM_SUPPLIES, sensor->supplies);
-     clk_disable_unprepare(sensor->xclk);
 }

 static int ov5640_set_power(struct ov5640_dev *sensor, bool on)
@@ -2886,8 +2886,8 @@ static int ov5640_probe(struct i2c_client *client,
     mutex_init(&sensor->lock);

     ret = ov5640_check_chip_id(sensor);
-     if (ret)
-         goto entity_cleanup;

     ret = ov5640_init_controls(sensor);
     if (ret)
@@ -2897,6 +2897,7 @@ static int ov5640_probe(struct i2c_client *client,
     if (ret)
         goto free_ctrls;

+     dev_info(dev, " probe OK\n");
     return 0;
```

5.1.2 Patching the STMIPID02 driver for debug

The driver can be found in the OpenSTLinux distribution package directory:

`<distri_pack_dir>/build-openstlinuxweston-stm32mp1-av96/tmp-glibc/work-shared/stm32mp1-av96/kernel-source/drivers/media/i2c/st-mipid02.c`

Exactly like the camera sensor, the clocks and power lines require to be forced on, and the reset line disabled while debugging, in order to probe the STMIPID02 device.

```
static int mipid02_probe(struct i2c_client *client,
[...]
power_off:
entity_cleanup:
static int mipid02_remove(struct i2c_client *client)
[...]



static void mipid02_apply_reset(struct mipid02_dev *bridge)
{

#if 0
    gpiod_set_value_cansleep(bridge->reset_gpio, 0);
    usleep_range(5000, 10000);
    gpiod_set_value_cansleep(bridge->reset_gpio, 1);
    usleep_range(5000, 10000);
    gpiod_set_value_cansleep(bridge->reset_gpio, 0);
#else
    gpiod_set_value_cansleep(bridge->reset_gpio, 0);
    usleep_range(5000, 10000);
#endif
}
```

5.1.3 Recompiling the kernel modules for debug

In order to ease the module manipulation, ensure that the dcmi, stmipid02 and ov5640 are declared as out-of-tree modules in the .config file located at: `<distribution-package>/<build>/tmp-glibc/work-shared/stm32mp1-av96/kernel-build-artifacts/`

```
CONFIG_VIDEO_OV5640=m
CONFIG_VIDEO_STM32_DCMI=m
CONFIG_VIDEO_ST_MIPID02=m
```

After applying the patch, the kernel modules requires to be recompiled and reloaded onto the following target:

```
PC $> bitbake virtual/kernel -C compile
```

The kernel objects ov5640.ko and st-mipid02.ko, or modules, are updated in the directory:

`<distri_pack_dir>/build-openstlinuxweston-stm32mp1-av96/tmp-glibc/work /stm32mp1_av96-ostl-linux-gnueabi/linux-stm32mp/<kernel_version>/image /lib/modules/<kernel version>/kernel/drivers/media/i2c`

and uploaded in the board SDcard rootfs partition:

`<rootfs>/lib/modules/<kernel version>/kernel/drivers/media/i2c`

To recreate a new image for debug to include the patched module, invoke a Bitbake command as follows on top of the previous one:

```
PC $> bitbake av96-weston
```

5.1.4 Probing the modules using I^{2}C tools

The I^{2}C-tools package is a collection of I2C Linux commands that are embedded in the OpenSTLinux distribution package and available to probe the camera sensor and the STMIPID02.

To list all I^{2}C bus connected and listed on board perform the following command:

```
root@stm32mp1-av96:~# i2cdetect -l
i2c-1   i2c           STM32F7 I2C(0x40013000)                 I2C adapter
i2c-2   i2c           STM32F7 I2C(0x5c002000)                 I2C adapter
i2c-0   i2c           STM32F7 I2C(0x40012000)                 I2C adapter
```

To list the I2C device registers connected to a specific I^{2}C bus, for instance i2c-1 corresponds to the STM32MP15x Series I2C2 peripheral. Perform the following command:

```
root@stm32mp1-av96:~# i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- UU -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The UU suggests that the register address is currently used. The `i2cdetect` command cannot probe this address since a driver is currently controlling this resource. For instance the I^{2}C address 0x3c (reported in the device tree file) points to the OV5640 camera and is detected as busy since already accessed by the DCMI Linux driver.

Once the ov5640.ko or the patched image is reloaded onto the target, the `I2C` command can now access the target when the module is probed. To retrieve the ID code of the OV5640 sensor, reading from the sensor register address 0x300a using I2C2 slave@0x3c, the command `i2cget` is used.

```
root@stm32mp1-av96:~# i2cset -f -y 1 0x3c 0x30 0x0a
root@stm32mp1-av96:~# i2cget -f -y 1 0x3c
0x56
root@stm32mp1-av96:~# i2cget -f -y 1 0x3c
0x40
```

The read value 0x5640 matches the published ID code for the OV5640 sensor. which is even more straightforward using the `i2ctransfer` command.

```
root@stm32mp1-av96:~# i2ctransfer -y -f 1 w2@0x3c 0x30 0x0a r2
0x56 0x40
```

The camera sensor is now accessible from the STM32MP15x Series, the camera registers can be set according to the sensor software specifications.

The STMIPID02 is controlled by i2c-2 corresponding to I2C4 in the board device tree @0x14.

```
root@stm32mp1-av96:~# i2ctransfer -y -f 2 w2@0x14 0x00 0x14 r1
0x42
```

For more information on i2c-tools refer to [R15].

5.2 STMIPID02 MIPI D-PHY clock settings

The camera sensor drives the MIPI CSI-2 signals, or serial links, to the STMIPID02. This includes the D-PHY TX clock connected to the STMIPID02 D-PHY RX, or the clock lane. Moreover, the STMIPID02 must adjust the sampling clock lane frequency according to the sensor bit rate. The camera pixel data rate is known to the driver. It is computed by the V4L2 (video for Linux framework) where the pixel width, height, and frame rate are passed as arguments by the API command to the driver. V4L2_CID_PIXEL_RATE and V4L2_CID_LINK_FREQ are read by the stmipid02 driver to calculate the clock lane frequency.

Depending on the frame format used, when the number of bits per pixel is deterministic (as for RGB, RAW), the clock lane can be derived from the desired data rate, and therefore directly from the V4L2_CID_PIXEL_RATE variable.

- Where MIPI clock frequency = pixel clock frequency × number-bits-per-pixel ÷ ((number of lanes) ÷ 2).
- When the number of bits per pixel is not known to the driver, like in YUV or JPEG format the size of the image can vary after each frame. The blanking period varies also after every line sent.

The MIPI D-PHY clock lane can be measured using an oscilloscope with a high-speed differential probe when the camera is active. The MIPI CSI-2 data rate should be set with the following formula:

- MIPI CSI-2 data rate = (MIPI clock frequency × 2) × number of data lanes ≥ pixel clock × number-bits-per-pixel.
- Where MIPI clock freq = pixel clock × number-bits-per-pixel ÷ ((number of lanes) ÷ 2).

On the OV5640 camera, the clock lane frequency can be measured using an oscilloscope:

- For a 640 × 480 RGB565 15 fps: 62 MHz
- For a 640 × 480 RGB565 30 fps: 120 MHz

The maximum throughput rate for a continuous video stream out the camera is:

- 24 Mpixel/s maximum therefore 48 Mbyte/s or 384 Mbit/s on 2 lines, or 192 Mbit/s per line.

For STMIPID02 clock lane 1 register (address 0x02). The following settings match the OV5640 clock lane sampling.

```
i2ctransfer -y -f 1 w3@0x14 0x00 0x02 0x19
i2ctransfer -y -f 1 w3@0x14 0x00 0x02 0x21
i2ctransfer -y -f 1 w3@0x14 0x00 0x02 0x29
```

These settings must potentially be tuned again if another camera sensor is mounted.

5.3 Checking data signal polarities

When the I^2C link is set between the camera sensor and the STM32MP15x Series, all clocks and data signal polarities must match between STM32MP15x Series, STMIPID02 and the camera sensor. These signals can be physically probed through the hardware board test points, and can be tunes using the I^2C register write commands. For data and clock camera sensor signals propagated through the MIPI CSI-2 bus, these are probed after the STMIPID02 deserializer bridge. Once all the probed signals have the same polarity, the signal polarity settings can be reported in the board device tree and verified at run time using the following commands detailed below.

Scripts description

1. Use the following prerequisite script.

```
echo "#!/bin/bash" > dtdumpentry.sh;echo "hexdump -e '\"=\"'  -e '20/1 \"%c\"\"\t\"' -e
'20/1 \"%02x\"\"\n\"' \$1" >> dtdumpentry.sh;chmod +x dtdumpentry.sh
 echo "#!/bin/bash" > dtdump.sh;echo "find  \$1* -type f -print0 -exec ./dtdumpentry.sh
{} \;" >> dtdump.sh;chmod +x dtdump.sh
```

2. Create an executable script with the following.

```
rm devicetree.txt
 echo "[devicetree]" >> devicetree.txt
 echo "|-[dcmi]" >> devicetree.txt
 ./dtdump.sh /proc/device-tree/soc/dcmi | sed 's/\/proc\/device-tree\/soc\//| |-/' >> de
vicetree.txt
 echo "|" >> devicetree.txt
 echo "|-[camera:" | tr -d "\n" >> devicetree.txt
 cat /proc/device-tree/soc/i2c*/camera*/compatible >> devicetree.txt
 echo "]" >> devicetree.txt
 ./dtdump.sh /proc/device-tree/soc/i2c*/camera* -type f -print0 -exec ./dtdump.sh {} \;
| sed 's/\/proc\/device-tree\/soc\//| |-/' >> devicetree.txt
 echo "" >> devicetree.txt
 cat devicetree.txt
```

3. Run the created script.
   The newly created script lists the camera device tree to check how the DCMI and the camera sensor ports are configured from the Linux perspective. It also checks that the signal polarities are correctly configured.

Script results

DCMI ports signals description.

```
root@stm32mp1-av96:~# ./runme.sh
[devicetree]
|-[dcmi]
…
| |-dcmi@4c006000/port/endpoint/hsync-active=       00000000
| |-dcmi@4c006000/port/endpoint/vsync-active=       00000000
| |-dcmi@4c006000/port/endpoint/remote-endpoint=    0000003d
| |-dcmi@4c006000/port/endpoint/pclk-max-frequency= 0496ed40
| |-dcmi@4c006000/port/endpoint/bus-width=          00000008
| |-dcmi@4c006000/port/endpoint/pclk-sample=        00000000

| |-dcmi@4c006000/port/endpoint/pclk-max-frequency= 0496ed40 (77 MHz)
```

For HSYNC and VSYNC signals, it is possible to monitor their pin polarity through the DCMI status register (DCMI_SR). It is also possible to check if these pins are correctly connected to the DCMI interface.

- Camera sensor CSI ports signals description.

```
-[camera:ovti,ov5640]
| |-i2c@40013000/camera@3c/port/endpoint/data-lanes= 0000000100000002

| |-i2c@40013000/camera@3c/port/endpoint/clock-lanes=        00000000

| |-i2c@40013000/camera@3c/port/endpoint/pclk-max-frequency= 0496ed40
```

PWRDWN and RESET signal polarity settings

- Camera sensor GPIO signal polarity description.

```
…
-[camera:ovti,ov5640]

| |-i2c@40013000/camera@3c/powerdown-gpios=    0000001d0000000500000000
| |-i2c@40013000/camera@3c/reset-gpios=        0000001c0000000c00000001
```

The reset (active low) and power down (active high) pins are driven through GPIOs. For the OV5640 over the D3 Engineering board, GPIOA-12 `(gpio-12)` and GPIOB-5 `(gpio-21)` are used respectively to release the sensor after power up.

```
root@stm32mp1-av96:~# cat /sys/kernel/debug/gpio
gpiochip0: GPIOs 0-15, parent: platform/soc:pin-controller@50002000, GPIOA:

 gpio-0   (                    |wakeup              ) in  hi

 gpio-12  (                    |reset               ) out hi

gpiochip1: GPIOs 16-31, parent: platform/soc:pin-controller@50002000, GPIOB:

 gpio-21  (                    |powerdown           ) out lo
```

As per the STMIPID02 reset (active low) the pin which needs to be GPIO driven. For the OV5640 over the D3 engineering board, GPIOZ-0 `(gpio-400)` is used.

```
root@stm32mp1-av96:~# cat /sys/kernel/debug/gpio
…
gpiochip9: GPIOs 400-415, parent: platform/soc:pin-controller-z@54004000, GPIOZ:

 gpio-400 (                    |reset               ) out hi
```

5.4 Frame interrupt handling

The DCMI peripheral features a DMA request capability to handle capture frames from the camera sensor to store them in the memory. For fixed frame formats it means that the sensor sends the same amount of data for each frame. The DMA can then detect each frame from a camera stream and send an end-of-transfer interrupt to signal to the CPU that a single frame is ready to be processed. However, this does not apply for JPEG frame format where the data amount per frame is not deterministic. In such a case, the DCMI end-of-frame interrupt, triggered by VSYNC, aborts the DMA transfer as shown below.

```
root@stm32mp1-av96:~# cat /proc/interrupts | grep dcmi
 52:          0          0     GIC-0 110 Level     4c006000.dcmi
```

5.5 Frame format customization

Some applications require to add specific frame coding format (YUV, RGB, RAW). The OV5640 camera sensor embeds an integrated image sensor processor that allows convert the internal data format among a list of supported frame formats and byte ordering. The STMIPID02 and the DCMI have no frame formatting capabilities, all camera supported formats must be created and listed accordingly in these Linux drivers.

In the OV5640 Linux driver, a list of the supported format in either 8-bit or 16-bit per pixel is given below. The STMIPID02 and the DCMI drivers must also include these formats if they are used in the application.

```
static const struct ov5640_pixfmt ov5640_formats[] = {
        { MEDIA_BUS_FMT_JPEG_1X8, V4L2_COLORSPACE_JPEG, },
        { MEDIA_BUS_FMT_UYVY8_2X8, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_YUYV8_2X8, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_RGB565_2X8_LE, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_RGB565_2X8_BE, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_SBGGR8_1X8, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_SGBRG8_1X8, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_SGRBG8_1X8, V4L2_COLORSPACE_SRGB, },
        { MEDIA_BUS_FMT_SRGGB8_1X8, V4L2_COLORSPACE_SRGB, },
};
```

5.6 Camera sensor and STMIPID02 input clock source

In the DH Avenger96 board design, the OV5640 and STMIPID02 input clock pins are both driven from a single clock source signal and connected to the STM32MP15x Series MCO1 pin, as defined in the board device tree. This method saves the cost of an additional external oscillator.

The MCO1 output pin is internally connected to the STM32MP15x Series external hi-speed oscillator (HSE clock source) which provides a reliable clock source with a frequency of 24 MHz.

In order to drive the MCO1 clock to both the camera sensor and the STMIPID02, only one pin muxing declaration must be done. The camera sensor device tree node (ov5640) must host the clock input pin muxing definition, as detailed below.

```
ov5640: camera@3c {
         compatible = "ovti,ov5640";
         reg = <0x3c>;
+        pinctrl-names = "default", "sleep";
+        pinctrl-0 = <&rcc_pins_a>;
+        pinctrl-1 = <&rcc_sleep_pins_a>;

@@ -598,9 +601,6 @@
         compatible = "st,st-mipid02";
         reg = <0x14>;
         status = "okay";
-        pinctrl-names = "default", "sleep";
-        pinctrl-0 = <&rcc_pins_a>;
-        pinctrl-1 = <&rcc_sleep_pins_a>
```

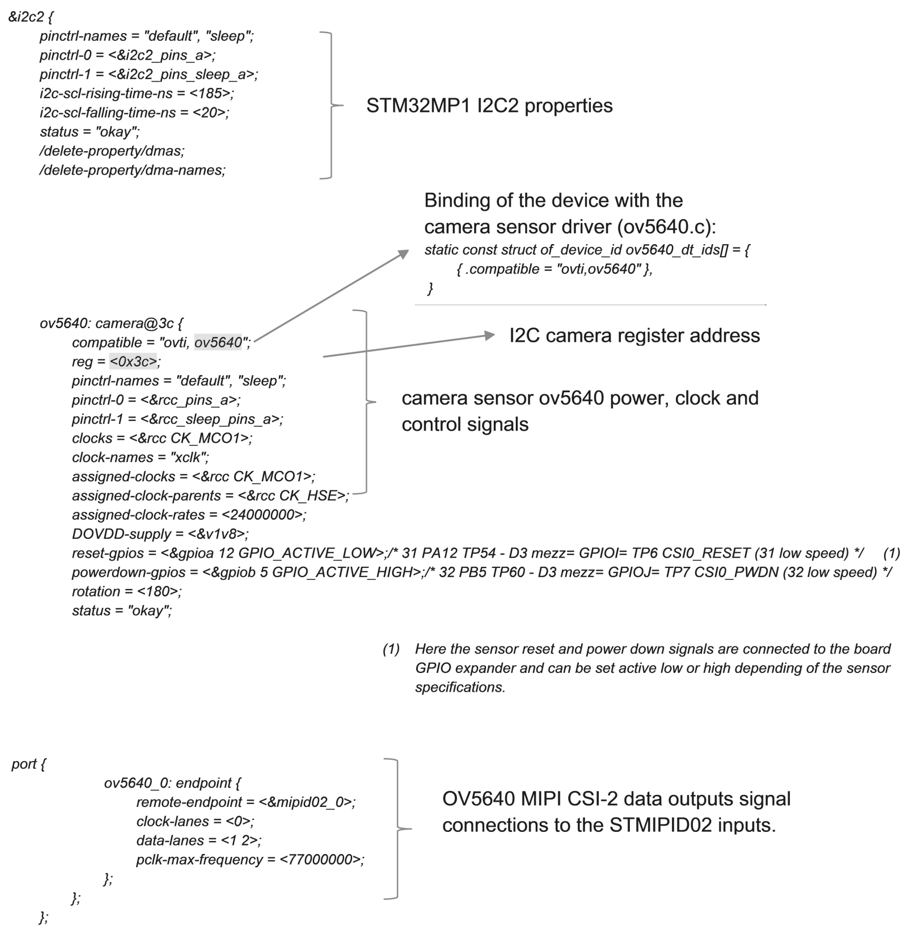
5.7 Device tree

The STM32 microprocessor device tree file contains the hardware description of the STM32MP157 line microprocessor which includes the DCMI (STM32MP15x camera interface) device tree node, stm32mp157c.dtsi file found at: `<kernel_source_dir>/arch/arm/boot/dts`

The description of the stm32mp157c.dtsi file contains the STM32 DCMI hardware specific features inherent to STM32MP15x Series that must not be modified unless the requirement of specific image formats must be implemented.

At a higher level, the boot loader U-boot loads the board specific compiled DTS file (DTB) that includes the DTSI file, with the kernel image. The camera sensor device is controlled by the STM32MP15x Series I^{2}C driver via the V4L2 (video for Linux version 2) framework. Because the I^{2}C bus has no dynamic discoverability features at runtime to enumerate the device (unlike the USB protocol for instance) the camera device properties must be fully described in the board DST file prior being compiled.

The OV5640 camera sensor device is controlled by the STM32MP157 line I2C2 peripheral, which is probably the same with any new camera sensor. The description starts with the I2C2 bus properties to match the camera device specifications. Then it must match with the OV5640 camera sensor power, clock and board connections signals, and finally with the port connections between the camera sensor and the STMIPID02 bridge. All the points mentioned need to be transposed in the new camera sensor. Detailed information on the device tree is shown below. The board device tree for the Avenger96 board can be found at: `<kernel-source>/arch/arm/boot/dts/stm32mp157a-av96.dts`



```
&i2c2 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&i2c2_pins_a>;
        pinctrl-1 = <&i2c2_pins_sleep_a>;
        i2c-scl-rising-time-ns = <185>;
        i2c-scl-falling-time-ns = <20>;
        status = "okay";
        /delete-property/dmas;
        /delete-property/dma-names;

        ov5640: camera@3c {
                compatible = "ovti, ov5640";
                reg = <0x3c>;
                pinctrl-names = "default", "sleep";
                pinctrl-0 = <&rcc_pins_a>;
                pinctrl-1 = <&rcc_sleep_pins_a>;
                clocks = <&rcc CK_MCO1>;
                clock-names = "xclk";
                assigned-clocks = <&rcc CK_MCO1>;
                assigned-clock-parents = <&rcc CK_HSE>;
                assigned-clock-rates = <24000000>;
                DOVDD-supply = <&v1v8>;
                reset-gpios = <&gpioa 12 GPIO_ACTIVE_LOW>;/* 31 PA12 TP54 - D3 mezz= GPIOI= TP6 CSI0_RESET (31 low speed) */     (1)
                powerdown-gpios = <&gpiob 5 GPIO_ACTIVE_HIGH>;/* 32 PB5 TP60 - D3 mezz= GPIOJ= TP7 CSI0_PWDN (32 low speed) */
                rotation = <180>;
                status = "okay";

port {
                        ov5640_0: endpoint {
                                remote-endpoint = <&mipid02_0>;
                                clock-lanes = <0>;
                                data-lanes = <1 2>;
                                pclk-max-frequency = <77000000>;
                        };
                };
        };
```

Binding of the device with the camera sensor driver (ov5640.c):

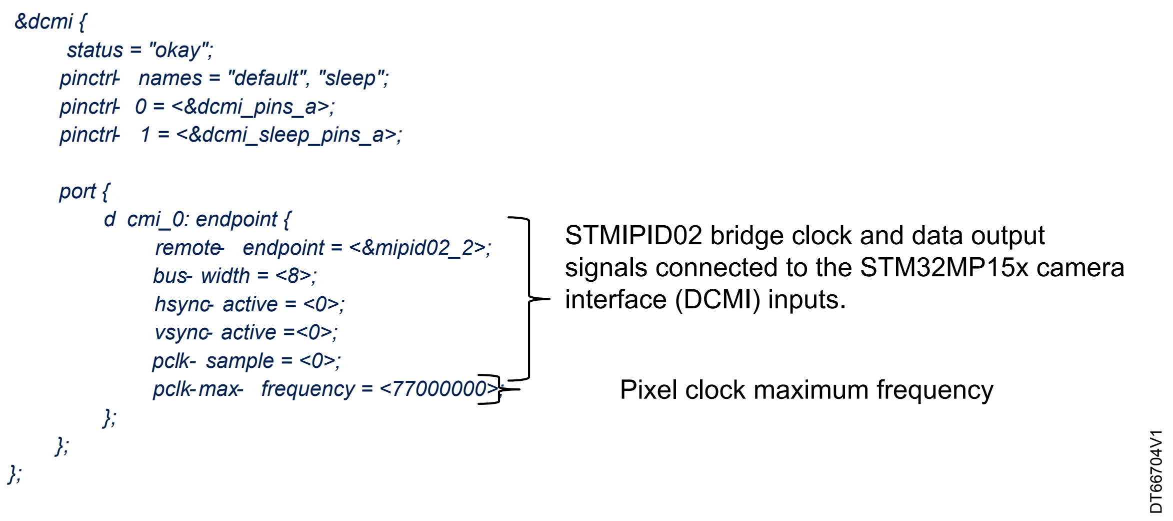
```
static const struct of_device_id ov5640_dt_ids[] = {
        { .compatible = "ovti,ov5640" },
}
```

STM32MP1 I2C2 properties
I2C camera register address
camera sensor ov5640 power, clock and control signals
OV5640 MIPI CSI-2 data outputs signal connections to the STMIPID02 inputs.

(1) *Here the sensor reset and power down signals are connected to the board GPIO expander and can be set active low or high depending of the sensor specifications.*

Figure 3. Device tree - part 1

The control connections between the STM32MP15x Series camera interface and the STMIPID02 deserializer bridge is found further down in this DST file, this must not be modified.



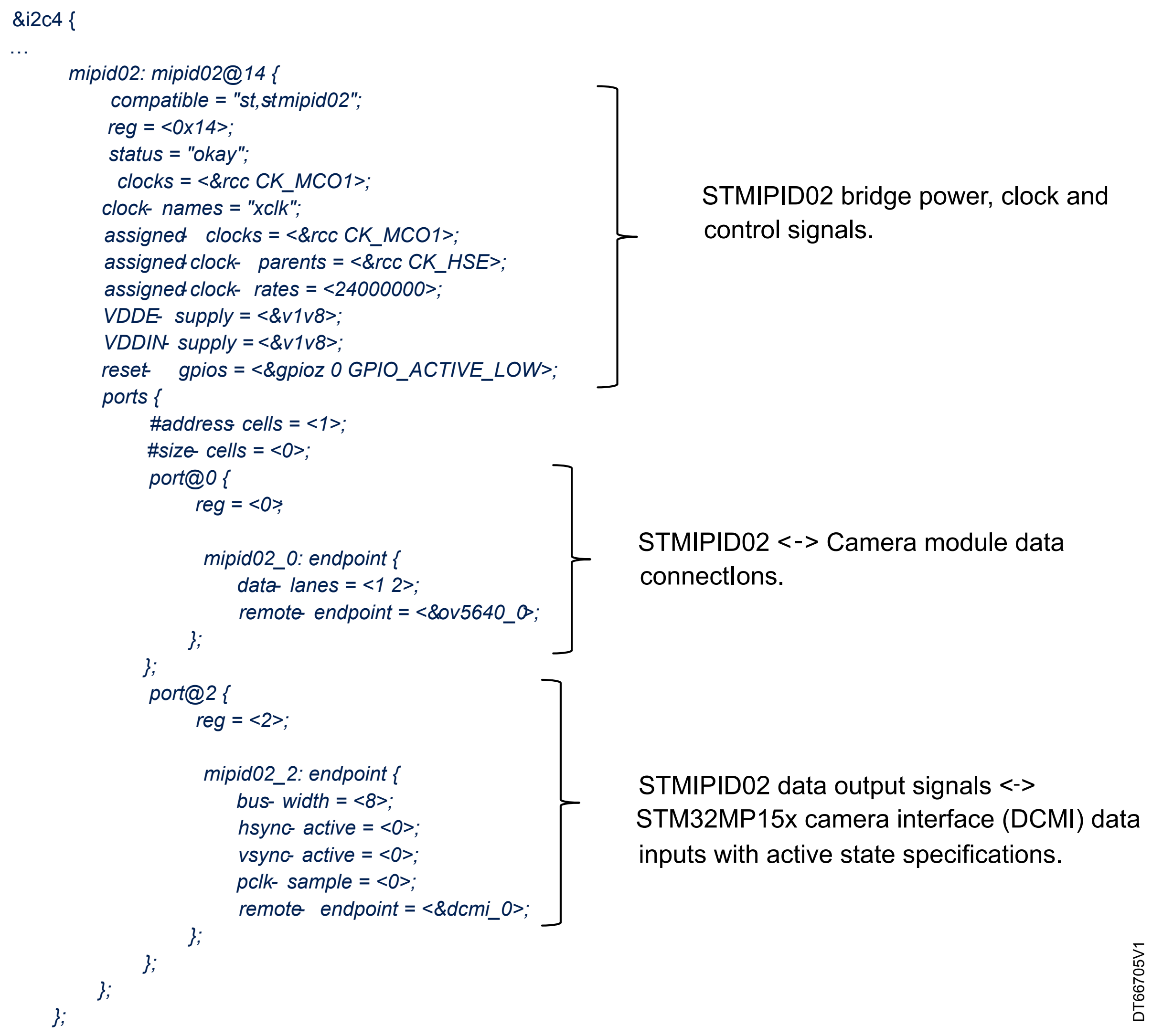
```
&dcmi {
    status = "okay";
    pinctrl-  names = "default", "sleep";
    pinctrl-  0 = <&dcmi_pins_a>;
    pinctrl-  1 = <&dcmi_sleep_pins_a>;

    port {
        d  cmi_0: endpoint {
            remote-   endpoint = <&mipid02_2>;
            bus-  width = <8>;
            hsync-  active = <0>;
            vsync-  active =<0>;
            pclk-  sample = <0>;
            pclk-max-   frequency = <77000000>;
        };
    };
};
```

STMIPID02 bridge clock and data output signals connected to the STM32MP15x camera interface (DCMI) inputs.
Pixel clock maximum frequency
DT66704V1

Figure 4. **Device tree - part 2**

The STMIPID02 bridge deserializer is controlled by the STM32MP15x Series I2C4 peripheral, it references the connections on both sides of the bridge, (STMIPID02 sensor, and STMIPID02-STM32MP15x/DCMI).

Figure 5. Device tree - part 3



```
&i2c4 {
...
        mipid02: mipid02@14 {
              compatible = "st,st-mipid02";
              reg = <0x14>;
              status = "okay";
              clocks = <&rcc CK_MCO1>;
              clock-names = "xclk";
              assigned-clocks = <&rcc CK_MCO1>;
              assigned-clock-parents = <&rcc CK_HSE>;
              assigned-clock-rates = <24000000>;
              VDDE-supply = <&v1v8>;
              VDDIN-supply = <&v1v8>;
              reset-gpios = <&gpioz 0 GPIO_ACTIVE_LOW>;
              ports {
                     #address-cells = <1>;
                     #size-cells = <0>;
                     port@0 {
                            reg = <0>;

                             mipid02_0: endpoint {
                                  data-lanes = <1 2>;
                                  remote-endpoint = <&ov5640_0>;
                            };
                     };
                     port@2 {
                            reg = <2>;

                             mipid02_2: endpoint {
                                  bus-width = <8>;
                                  hsync-active = <0>;
                                  vsync-active = <0>;
                                  pclk-sample = <0>;
                                  remote-endpoint = <&dcmi_0>;
                            };
                     };
              };
       };
```

STMIPID02 bridge power, clock and control signals.

STMIPID02 <-> Camera module data connections.

STMIPID02 data output signals <-> STM32MP15x camera interface (DCMI) data inputs with active state specifications.

5.8 Checking the GPIO pin connections

Once the connections are finalized, it is important to check that all the camera sensor signals propagating through the STMIPID02 are correctly configured from the STM32MP15x Series host perspective.

5.8.1 STM32MP15x Series camera interface GPIOs settings

This can be easily done using the CubeIDE tool. An STMicroelectronics graphical interface tool to configure all STM32MP15x Series required GPIOs, below for instance the DCMI interface.

```
 root@stm32mp1-av96:~# cat /sys/kernel/debug/pinctrl/soc\:pin-controller*/pin* | grep dcmi
pin 4 (PA4): device 4c006000.dcmi function af13 group PA4
pin 6 (PA6): device 4c006000.dcmi function af13 group PA6
pin 9 (PA9): device 4c006000.dcmi function af13 group PA9
pin 23 (PB7): device 4c006000.dcmi function af13 group PB7
pin 64 (PE0): device 4c006000.dcmi function af13 group PE0
pin 65 (PE1): device 4c006000.dcmi function af13 group PE1
pin 70 (PE6): device 4c006000.dcmi function af13 group PE6
pin 119 (PH7): device 4c006000.dcmi function af13 group PH7
pin 122 (PH10): device 4c006000.dcmi function af13 group PH10
pin 126 (PH14): device 4c006000.dcmi function af13 group PH14
pin 129 (PI1): device 4c006000.dcmi function af13 group PI1
pin 132 (PI4): device 4c006000.dcmi function af13 group PI4
pin 134 (PI6): device 4c006000.dcmi function af13 group PI6
```

For the GPIO pins and associated function (AF13) refer to specific DCMI signals that can be checked in the STM32MP15x Series datasheets, refer to Table 2. Reference documents

Table 4. **GPIO pins and associated function**

STM32MP15x Series GPIO pin	STM32MP15x Series alternate function (AF13)
PA4	DCMI_HSYNC
PA6	DCMI_PIXCLK
PA9	DCMI_D0
PB7	DCMI_VSYNC
PE0	DCMI_D2
PE1	DCMI_D3
PE6	DCMI_D7
PH7	DCMI_D9
PH10	DCMI_D1
PH14	DCMI_D4
PI1	DCMI_D8
PI4	DCMI_D5
PI6	DCMI_D6

Note: *10-bit data are physically connected to the DCMI peripheral interface, 8-bit of data DCMI[7:0] are effectively handled by the STMIPID02 and DCMI driver in MIPI CSI-2 mode. This is handled in the STMIPID02 register settings (Mode_Reg1 bit7) matching the STMIPID02 specifications for the selected image format.*

5.9 Debug and trace commands

This section gives some useful Linux console commands to probe the sensor or to ensure that the camera device driver is correctly loaded by the Linux kernel.

5.9.1 Trace command

The `dmesg` command must be used to print (or control) the kernel ring buffer, for trace purposes. By default, only errors and warnings are traced. The output to the console by changing the granularity level of details to the maximum (level 8) allows all traces to be outputted synchronously.

The command`$> dmesg -n8` prints all trace debug (dev_dbg) in the console. Beware that this command displays a large amount of trace data. It is a good practice to be selective on the search. This command traces only the kernel messages related to the camera sensor ov5640.

```
root@stm32mp1-av96:~# dmesg | grep ov5640
[    8.329669] ov5640 0-003c: Linked as a consumer to regulator.15
[    8.329718] ov5640 0-003c: 0-003c supply DVDD not found, using dummy regulator
[    8.329818] ov5640 0-003c: Linked as a consumer to regulator.0
[    8.329846] ov5640 0-003c: 0-003c supply AVDD not found, using dummy regulator
```

5.9.2 Camera module probing

When using `rmmod` and `modprobe`-commands it is possible to respectively unbind and bind the device drivers after Linux has booted. These commands are useful for checking the current status of the device, and whether it is correctly loaded by the Linux kernel. Otherwise the log error hints on the source of the problem. For example if a pin muxing GPIO alternate function conflicts with another peripheral sharing the same pin.

`modeprobe` can also retrieve the product device ID, and give details on which peripheral it is used to communicate with that device. It can also list all the dependency modules connected to the camera sensor.

```
root@stm32mp1-av96:~#modprobe -D ov5640
insmod /lib/modules/4.19.49/kernel/drivers/media/media.ko
insmod /lib/modules/4.19.49/kernel/drivers/media/v4l2-core/videodev.ko
insmod /lib/modules/4.19.49/kernel/drivers/media/v4l2-core/v4l2-common.ko
insmod /lib/modules/4.19.49/kernel/drivers/media/v4l2-core/v4l2-fwnode.ko
insmod /lib/modules/4.19.49/kernel/drivers/media/i2c/ov5640.ko
```

5.9.3 Dynamic debug

Performing a dynamic debug allows dynamically to retrieve kernel information from a specific driver, such as, DCMI dev_dbg messages, without having to patch and recompile the module. Unbinding/Binding the DCMI driver is an alternative to rmmod/modprobe to manually disconnect a driver from a device from the end-user space, while keeping the dynamic debug configuration active for this module. This probes output debug traces.

```
root@stm32mp1-av96:~# dmesg -n8
root@stm32mp1-av96:~# echo "module stm32_dcmi +p" > /sys/kernel/debug/dynamic_debug/control
root@stm32mp1-av96:~# echo -n "4c006000.dcmi" >  /sys/bus/platform/drivers/stm32-dcmi/unbind;
echo -n "4c006000.dcmi" >  /sys/bus/platform/drivers/stm32-dcmi/bind
[ 5431.821221] stm32-dcmi 4c006000.dcmi: Removing video0
[ 5431.830087] stm32-dcmi 4c006000.dcmi: Device registered as video0
[ 5431.841444] stm32-dcmi 4c006000.dcmi: Subdev "st-mipid02 2-0014" bound
[ 5431.849759] stm32-dcmi 4c006000.dcmi: DCMI is now linked to "st-mipid02 2-0014"
[ 5431.858627] stm32-dcmi 4c006000.dcmi: Supported fourcc/code: RGBP/0x1008
[ 5431.866478] stm32-dcmi 4c006000.dcmi: Supported fourcc/code: YUYV/0x2008
[ 5431.871725] stm32-dcmi 4c006000.dcmi: Supported fourcc/code: UYVY/0x2006
[ 5431.882724] stm32-dcmi 4c006000.dcmi: Supported fourcc/code: JPEG/0x4001
[ 5431.895508] stm32-dcmi 4c006000.dcmi: Probe done
```

The dynamic debug trace enables to monitor the stm32-dcmi driver bindings linked to the bus identifier of the device. Moreover, it enables to retrieve the Fourcc format identifier supported by the DCMI driver, and to ensure that the STMIPID02 is correctly bound as a sub-device to the DCMI.

```
root@stm32mp1-av96:~#echo "module st_mipid02 +p" > /sys/kernel/debug/dynamic_debug/
control
```

```
[ 1244.381391] st-mipid02 2-0014: mipid02_set_fmt for 0
[ 1244.444326] st-mipid02 2-0014: mipid02_get_fmt probe 0
[ 1244.448281] st-mipid02 2-0014: mipid02_s_stream : requested 1 / current = 0
[ 1244.455136] st-mipid02 2-0014: detect link_freq = 384000000 Hz
[ 1244.465744] st-mipid02 2-0014: mipid02_s_stream current now = 1 / 0
```

5.9.4 Checking the frame rate

To check for any discrepancies between the frequency at which the individual images are produced and measured by the v4l2 framework, and the frame rate command set to the sensor, the following v4l2-ctl command can be run.

```
root@stm32mp1-av96:~# v4l2-ctl --set-parm=15;

root@stm32mp1-av96:~# v4l2-ctl --set-fmt-video=width=1280,height=720,pixelformat=RGBP --strea
m-mmap --stream-count=-1 &

Frame rate set to 15.000 fps

<<<<<<<<<<<<<<<<< 15.14 fps
<<<<<<<<<<<<<<<<< 15.14 fps
…
```

This is detailed as follows:

- Desired frame rate

```
root@stm32mp1-av96:~# v4l2-ctl --set-parm=15;
```

- Measured frame rate

```
<<<<<<<<<<<<<<<<< 15.14 fps
<<<<<<<<<<<<<<<<< 15.14 fps
…
```

5.9.5 Comparing jpg files

The following result is a combination of v4l2 and GStreamer to binary compare jpg files (two test pattern images and one capture frame).

```
root@stm32mp1-av96:~# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=JPEG --set-ct
rl=test_pattern=1 --set-parm=30 --stream-mmap --stream-count=1 --stream-to=sanity-colorbars-0
.jpeg
Frame rate set to 30.000 fps
<
root@stm32mp1-av96:~# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=JPEG --set-ct
rl=test_pattern=1 --set-parm=30 --stream-mmap --stream-count=1 --stream-to=sanity-colorbars-1
.jpeg
Frame rate set to 30.000 fps
<
root@stm32mp1-av96:~# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=JPEG --set-ct
rl=test_pattern=0 --set-parm=30 --stream-mmap --stream-count=1 --stream-to=sanity-nocolorbars
.jpeg
Frame rate set to 30.000 fps
<
root@stm32mp1-av96:~# echo "**** Camera sanity check ****" >> camerasanity.txt
cho root@stm32mp1-av96:~# echo "Similarities between 2 consecutives pictures (should be > 0.9
):" >> camerasanity.txt
root@stm32mp1-av96:~# gst-launch-1.0 filesrc location= sanity-colorbars-0.jpeg ! decodebin !
compare name=cmp method=ssm
 meta=none threshold=0  ! fakesink  filesrc location= sanity-colorbars-1.jpeg ! decodebin ! c
mp. -v --gst-debug=*BUS*:5 2
>&1 | grep "content=" | grep dispatch  | awk -F")" '{print $(NF)}' >> camerasanity.txt
root@stm32mp1-av96:~# echo "Similarities with control picture (should be < 0.7):" >> camerasa
nity.txt
root@stm32mp1-av96:~# gst-launch-1.0 filesrc location= sanity-colorbars-0.jpeg ! decodebin !
compare name=cmp method=ssim
 meta=none threshold=0  ! fakesink  filesrc location= sanity-nocolorbars.jpeg ! decodebin ! c
mp. -v --gst-debug=*BUS*:5 2
>&1 | grep "content=" | grep dispatch | awk -F")" '{print $(NF)}' >> camerasanity.txt
root@stm32mp1-av96:~# weston-image sanity-colorbars-0.jpeg sanity-colorbars-1.jpeg sanity-noc
olorbars.jpeg &

root@stm32mp1-av96:~# sleep 5
root@stm32mp1-av96:~# kill %1
root@stm32mp1-av96:~# cat camerasanity.txt
**** Camera sanity check ****
Similarities between 2 consecutives pictures (should be > 0.9):
1;
Similarities with control picture (should be < 0.7):
0.63874228088429863;
```

Revision history

Table 5. **Document revision history**

Date	Version	Changes
03-Sep-2020	1	Initial version
26-Sep-2023	2	STM32MP1 RPN changed to STM32MP15x

Contents

List of tables

List of figures