Introduction to digital camera interface pixel pipeline for STM32 MCUs

Introduction

As the demand for capturing and processing high-quality images increases, the imaging domain continually evolves, giving rise to advanced technologies across a wide range of applications.

Nowadays, fields such as automotive, security, and home automation introduce new requirements like computer vision and image processing. Consequently, there is a growing need for robust image acquisition capabilities at the edge, which is ensured by DCMIPP on the STM32 microcontrollers.

The STM32 MCUs offer CPU, MCU subsystem, DSP, and FPU. They also provide an extensive set of peripheral and interface combinations, such as XSPI, UART, I2C, SDIO, USB, Ethernet, or I2S. Additionally, STM32 MCUs boast a rich graphical portfolio, including LTDC, DSI, GPU2D, GFXMMU, and DMA2D. Furthermore, they are supported by an industry-leading development environment that ensures sophisticated applications and connectivity solutions (IoT).

This application note provides the STM32 users with essential information about the DCMIPP peripheral along with configuration and examples on how to use it.

Table 1. Applicable products

Type	Product lines
Microcontroller	STM32H7R3/7S3 and STM32H7R7/7S7
	STM32N6x5 and STM32N6x7

1 General information

The STM32 devices are 32-bit microcontrollers based on the Arm® Cortex® processor.

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Reference documents

[1] *STM32H7Rx/7Sx Arm®-based 32-bit MCUs*, reference manual (RM0477)

[2] STM23H7Sxxx datasheet (DS14359)

[3] STM23H7Rxxx datasheet (DS14360)

[4] *STM32N647xx/657xx Arm®-based 32-bit MCUs, reference manual (RM0486)*

[5] STM32N6x5xx/STM32N6x7xx datasheet (DS14791)

[6] *Getting started with Octo-SPI, Hexadeca-SPI, and XSPI interfaces on STM32 MCUs*, application note (AN5050).

[7] *Discovery kit with STM32H7S7L8 MCU*, user manual (UM3289)

[8] *Introduction to LCD-TFT display controller (LTDC) on STM32 MCUs*, application note (AN4861)

[9] *Discovery kit with STM32N657X0 MCU*, user manual (UM3300)

2 Digital camera interface pixel pipeline description

2.1 Introduction

This section provides a general overview of the DCMIPP peripheral availability across the STM32H7R7/7S7 and STM32N6x5/6x7 products and gives an easy-to-understand explanation on the DCMIPP integration in the STM32 MCUs architecture.

The DCMIPP provides communication interfaces with high-resolution camera sensor and a pixel processing pipeline to process the acquired pixels into a usable form by the application. Two acquisition interfaces are available: a parallel interface with pins for pixel clock and HSYNC and VSYNC synchronization signals as well as a CSI-2 compliant serial interface to capture pixels from camera sensors. The pixel pipeline applies processing such as decimation, cropping, downsizing, color conversion, gamma correction directly on the received pixels.

It offers higher performance compared to former MCUs by moving from AHB slave (DCMI) to AXI master, increasing data transfer rates from 48 Mbytes to 2800 Mbytes. It supports modern CSI-2 interfaces, accommodating high-resolution sensors.

Additionally, the DCMIPP features a dedicated applicative pipe that spares postprocessing and bandwidth by adapting resolution and pixel format to the application needs. Typically, it downsizes on-the-fly to 1080p-YUV420 for video encoding and to QVGA-RGB for NPU analysis.

The DCMIPP application offers high-speed uncompressed image capture for superior image quality and supports compressed image capture in JPEG format for efficient storage space utilization.

The DCMIPP offers flexible operating modes to support most camera sensors.

It supports both color and monochrome cameras, using various data formats. It is compatible with multiple types of external sensors, including:

- Raw Bayer sensors without internal ISP, which output unprocessed raw Bayer pixels
- Sensors with an internal ISP, which usually output RGB or YUV pixels
- Sensors supporting video compression, which output a bit-stream, such as JPEG or H.264
- Sensors with interlaced video, which output their odd and even fields sequentially

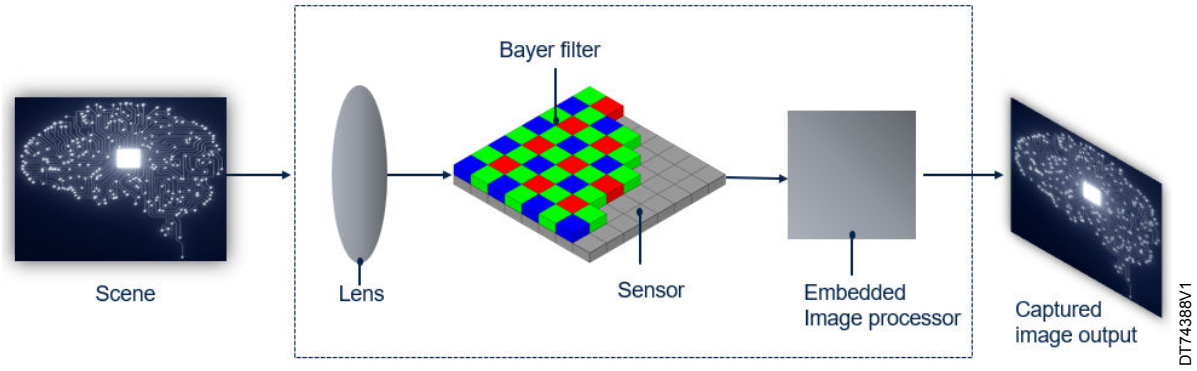
2.2 Camera module

The camera module integrates the following components:

- Lens: Focuses light onto the sensor
- Bayer matrix and filter: Used to capture color information by filtering the incoming light into red, green, and blue components
- Optionally, an image signal processor (ISP): Processes images, converting the raw data from the sensor into a usable format
- Data interface such as CSI-2 interface or parallel interface

These elements work together to capture and process images. Digital cameras typically generate YUV, RGB, and raw Bayer output formats. The sensor modules, in particular, generate Raw bayer output format with bit depths that can vary, including 8, 10, 12, and 14 bits. This raw Bayer format preserves the unprocessed data from the sensor, allowing for greater flexibility and precision in postprocessing and image enhancement.

Figure 1. Camera module components



2.3 Hardware interfaces

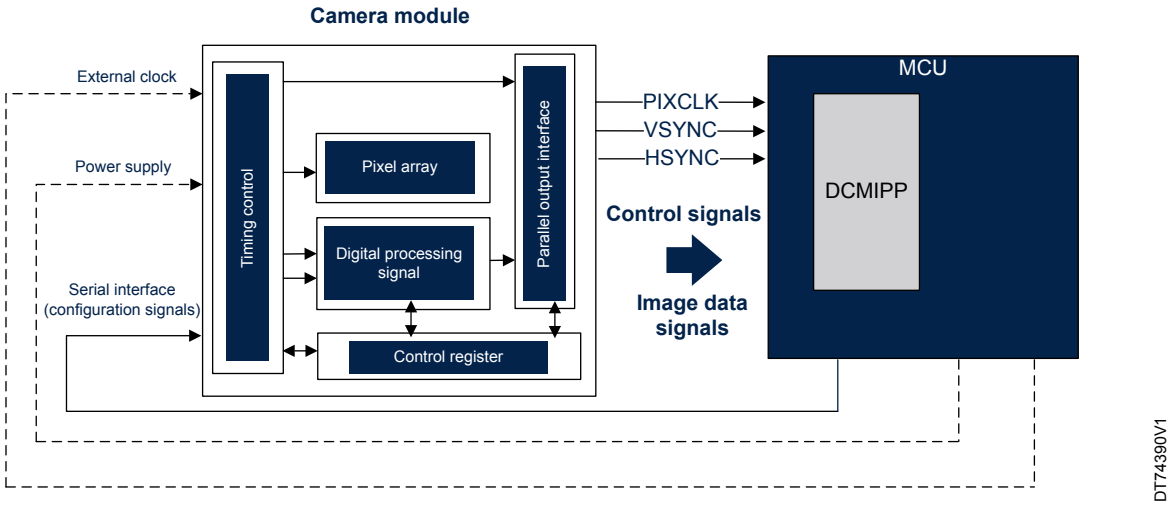
The camera module can be interfaced with an MCU using two types of interfaces: parallel and CSI-2.

2.3.1 Parallel interface

A camera module requires four main types of signals to transmit image data properly: control signals, image data signals, power supply signals, and camera configuration signals.

Figure 2 illustrates a typical interconnection with an MCU.

Figure 2. Parallel interfacing a camera module with an STM32 MCU



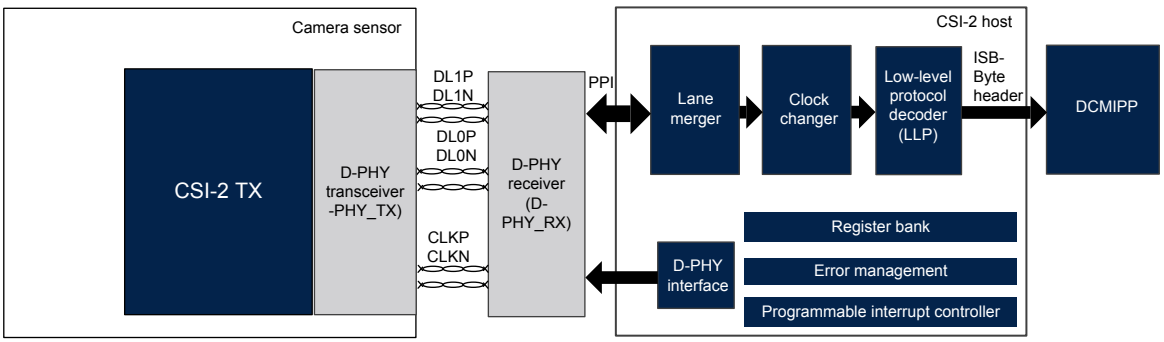
The interconnection has the following characteristics:

- **Control signals** are used for clock generation and data transfer synchronization
  The camera clock must be provided according to the camera specification. The camera also provides two data synchronization signals: HSYNC (for horizontal/line synchronization) and VSYNC (for vertical/frame synchronization).
- **Image data signals**
  Each of them transmits a bit of the image data. Their width represents the number of bits to be transferred at each pixel clock. This number depends on both the parallel interface of the camera module and the embedded system interface.
- **Power supply signals**
  As any embedded electronic system, the camera module needs to have a power supply. The operating voltage of the camera module is specified in Operating condition section inside the device datasheet reference.
- **Configuration signals** are used for the following:
  - Configuring image resolution, format, frame rate, contrast, and brightness, over a dedicated I2C interface, even for the parallel interface (to the sensor).
  - The remaining configuration signal used to select the type of interface could be a combination of a sensor pin configuration and/or a firmware initialization sequence sent to the sensor over I2C.

Note: Most camera modules are parametrized through an I²C communication bus.

2.3.2 CSI-2 interface

Figure 3. Serial interfacing a camera module with an STM32 MCU



The camera serial interface 2 (CSI-2) is part of a group of communication protocols defined by the MIPI® Alliance.

The MIPI® CSI-2 Host controller is a digital core that implements all protocol functions defined in the MIPI® CSI-2 v1.2 specification.

It provides an interface between the system and the MIPI® D-PHY, enabling communication with CSI-2 compliant cameras. The CSI-2 host connects external serial cameras to the CSI-2 Host controller, using unidirectional slave data lanes and clock lanes to handle data transmission from CSI-2 protocol-based camera devices to subsystems for data processing handled by the DCMIPP.

The STM32N6 interfaces with serial camera modules by incorporating the MIPI® CSI-2 Host controller. This controller, equipped with a D-PHY receiver physical layer, efficiently and cost-effectively extracts and decodes data packets, processing them frame by frame from the CSI-2 device to the host.

The CSI-2 host connects external serial cameras to the CSI-2 Host controller. It is based on unidirectional slave data lanes and clock lanes to handle data transmission from the CSI-2 protocol-based camera devices to subsystems for data processing handled by the DCMIPP.

2.4 DCMIPP availability and features across STM32 MCUs

Table 2 summarizes the STM32 products embedding the DCMIPP. The table also highlights the availability of other hardware resources that facilitates DCMIPP operation. These hardware resources can also be used with the DCMIPP in the same application.

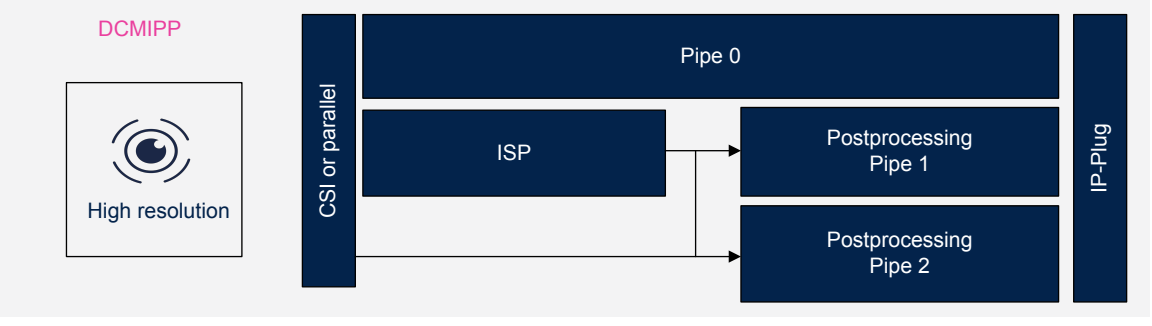
Table 2. DCMIPP capabilities

STM32	Max DCMIPP pixel clock input freq. (MHz)	Max Kernel clock freq. (MHz)	Max DCMIPP Interconnect clock freq. (MHz)	Pipe0 (dump pipe)	Pipe1 (Main Pipe)	Pipe2 (ancillary pipe)	Serial interface (CSI-2)	Parallel interface
STM32H7R3/7S3 STM32H7R7/7S7	120	250	400	Yes	No	No	No	Yes
STM32N6x5 STM32N6x7	120	333	250	Yes	Yes	Yes	Yes	Yes

2.5 DCMIPP architecture

The DCMIPP handles image acquisition and processing through the integrated parallel and abutted CSI-2 interfaces. On the acquisition path, DCMIPP reads input from these interfaces, while on the output path, it writes to memory through the IP-Plug block, acting as an AXI master and using output FIFOs to manage pixel transfers and address memory latency issues. The DCMIPP features include ISP functions in Pipe 1 (shareable with Pipe 2) and differentiated postprocessing pipelines. Pipe 1 and Pipe 2 offer cropping, downsizing, and pixel formatting capabilities.

Figure 4. DCMIPP overview



The DCMIPP offloads the MCU from all image acquisition tasks through up-to-three dedicated camera pipelines, incorporating ISP features for sensor modules such as:

- RGB Bayer pattern processing
- Bad pixel correction
- Decimation
- Black level adjustment
- Exposure control
- Demosaicing
- Color conversion
- Contrast enhancement

Additionally, it offers robust postprocessing functionalities including:

- Cropping
- Decimation
- Downsizing
- Region of interest (ROI) selection
- Gamma correction
- YUV conversion
- Pixel packing

2.5.1 STM32H7R7/7S7 system architecture

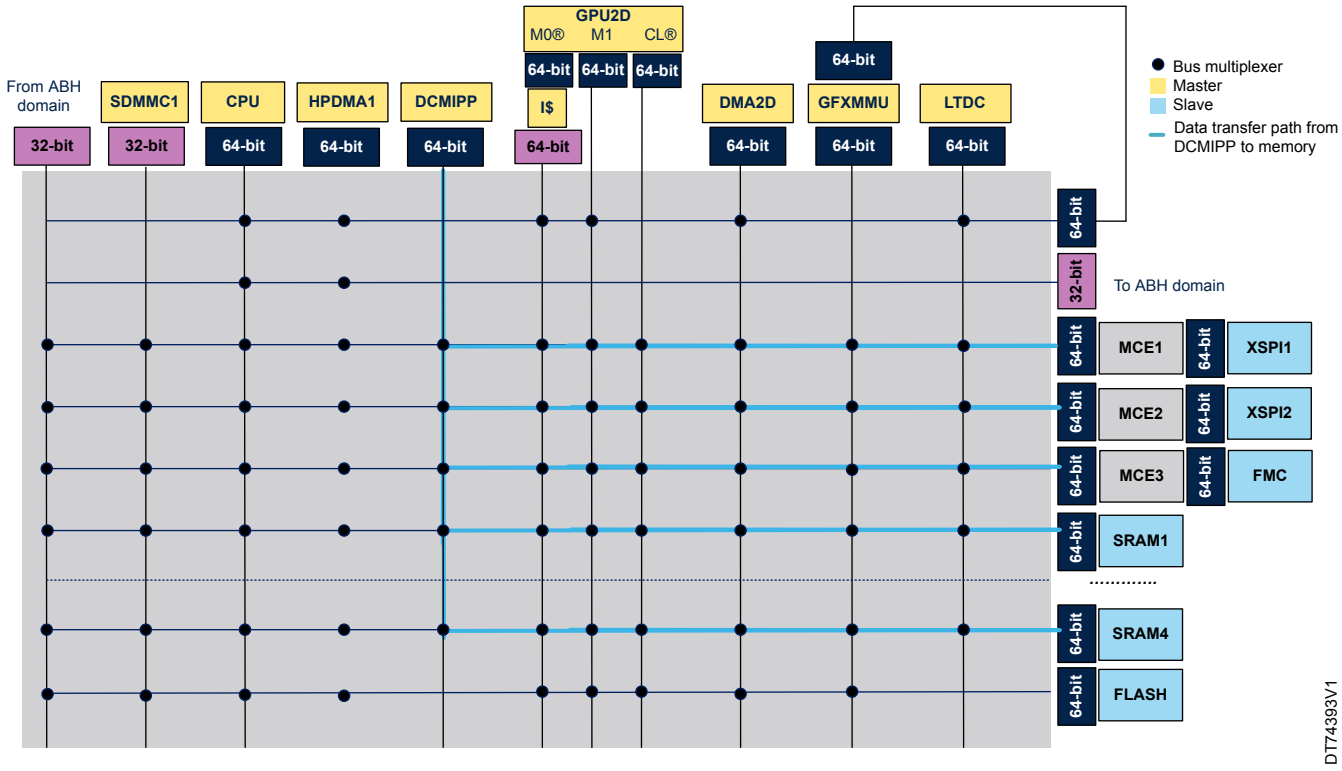
The STM32H7R7/7S7 line is a bootflash-based MCU powered by a Cortex®-M7 core running up to 600 MHz, with 64 Kbytes of user bootflash, 620 Kbytes of flexible SRAM, and 32 x 32 Kbytes of cache with Flex ECC.

The STM32H7R7/7S7 devices embed two bus matrices: one 64-bit AXI matrix and one 32-bit AHB matrix. This configuration allows efficient and simultaneous operation of high-speed peripherals and removes bus congestion when several masters are simultaneously active (different masters located in separate bus matrices). Each bus matrix ensures and arbitrates concurrent accesses from multiple masters to multiple slaves. The arbitration uses a round-robin algorithm. The maximum bus matrix frequency is half the maximum CPU frequency.

The DCMIPP embeds an AXI bus master to drain all the valid data captured into the pipe from the internal FIFO to the targeted memory.

Figure 5 shows the DCMIPP interconnection and the data path in STM32H7R7/7S7 devices.

Figure 5. DCMIPP interconnection and data path on STM32H7R7/7S7



2.5.2 STM32N6x5/6x7 system architecture

The STM32N6x5/6x7 is based on the Arm® Cortex®-M55 running at 800 MHz, the first CPU to introduce Arm Helium vector processing technology, bringing DSP processing capability to a standard CPU.

The STM32N6x5/6x7 is the first STM32 MCU to embed the ST Neural-ART accelerator™, an in-house developed neural processing unit (NPU) engineered for power-efficient edge AI applications. Clocked at 1 GHz and providing up to 600 GOPS, it enables real-time neural network inference for computer vision and audio applications.

A dedicated computer vision pipeline with a MIPI® CSI-2 interface and image signal processing (ISP) ensures compatibility with a wide range of cameras. The STM32N6x5/6x7 also features an H264 hardware encoder and the NeoChrom™ Accelerator for graphics, making it suitable for feature-rich products.

Figure 6 represents the architecture of the system based on the Cortex-M55 processor and the Neural-ART processing unit (NPU). The Cortex-M55 includes 64 KB ITCM, 128 KB DTCM, and 32 KB I-CACHE, and DCACHE (32 KB) memories. The Neural-ART unit is associated with AXI SRAM3, AXI SRAM4, AXI SRAM5, and AXI SRAM6 memories (448 KB each), as well as a cache memory (CACHERAM of 256 KB) via the NPU NIC AXI 64-bit interface.

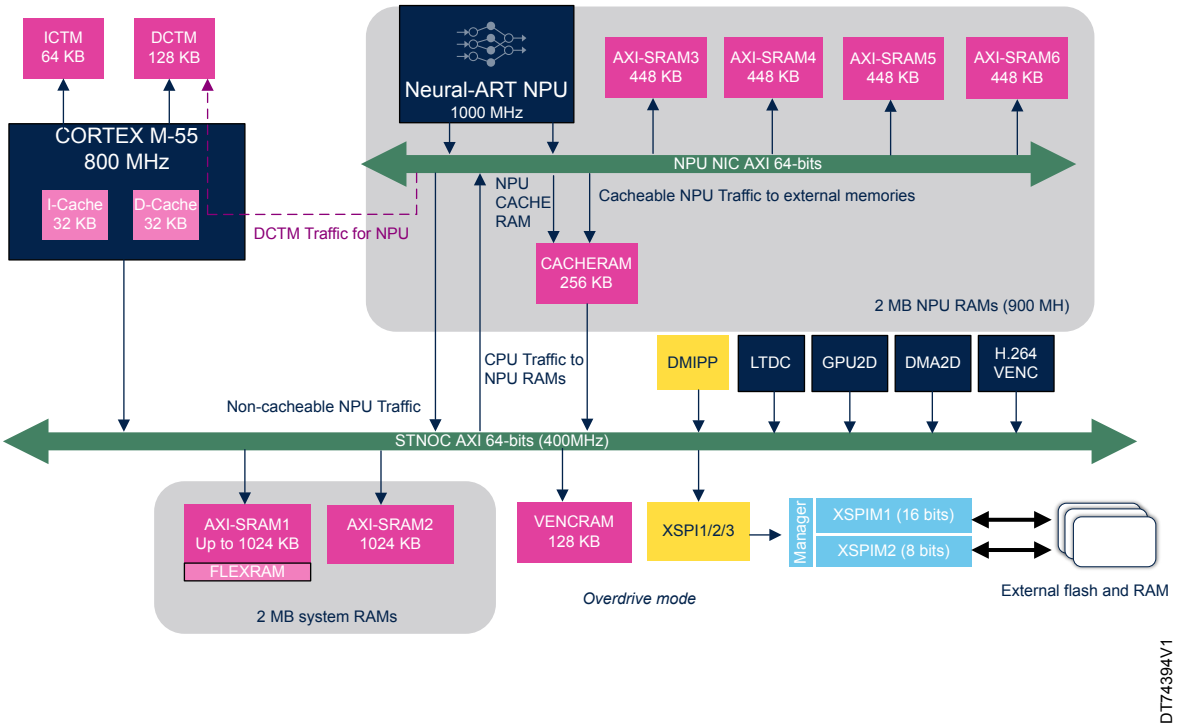
The memory system includes up to 1024 KB AXI SRAM1, 1024 KB AXI SRAM2, 128 KB VENC RAM, XSPIM1 (16-bit), and XSPIM2 (8-bit) interfaces for accessing external flash and RAM memories.

The graphical peripherals are composed of:

- Digital camera interface pixel pipeline (DCMIPP)
- LCD display controller (LTDC)
- 2D graphics accelerator (GPU2D)
- Chrom-ART graphics accelerator (DMA2D)
- H.264 video encoder (VENC)

The STNoC AXI 64-bit bus (400 MHz) manages noncacheable NPU traffic and CPU traffic to NPU memories, enabling advanced processing and memory capabilities for applications such as computer vision, signal processing, and machine learning.

Figure 6. DCMIPP interconnection and data path on STM32N6x7



2.6 DCMIPP functional description

This section details the DCMIPP and how it handles image data and synchronization signals.

2.6.1 Block diagram

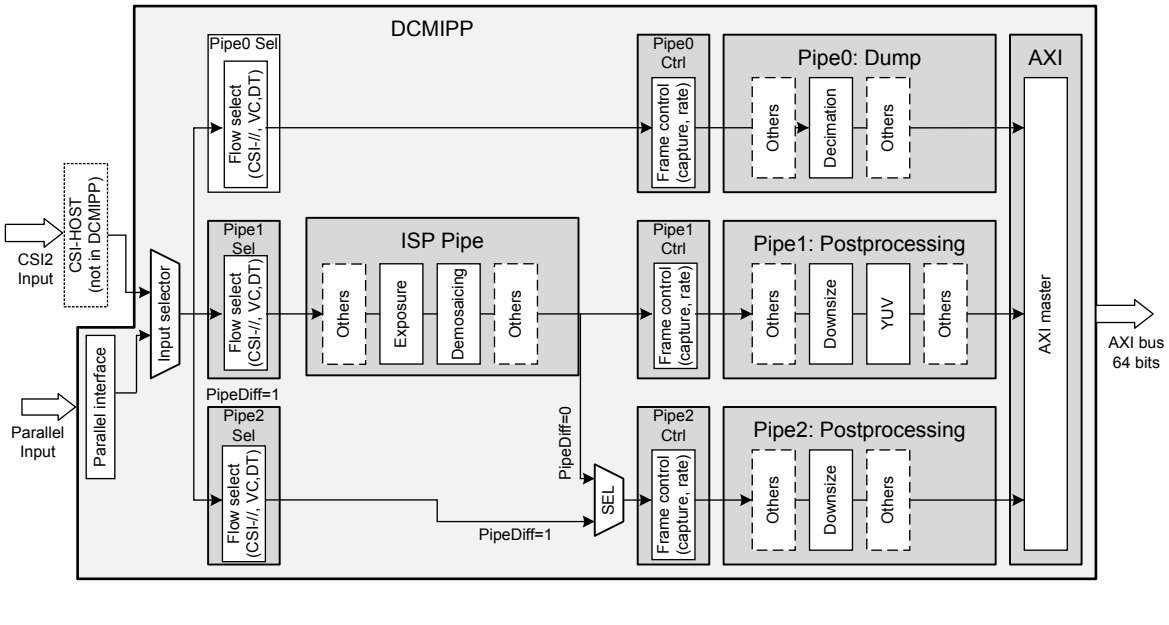
As mentioned in Figure 7, the DCMIPP receives pixels from either a parallel or CSI interface. Once the pixel data enters, it passes through a flow selector (FLOWSEL) that directs the data to the appropriate processing pipelines.

The architecture features three main pipelines for pixel processing:

- **Pipe0-Dump** serves as a side dump pipe for legacy functions (similar to DCMI), statistics, and raw Bayer data for still images. Depending on the product configuration, it includes stages such as data decimation, reformatting, frame control, and image cropping.
- **Pipe1-ISP** is more complex, it converts raw Bayer into full-resolution RGB. It also adjusts exposure and white balance parameters, with stages such as byte conversion, defective pixel correction, demosaicing, color conversion, and contrast adjustment.
- **Pipe1-PostProc** and **Pipe2-PostProc** handle postprocessing for different applications, adapting full-resolution RGB to specific needs such as capture rate, cropping, resizing, and color conversion, with the capability to convert to YUV420 for JPEG or H.264 encoding. Both postprocessing pipes include stages such as frame control, cropping, decimation, downsizing, gamma correction, and pixel packing. The Pipe2-PostProc offers an additional option to sample RGB data before ISP processing. The processed data is then sent to the AXI interface for further use.

*Note: STM32H7R3/7S3 includes only **Pipe0-Dump** pipeline in the DCMIPP architecture.*

Figure 7. DCMIPP architecture

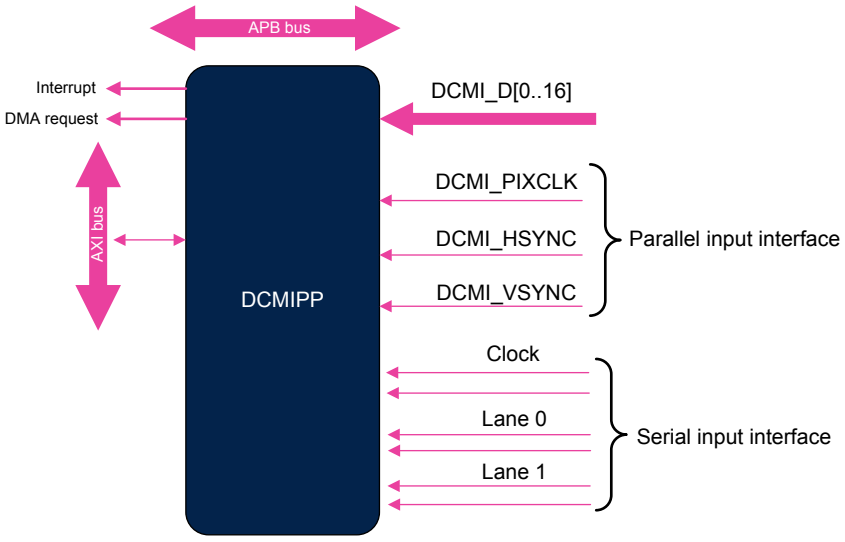


2.6.2 Hardware interfaces

The DCMIPP includes:

- Parallel input interface:
  - Up to 16 data lines (D15-D0)
  - Pixel clock line DCMIPP _PIXCLK
  - DCMIPP _HSYNC line (horizontal synchronization)
  - DCMIPP _VSYNC line (vertical synchronization)
- CSI-2 camera serial interface:
  - Clock lane
  - Data lanes (Lane 0, Lane 1)

Figure 8. DCMIPP signals



2.6.3 Parallel interface

2.6.3.1 *Hardware synchronization mode*

In hardware synchronization mode, the system uses two synchronization signals, DCMIPP_HSYNC and DCMIPP_VSYNC. Depending on the camera module and mode, data can be transmitted during horizontal and vertical synchronization periods. DCMIPP_HSYNC and DCMIPP_VSYNC act as blanking signals, as all data received during DCMIPP_HSYNC and DCMIPP_VSYNC active periods are ignored.

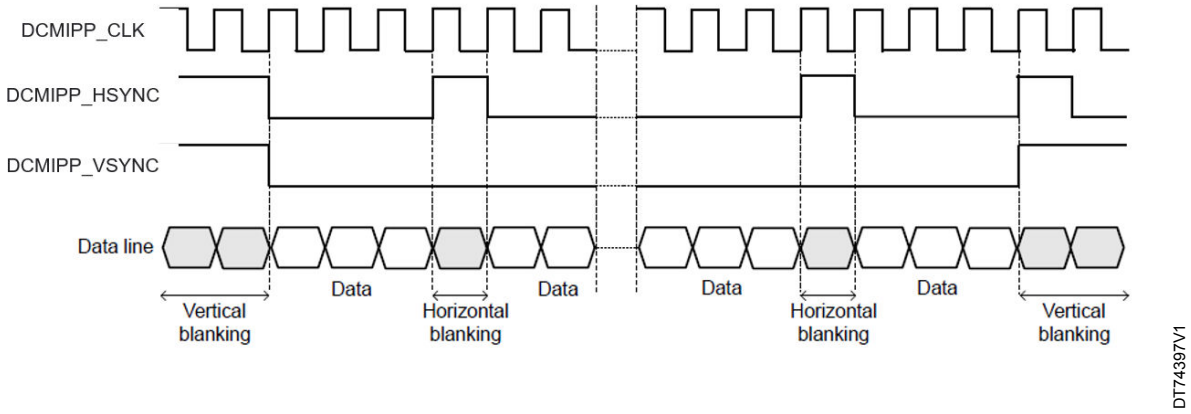
To correctly transfer images in the RAM buffer, data transfer is synchronized with the DCMIPP_HSYNC and DCMIPP_VSYNC signals.

When the hardware synchronization mode is selected and capture is enabled (CPTREQ bit set in DCMIPP_PxFCTCR), data transfer is synchronized with the deassertion of the DCMIPP_VSYNC signal (next start of frame).

Then, the transfer can be continuous, with successive frames transferred by the IP-Plug to either the same buffers, succesive buffers or circular buffers.

Figure 9 illustrates an example of data transfer when DCMIPP_VSYNC and DCMIPP_HSYNC signals are active high, and when the capture edge for DCMIPP_PIXCLK is the rising edge.

Figure 9. Frame structure in hardware synchronization mode



2.6.3.2 *Embedded data synchronization mode*

The embedded data synchronization mode is available only with the parallel interface capture mode. In this mode, the data flow is synchronized using embedded 32-bit codes with the values 0x00 and 0xFF, which are no longer used in the data.

There are four types of codes, all with 0xFF0000XY format. The embedded synchronization codes are supported only in 8-bit parallel data capture (in the DCMIPP_PRCR register, the EDM[2:0] bits must be programmed to “000”). For other data widths, the synchronization mode generates unpredictable results.

Note: Camera modules generate up to eight synchronization codes when in interleaved mode, while the DCMIPP reacts to one single code. Therefore, in an interleaved flow with an embedded synchronization, every other frame is discarded because it is not detected.

Mode 1

This mode is ITU656 compatible (ITU656 is the digital video protocol of ITU-R BT.656).

The following reference codes indicate a set of four events:

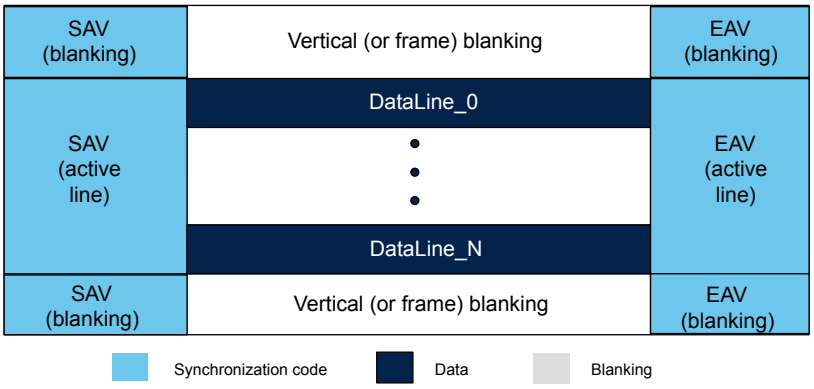
- SAV (active line): line-start
- EAV (active line): line-end
- SAV (blanking): line-start during inter-frame blanking period
- EAV (blanking): line-end during inter-frame blanking period

This mode can be supported by programming the following codes:

- FS ≤ 0xFF
- FE ≤ 0xFF
- LS ≤ SAV (active)
- LE ≤ EAV (active)

Figure 10 illustrates the frame structure using this mode.

Figure 10. Frame structure in embedded synchronization mode 1



DT74398V1

Mode 2

The embedded synchronization codes signal another set of events:

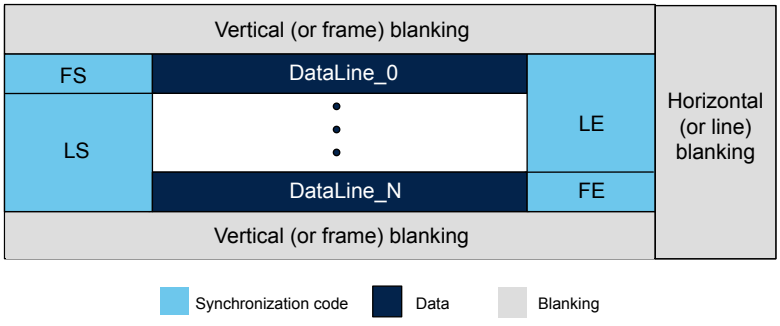
- Frame-start (FS)
- Frame-end (FE)
- Line-start (LS)
- Line-end (LE)

A 0xFF value programmed as a frame-end (FE) means that all the unused codes (the possible values of codes other than FS, LS, LE) are interpreted as valid FE codes.

In this mode, once the camera interface has been enabled, the frame capture starts after the first occurrence of an FE code followed by an FS code.

Figure 11 illustrates the frame structure when using this mode.

Figure 11. Frame structure in embedded synchronization mode 2



DT74399V1

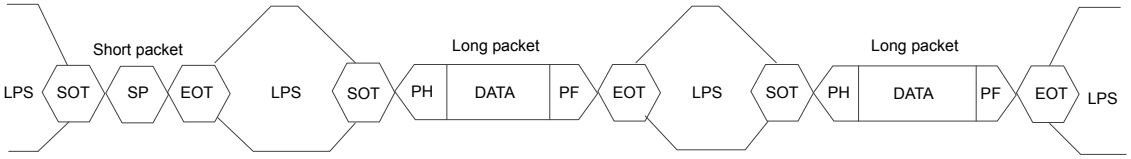
2.6.4 CSI-2 camera serial interface

2.6.4.1 *CSI-2 overview*

The CSI-2 low-level protocol supports the transport of arbitrary data using both short and long packet formats, with packets transmitted over the physical D-PHY interface. The protocol has the following characteristics:

- Transport of arbitrary data
- 8-bit wordsize
- Special packets for frame start, frame end, line start, and line end information
- Descriptor for the type, pixel depth, and format of the application-specific payload data
- 16-bit checksum code for error detection
- 6-bit error correction code (ECC)

Figure 12. Low-level protocol overview



LPS: low-power state, SOT: start of transmission, EOT: end oF transmission, PH: packet header, PF: packet footer

DT53048V1

As illustrated in the figure before, the sequence of the CSI-2 frames begins with an exit from the low-power state (LPS), followed by the start of transmission (SOT) signal, indicating the start of the packet.

Initially, a short packet (SP) is transmitted, followed by the end of transmission (EOT) signal, and a return to low-power state. The sequence then transitions again with SOT to transmit a long packet, which includes a packet header (PH), payload data (DATA), and a packet footer (PF) for error detection, followed by EOT and a return to low-power state. This process repeats for subsequent long packets, ensuring efficient and robust communication.

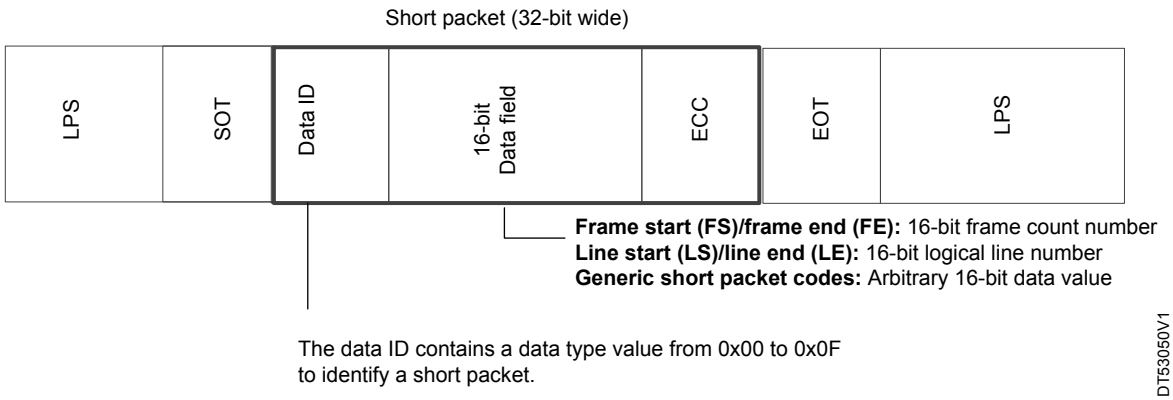
2.6.4.2 *CSI-2 packet description*

The CSI-2 low-level protocol consists of short and long packets, each with a distinct role.

Short packet

The CSI-2 short-packet format for a D-PHY physical layer is shown in the figure below .It is similar to the long packet header, except that the word count field is replaced by a 16-bit data field. The ECC 8-bit code is present to secure headers having potential bit corruption.

Figure 13. CSI-2 short-packet structure based on D-PHY physical layer



DT53050V1

Long packet

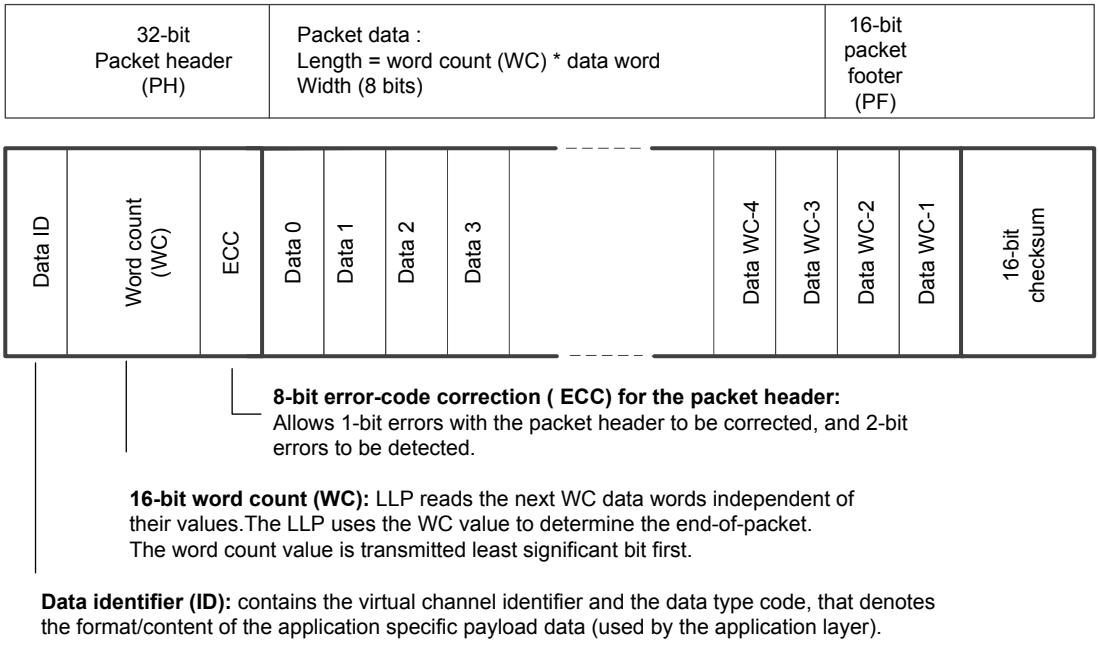
The figure below presents the long packet format. It includes a data ID that consists of a virtual channel number, associated with the data packets, and the corresponding data type.

The word-count field allows the LLP to determine the data-payload width before detecting the packet footer and the EOT. This information is used to detect shorter-than-expected packets, and to trigger a corresponding error

The ECC corrects a single erroneous bit in the packet header in the case of data corruption.

The packet data corresponds to the data payload for a given virtual channel and for an identified data type. To complete the long-packet reception block, a packet footer that contains a 16-bit CRC checksum is received and compared to that which is internally calculated by the LLP, to check the data integrity of the received data payload.

Figure 14. CSI-2 long-packet structure for a D-PHY physical layer



2.6.5 DCMIPP interrupts

The DCMIPP handles the following interrupts for both the parallel interface and all pipes (Pipe 0, Pipe 1, and Pipe 2) :

- Parallel interface interrupts
  - IT_ERR indicates the detection of an error in the embedded synchronization code sequence (only in embedded synchronization mode). This interrupt is indicated by bit 6 in the DCMIPP_PRSR register.
- General pipe interrupts:
  Specific to Pipe 0
  - IT_LIMIT: This interrupt is triggered when the received volume is larger than the maximum allowed dump volume. It indicates unexpected behavior in Pipe 0, which usually does not occur if there are no issues.
  - IT_LINE: This interrupt is for stripe-based operators and is triggered after every 1/2/4/8/16/32/64/128 dumped lines and the last line. It is measured and extracted at the end of the pipe, close to the output to memory.
  - IT_FRAME: This is a secondary interrupt, serving as a backup for fast software. It is triggered after the last data dump of this pipe and is inactive if no capture has occurred. It is typically used to quickly reconfigure non-shadow registers (during only vertical blanking) and to trigger the usage of the previously captured frame.
  - IT_OVR: This interrupt is triggered when there is a data overflow, meaning the memory was too slow compared to the received pixels. This indicates unexpected behavior in Pipe0, which usually does not occur if there are no issues. The recommendation is to skip the impacted frame.
  - IT_VSYNC: This is the main interrupt, where most software can sit. It is triggered at VSync (mid blanking) and is permanent, even if no dump is active. It is typically used to slowly reconfigure the pipes (at least shadow registers) for the next frame and to trigger the usage of the previously captured frame.

Each pipe has a dedicated interrupt local register that controls the fourth interrupt event (OVR, VSYNC, FRAME, LINE), as indicated in the DCMIPP_P0SR, DCMIPP_P1SR and DCMIPP_P2SR registers for Pipe0, Pipe1, and Pipe2.

2.6.6 DCMIPP events

The DCMIPP contains four events to alert the software to react to certain application situations. These events are linked to the pipeline. The table below mentions these events:

Table 3. DCMIPP events

Pipe	Event
Pipe0	HSYNC
	VSYNC
	Frame
	Line
Pipe1	HSYNC
	VSYNC
	Frame
	Line
Pipe2	HSYNC
	VSYNC
	Frame
	Line

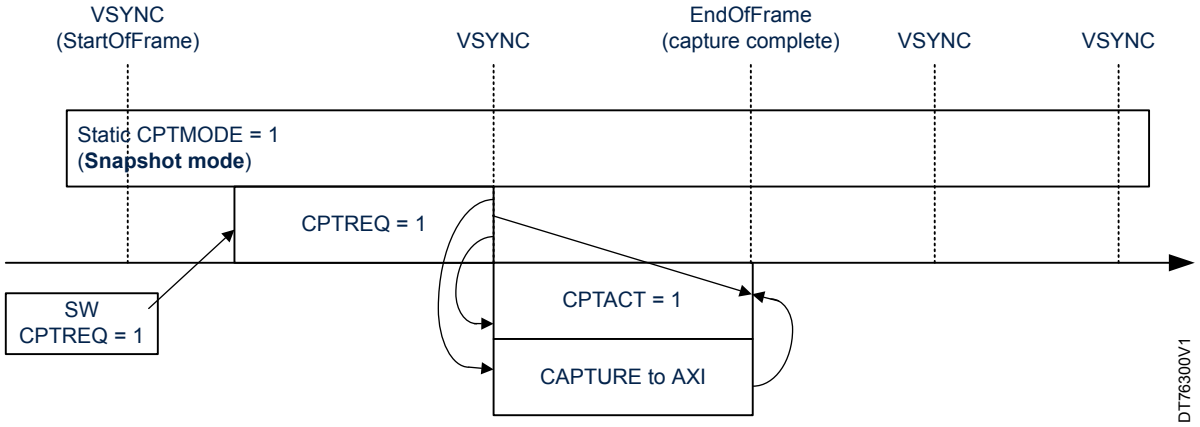
2.6.7 Capture modes

The DCMIPP captures only the frames required by the application optimizing memory access, AXI bus occupation and power consumption. The applicative software may request to get continuous frames from the camera sensor, or only frame in snapshot mode. These two different modes are selected thanks to the bit CPTMODE inside the DCMIPP_PXFSCR register.

2.6.7.1 *Snapshot mode*

In snapshot mode, the software triggers the bit CPTREQ to launch the acquisition from the next start of frame for a single frame only. The bit CPACT remains high only for one frame. It is deasserted at the end of the active frame. At the same time, the bit CPTREQ is cleared by hardware to be ready for future requests, other snapshot acquisitions, or even continuous acquisition after modifying CPTMODE bit in such case.

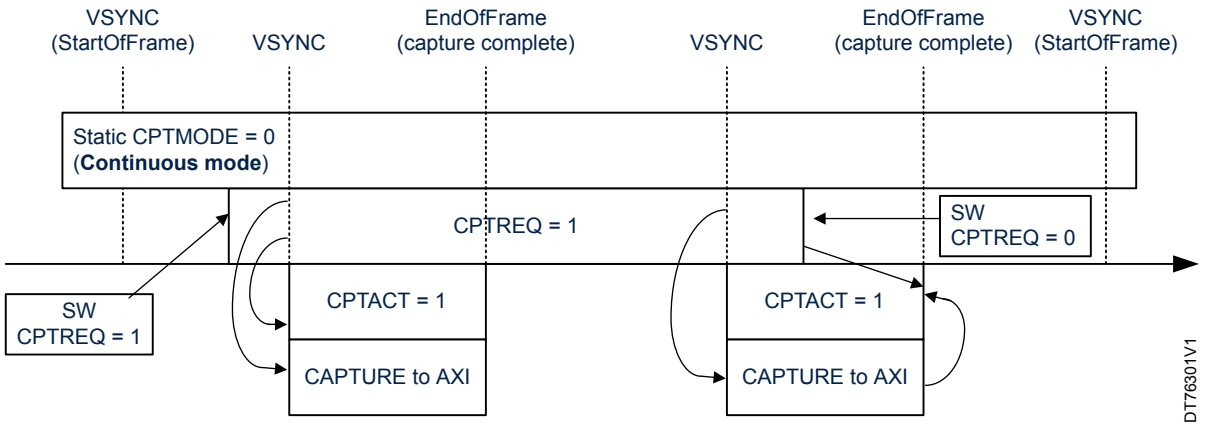
Figure 15. Snapshot capture mode



2.6.7.2 *Continuous mode*

In continuous mode, the DCMIPP processes the frames continuously. The software needs to arm the bit CPTREQ = 1, and the next VSYNC triggers the data acquisition until the deassertion of this bit. The bit CPTACT remains high, indicating this continuous mode operation.

Figure 16. Continuous capture mode



2.7 DCMIPP pixel format support

This section describes the pixel formats used in the parallel interface, pixel pipes, and dump pipe.

The DCMIPP supports the following data formats:

- Monochrome (8, 10, 12, 14 bit)
- RGB444, RGB565, RGB666, RGB888

- Raw Bayer (8, 10, 12, 14 bit)
- YUV422 and YUV444

Note: The parallel interface does not input specifically RGB444, RGB555 and RGB666. However, a sensor with this output can connect them onto the DCMIPP, by selecting RGB565 and RGB888. The missing bits can then be connected or strapped to the MSB of the sensor output.

Wide input pixel formats can be sampled in single, double, triple, or quadruple cycles. The table below describes the input pixel formats for the parallel interface and the number of cycles of each format.

Table 4. Supported pixel formats on the parallel interface

I/O pin	1 cycle	2 cycles	3 cycles	4 cycles
Byte	X	-	-	-
Raw 8-bit	X	-	-	-
Raw 10-bit	X	-	-	-
Raw 12-bit	X	-	-	-
Raw 14-bit	X	-	-	-
Monochrome 8-bit	X	-	-	-
Monochrome 10-bit	X	-	-	-
Monochrome 12-bit	X	-	-	-
Monochrome 14-bit	X	-	-	-
Sensor RGB444 to DCMIPP RGB565	X	-	-	-
Sensor RGB555 to DCMIPP RGB565	X	-	-	-
RGB565 – 8-bit	-	X	-	-
RGB565 – 16-bit	X	-	-	-
RGB444 – 12-bit	-	X	-	-
Sensor RGB555 to DCMIPP-RGB565	X	-	-	-
Sensor RGB444 to DCMIPP-RGB565	X	-	-	-
RGB888 - 24-bit	-	-	X	-
YUV444 - 24-bit	-	-	X	-
Sensor RGB666 to RGB888	-	-	X	-
YUV422 - 16-bit (YUYV)	-	X	-	-

2.7.1 Monochrome pixel format

The DCMIPP supports monochrome formats with bit depths of 8, 10, 12, and 14 bits per pixel (bpp), which are used to represent different shades of gray. For example, an 8-bit monochrome image can display 256 shades of gray.

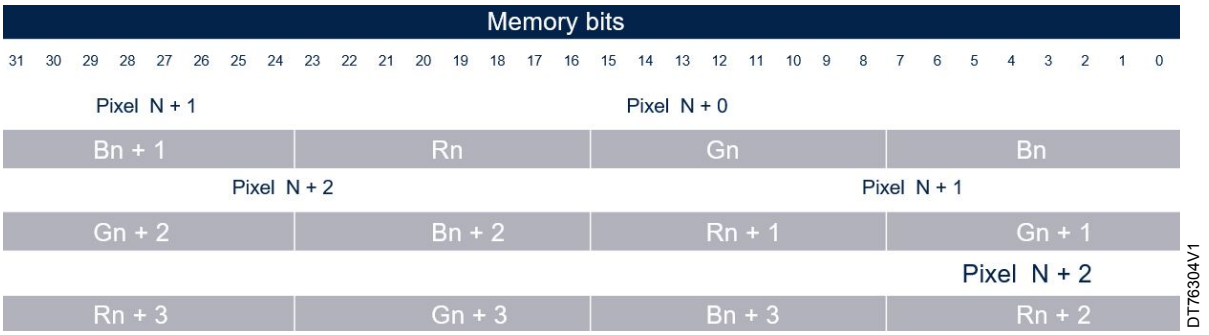
2.7.2 Raw Bayer pixel formats

The DCMIPP supports raw Bayer formats with bit depths of 8, 10, 12, and 14 bits per pixel (bpp). The raw Bayer format captures raw image data directly from the camera sensor without any color processing. The sensor pixels are arranged in a Bayer pattern, consisting of alternating rows of red, green, and blue filters.

Each pixel in the raw Bayer format represents only one color component (red, green, or blue), and the full-color image is reconstructed through a process called demosaicing.

The memory bits have the structure illustrated in Figure 17.

Figure 17. DCMIPP data register filled with monochrome/raw Bayer data



2.7.3 RGB pixel formats

RGB refers to red, green, and blue, which represent the three hues of light. Any color is obtained by mixing these three colors.

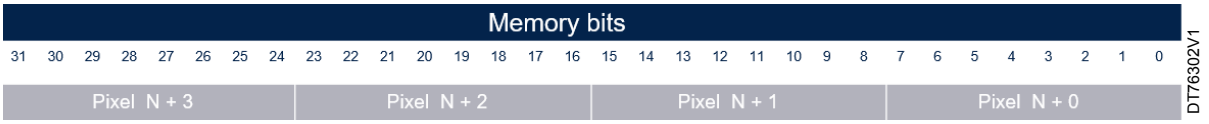
The different RGB formats (RGB444, RGB565, RGB666, RGB888) indicate the number of bits allocated to each color channel.

As an example, RGB888 is used to indicate that each pixel consists of 24 bits, divided as follows:

- 8 bits for encoding the red value (the most significant 5 bits)
- 8 bits for encoding the green value
- 8 bits for encoding the blue value (the less significant 5 bits)

Each component has the same spatial resolution. Each sample has a red (R), a green (G) and a blue (B) component. Figure 18 illustrate the memory bits containing RGB data, when an 8-bit data width is selected.

Figure 18. DCMIPP data register filled with RGB data



2.7.4 YUV pixel formats

YUV is a family of color spaces that separates the luminance or luma (brightness) from the chrominance or chroma (color differences).

YUV consists of three components:

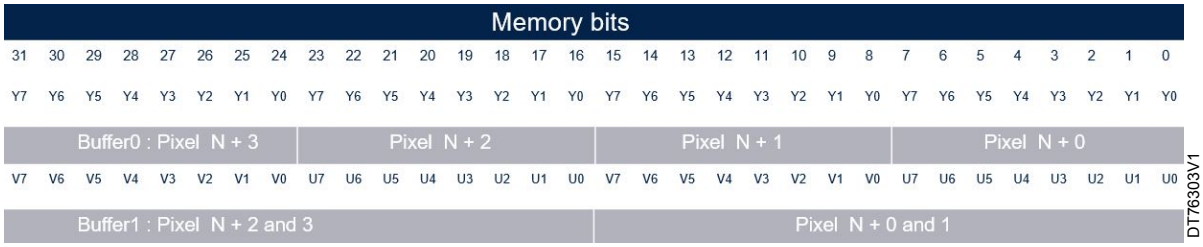
- Y refers to the luminance or luma (black and white)
- U refers to the blue chroma
- V refers to the red chroma

For YUV422-1 buffer (YUYV), each pixel is stored with 8 bits for Y (luminance), and the chrominance components U and V are shared between two pixels. Specifically, the memory layout alternates between Y, U, Y, V for consecutive pixels (that is, pixel N has Y0, U0, Y1, V0, and pixel N+1 has Y2, U1, Y3, V1).

For example, in the YUV422-2 buffers configuration, the Y components are stored in one buffer and the U and V components are stored in another buffer. For instance, the buffer contains Y values for pixel N, N+1, N+2, and N+3, while buffer contains the U and V values for pixel N and N+1.

Figure 19 illustrates the memory bits containing YUV422-1 data when an 8-bit data width is selected.

Figure 19. DCMI data register filled with YUV data



2.8 Supported pixel formats on the CSI-2 interface

This section describes the pixel formats used in CSI-2 interface, pixel pipes, and dump pipe.

Table 5 lists the supported pixel formats in the input interfaces (parallel, CSI), as input to the pixel Pipe1 and Pipe2, and in the three output pipes.

Table 5. Supported pixel formats on the CSI-2 interface

Pixel format	BPP	Parallel input (clk/pix)(1)	CSI input	Pipe0 output (dump)	Pipe1 and Pipe2 input	Pipe1 output (main)	Pipe2 output (ancillary)
Bytes (CSI: long packets and user)	8	-	Y	Y	-	-	-
Byte stream (JPEG, compressed video)	8	Y, 1	-	Y	-	-	-
Other data	8 to 16(2)	Y, 1		Y	-	-	-
Raw6	6	(3)	Y(4)	Y(4)	-	-	-
Raw7	7	(3)	Y(4)	(4)	Y	-	-
Raw8	8	Y, 1	Y	Y	To 8 bpc	Y	Y
Raw10	10	Y, 1	Y	To 16 bpc	To 8 bpc	-	-
Raw12	12	Y, 1	Y	To 16 bpc	To 8 bpc	-	-
Raw14	14	Y,1	Y	To 16 bpc	Y	-	-
Mono8	8	Y, 1	-	Y	To 8 bpc	Y	Y
Mono10	10	Y,1	-	To 16 bpc	To 8 bpc	-	-
Mono12	12	Y, 1	-	To 16 bpc	To 8 bpc	-	-
Mono14	14	Y,1	-	To 16 bpc	To 8 bpc	-	-
RGB444	12	(3)	Y	(3)	To 8 bpc	-	-
RGB555	15	(3)	Y	(3)	To 8 bpc	-	-
RGB565	16	Y, 1, 2	Y	Y	To 8 bpc	Y	Y
RGB666	18	(3)	Y	(3)	To 8 bpc	-	-
RGB888 / YUV444	24	Y, 2, 3	Y	Y	Y	Y	Y
xRGB888 / xYUV444	32	-	-	Y	-	Y	Y
RGBx888 / YUVx444	32	-	-	Y	-	Y	Y
YUV422-1 (YUYV)	16	Y, 1, 2	Y	Y	Y	Y	Y
YUV422-2	16	-	Y(4)	Y(4)	Y	Y	-

Pixel format	BPP	Parallel input (clk/pix)(1)	CSI input	Pipe0 output (dump)	Pipe1 and Pipe2 input	Pipe1 output (main)	Pipe2 output (ancillary)
YUV420-2 (NV21/ NV12)	12	-	Y(4)	Y(4)	Y	Y	-
YUV420-3 (YV12)	12	-	Y(4)	Y(4)	Y	Y	-
YUV422 10 bpc20	20	-	Y(4)	Y(4)	-	-	-
YUV420 10 bpc	15	-	Y(4)	Y(4)	-	-	-
Any other CSI-2	Any	-	Y(4)	Y(4)	-	-	-

1. *Index is used to link the input pixel format to the DCMIPP_PRCR register. The index has a meaning only for camera connected with the parallel interface .*
2. *Pixel formats on 16 bpp (like RGB565 or YUV422-1) can be received either on an 8-bit interface (using 2 cycles), or on a 16-bit input interface (using one cycle). Pixel formats on 24 bpp (like RGB888) can be received either on an 8-bit interface (using three cycles), or on a 12-bit input interface (using two cycles).*
3. *On the parallel interface input, some sensor pixel formats are supported by the DCMIPP, by selecting a wider pixel format in DCMIPP, by mapping the sensor wires onto the MSB of the DCMIPP interface. On the missing input pins, it is recommended to replicate the sensor MSB pins onto the missing input LSB pins. This helps achieving full dynamic on the extended pixel format.*

   The workaround can be applied on the following formats (sensor output on left, mapping on DCMIPP format on right):

   - *Raw6: Raw8 with 1-cycle input, processed as an 8 bpp*
   - *Raw7: Raw8 with 1-cycle input, processed as an 8 bpp*
   - *RGB444: RGB565 with 1-cycle input, processed as a 16 bpp*
   - *RGB555: RGB565 with 1-cycle input, processed as a 16 bpp*
   - *RGB666: RGB888 with 3-cycle input, processed as a 24 bpp*
4. *The same bit mapping is used for the input RGB888 and YUV444, the input xRGB and xYUV, and similarly for the outputs. The difference between RGB and YUV is the color reference, set in the color conversion that occurs after the input or before the output.*

2.9 AXI IP-Plug

The AXI IP-Plug is designed to manage data transfer from various pipes to memory using the AXI protocol. This module, shared by all pipes ensures efficient handling of data dumps from these pipes to memory.

The AXI IP-Plug is equipped with key features designed to optimize data transfer and memory management.

It includes Pipe ID differentiation to identify data sources and manage quality of service (QoS) accordingly. The FIFO buffers are sized to handle up to 2 microseconds of latency. It supports burst lengths of 64 and 128 bytes. The IP-Plug can manage up to 16 outstanding transactions simultaneously, addressing high-bandwidth requirements effectively. A bandwidth limiter is included to prevent memory overload by controlling the bandwidth usage of each pipe. Additionally, the unique AXI-QOS mechanism ensures that all pipes meet their real-time constraints without differentiation.

2.9.1 IP-Plug configuration steps

The following steps outline the programming procedure for accessing idle registers.

1. **Lock request**
   The CPU requests to lock the IP-Plug by writing 1 in the P_START register flag.
2. **Status polling**
   The CPU waits that the IDLE register flag is set to 1 by the IP-Plug.

3. **Programming**
   The CPU writes all the required IP-Plug registers. This includes:
   - Setting the MEMORYPAGE register for efficient DRAM handling.
   - Identifying clients corresponding to each pipe or buffer.
   - Configuring OTR (outstanding transactions) for each client.
   - Setting FIFO size by configuring DPREGSTART and DPREGEND for each client based on their peak bandwidth requirements.
   - Setting TRAFFIC (burst size) for each client.
   - Allocating WLRU (bandwidth proportion) for each client.
   - Setting SVC (system virtual channel) for each client.
4. **Unlock**

The CPU unlocks the IP-Plug by writing 0 in the P_START register flag.

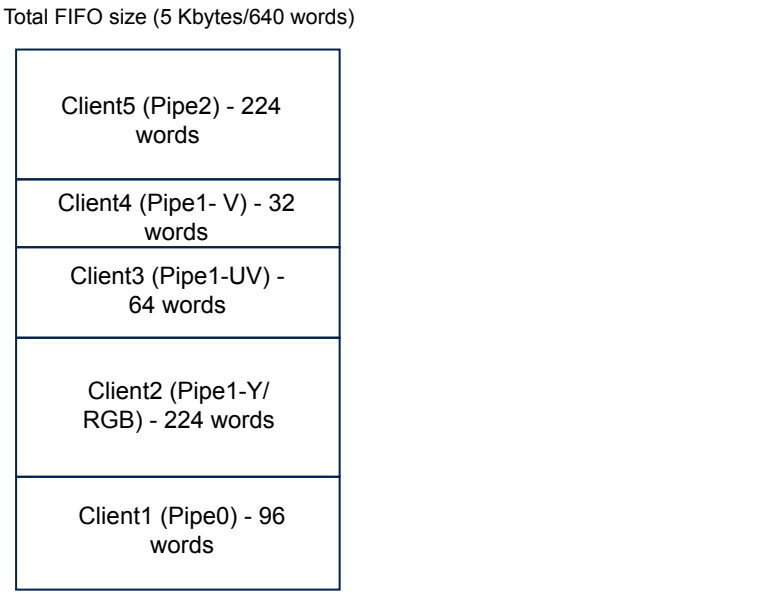
In the STM32N6x5/6x7, the DCMIPP includes three pipes (Pipe0, Pipe1, Pipe2), with Pipe1 supporting full planar capability through three buffers (Y, U, V).

Each client of the IP-Plug is allocated a proportion of the total FIFO size and the total outstanding available transactions based on their bandwidth requirements. The total FIFO size is 5 Kbytes (5120 bytes), which translates to 640 words (since AXI width is 64 bits). The allocation of the FIFO size among the clients is illustrated in the diagram below. The parameters used for allocation are DPREGSTART, which is the starting word of the FIFO for the client (for example, Client1 starts at word 0), and DPREGEND, which is the ending word of the FIFO for the client (that is, Client1 ends at word 95). The total number of words for each client is calculated based on their allocated proportion of the total FIFO size.

The capabilities of the DCMIPP are shared among the clients The total bandwidth proportion, when all pipes are used, is 12 with the following assumptions:

- Client1 for Pipe0: dump of raw Bayer pixels, padded to 16 bpp, BW = 2 / 12
- Client2 for Pipe1-Y: dump of ARGB pixels, on 32 bpp, BW = 4 / 12
- Client3 for Pipe1-UV: dump of UV components, on 8 bpp, BW = 1 / 12
- Client4 for Pipe1-V: dump of V component, on 4 bpp, BW = 1 / 12
- Client5 for Pipe2: dump of ARGB pixels, on 32 bpp, BW = 4 / 12

Figure 20. FIFO allocation among IP-Plug clients for STM32N6x5/6x7



Note: If a client is assigned 1 KB of FIFO space, which corresponds to 128 words, and DPREGSTART is 0, then DPREGEND would be 127. This is calculated as follows:

FIFO size in words = 1 KB / 8 bytes per word = 128 words

DPREGEND = DPREGSTART + (FIFO size in words - 1)

3 DCMIPP pipes description

This section describes the functional pipes of DCMIPP. For each element, it details the camera pipeline and the features used for pixel data processing.

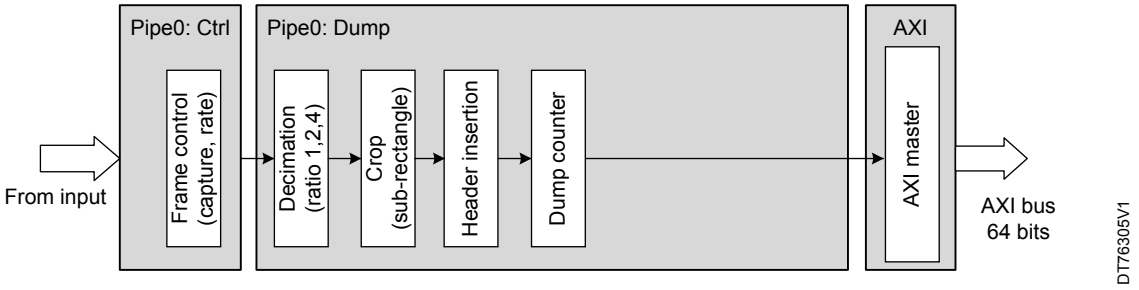
3.1 Pipe0: Dump pipe

3.1.1 Overview

Pipe0 acts as a dump pipe, extracting data from the camera sensor module and transferring it directly to the targeted memory with basic decimation and 2D cropping operations. It retrieves data via the 16-bit parallel interface or the CSI-2 interface, managed by the INSEL bit in the DCMIPP_CMCR register. The flow selection determines the virtual channels and data types to be processed, while the frame controller handles the camera acquisition mode.

When valid input data is detected, 2D cropping and decimation can be configured by software. Dump counter and data limit ensure proper handling of data length within a frame. Pipe0 can be reconfigured periodically, with events triggering software reconfiguration routines. Shadow registers can be written during frame processing, with values loaded into physical registers at the start of the frame event.

Figure 21. Data flow and processing modules in Pipe0

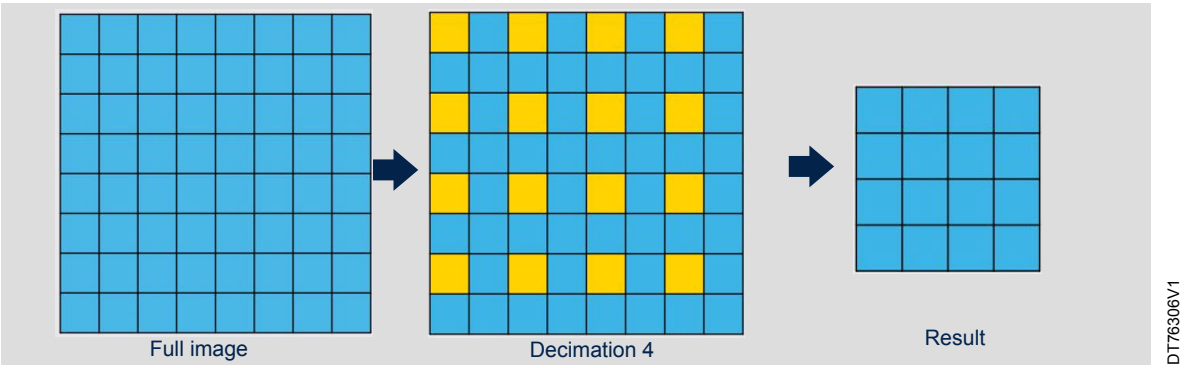


3.1.2 Decimation

Decimation is used to reduce image resolution while maintaining the field of view. A typical application involves using a lower resolution image for live view to reduce camera bandwidth and increase the frame rate. This also helps in providing smaller images for neural network processing. It is possible to downsize a frame by a factor 2x, 4x and 8x in both horizontal and vertical directions.

The figure below illustrates an example of an image with a resolution of 4096 x 2048. In 4 x 4 decimation mode, the output resolution is 1024 x 512.

Figure 22. Decimation module



3.1.3 Pixel 2D cropping

The crop operation extracts and forwards a rectangle of pixels. The dump pipe (Pipe0) can handle 2D crop processing. Vertical cropping is based on lines, facilitated by the HSYNC event.

However, the horizontal cropping is applied based on 32-bit wide data rather than individual pixels, as this pipe does not directly consider pixels. Instead, it manages the data flow with 32-bit granularity.

3.1.4 Downsize

The downsize module implements a downsize based on box filtering, which allows ratios from 1x to 8x independently, in both X and Y directions.

3.1.5 Header insertion

The header insertion module is designed works with dump pipes and can insert a header before the data of a former CSI packet. This header can either be a frame sync short packet or a header of a long data packet. It helps software interpret the following data payload:

- The frame sync short packet provides information about the frame type (SOF or EOF), frame number, and virtual channel ID.
- The header of the data long packet provides information about the number of bytes in the payload, data type ID, and virtual channel ID.

3.1.6 Dump counter

The DCMIPP offers the capability to count the number of 8-bit data dumped within a frame for input data format with unknown length (JPEG for instance). It avoids dumping greater number of data which would compromise memory integrity if the reserved area for the dump is smaller than the number of data under capture.

The limit is set by the software in the DCMIPP_P0DCLMTR register. When this limit is reached, an interrupt may be generated to notify the software that any data beyond the limit is rejected, resulting in an incomplete frame acquisition. The counter continues to increment after the limit is crossed to indicate the amount of data that has been discarded, and it saturates at 0x03FF_FFFF.

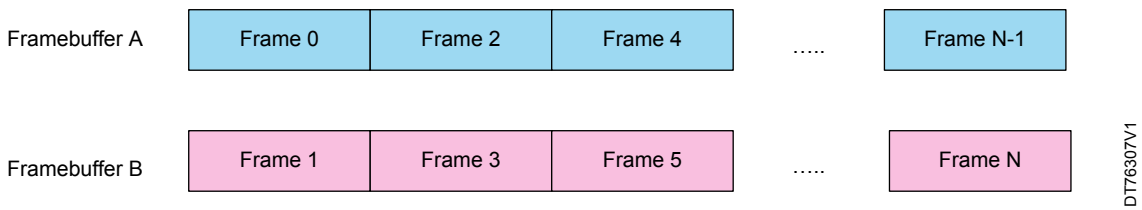
3.1.7 Double buffer mode

The dump pipe uses an AXI master interface to dump the data from the internal FIFO to the external memory. In the application, it is possible to handle frame data swapping memory area frame by frame as illustrated in Figure 23. The double buffer mode supports this function.

There are two memory address registers set to initialize the base addresses of these memories areas. Each start of captured frame event swaps the memory base address to handle double buffering mode.

Double buffer mode allosw postprocessing on a buffer (framebuffer) while the other buffer is read for display (display buffer) or used as input in the NPU.

Figure 23. Double buffer example

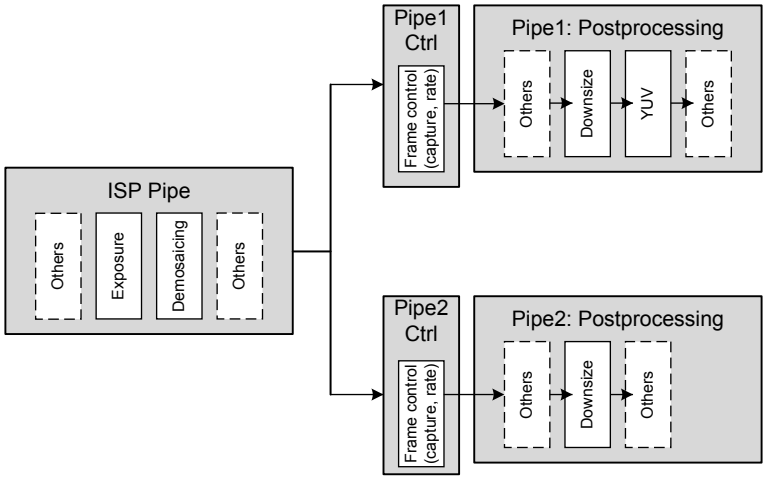


3.2 Pipe1: Main pipe, ISP part

Pipe1 is a pixel pipeline that integrates various image signal processing (ISP) and postprocessing functions to relieve the software from specific time and memory-consuming operations. Pipe1 is typically used for primary tasks such as video encoding, software analysis, and neural networks. It allows for input downsizing, color conversion, and dumping to multiple pixel formats, including multi-planar ones.

Pipe1 can adjust luminosity through gamma conversion, perform autoexposure, and calibrate black levels using the embedded image-processing functions. This capability is particularly useful for handling camera sensors without internal ISPs. Additionally, Pipe1 can convert raw Bayer data to RGB (demosaicing). Importantly, Pipe1 shares all its image-processing functions with Pipe2.

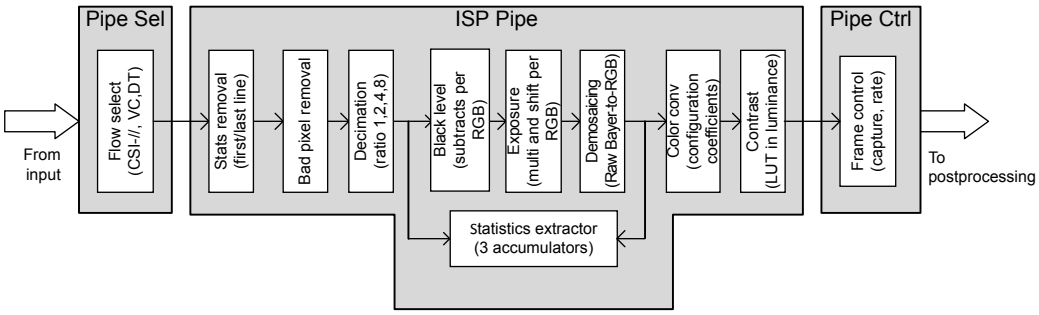
Figure 24. Data flow and processing modules in ISP, Pipe1, and Pipe2



3.2.1 Pipe1 (ISP part)

This section details each module in the ISP part, which implements various functions as illustrated in Figure 25. These functions include: statistics removal within the data flow, bad pixel detection to automatically remove noisy pixels, and basic decimation to reduce high-resolution camera sensor data to fit within the image-processing maximum resolution tolerance. Additionally, the ISP performs auto exposure and statistics extraction based on pixel analysis. It employs a demosaicing algorithm to convert raw Bayer input pixels to RGB, and includes color conversion and contrast enhancement functions.

Figure 25. Data flow and processing modules in ISP Pipe



3.2.1.1 *Byte to pixel conversion*

The module operates on pixel pipes by receiving 32-bit words from the input interfaces (parallel or CSI). It extracts pixels based on the provided pixel format and maps them into the triple-component pixel pipeline, which is 14 bits per component (bpc) at this stage and 8 bpc later downstream.

Note: YUV420 to YUV444 conversion is not supported. Pipe1 and Pipe2 cannot extract and process YUV420 pixels received from the CSI interface.

3.2.1.2 *Statistics removal*

The module is used to remove potential statistic data inserted by a sensor, prior to starting the demosaicing conversion that is multipixel and would be impacted by remaining statistics. It is typically used by sensors with a parallel interface that cannot distinguish between statistics and pixel data. CSI-2 sensors typically split these flows using different virtual channel IDs.

Features:

- Suppression of some of the first lines of the frame
- Suppression of the last lines of the frame, after a configured line N

3.2.1.3 *Bad pixel removal*

This module aims at detecting and correcting artifacts generated by a bad pixel on the sensor array. It is based on a correlation extraction, and not on fixed locations.

3.2.1.4 *Input decimation*

The decimation is redundant with the downsize, that brings a higher quality (the decimation discards the data of the dropped pixels, while the downsize averages all pixels and does not lose information).

The decimation has two main usages:

1. Reduces the width of very high-resolution sensors to one accepted by the raw Bayer to RGB conversion. Indeed, the raw Bayer to RGB embeds a line buffer that limits the processed images to a maximum width of 2688 pixels. For instance, to plug a 12 megapixel sensor (4000 x 3000), decimate its width by a factor 2x to 2000 pixels wide, and have it converted to RGB.
2. Complements the downsize (limited to ratio of 8 x 8), to have a combined downsizing ratio of 64 x 64. It is recommended to have the largest ratio implemented in the downsize, and the decimation complement to that. For instance, for a ratio 30x, it is recommended to have a ratio 4x in decimation, and a ratio 30 / 4 = 7.5 in the downsize.

3.2.1.5 *Statistics extraction*

This module extracts frame statistics to provide inputs for software image compensation algorithms. These algorithms then adjust hardware modules such as black level calibration, exposure compensation, white balance, and contrast enhancement based on the feedback.

Statistics are accumulated during a full frame and sampled at the end, allowing the software the entire next frame to read the accumulated data.

The module features three accumulators, enabling the extraction of three pieces of information per frame. It can focus on specific subscenes by extracting statistics from a designated subrectangle and can select specific components, such as R, G, B, or luminance.

Two types of statistics can be extracted: the average and the spectrum, which is split into 12 bins across the range of 0 to 255.

Note: One limitation is that constructing a 12-bin spectrum with three data points per frame requires four frames, which reduces reactivity

3.2.1.6 *Black level calibration*

This module suppresses a potential positive black level offset, which happens when the sensor sends non-null pixels when capturing a black picture.

3.2.1.7 *Exposure and white balance*

This module aims to compensate for bad exposure, whether too short and dark or too long and light, and corrects color balance to achieve a more neutral white.

The exposure and white balance operations are purely linear, ensuring no color distortion if performed correctly.

The module reduces the pixel dynamic range from 14 bits to 8 bits, lowering the implementation cost of downstream modules, which operate at 8 bits instead of 14.

It also recenters the RGB values towards a neutral white color scene. The module features linear amplification applied to the input component, which is then saturated to 8 bits, minimizing color distortions among R, G, and B.

It selects an 8-bit dynamic range from the 14-bit input: the eight most significant bits (MSB) for well-lit scenes, the eight least significant bits (LSB) for dark scenes, or any intermediate range for medium lighting.

The amplification is configurable via software to adapt to the scene, utilizing statistical extractions to compute the ideal amplification coefficient. Additionally, it differentiates between R, G, and B to recenter the colors to white.

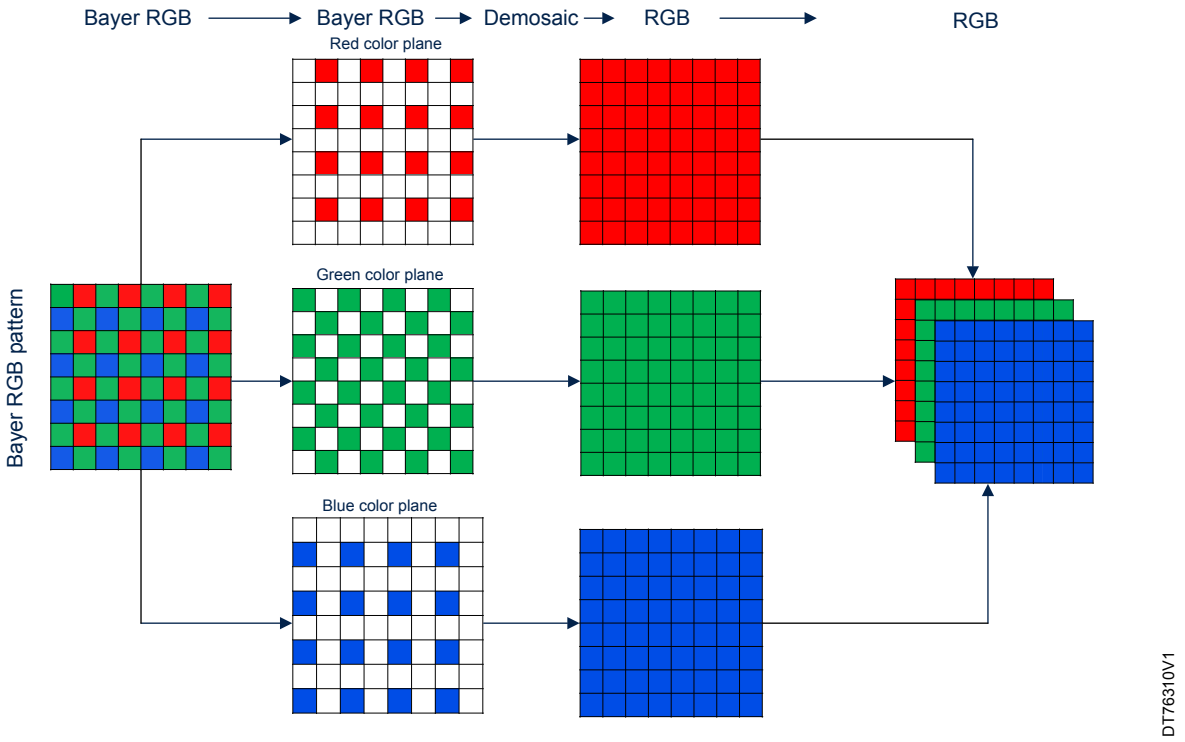
3.2.1.8 *Demosaicing*

The module works on the pixel pipes and implements a mid-quality conversion from raw Bayer 8 bpp to an RGB 24 bpp, with no implicit half-sizing: the output RGB pixel rate is the same as the input component rate.

The conversion targets real-time but medium quality purposes, such as display or low-resolution software analysis. For higher-quality conversions, it is recommended to convert the raw Bayer in software, with higher-quality algorithms but limited to non-real-time, thus still-picture applications. The algorithm takes the surrounding 3 x 3 components in input, applies linear heuristics to determine the most likely missing components, and generates an output RGB888 pixel as illustrated in Figure 26.

To provide the 3 x 3 input matrix to the algorithm, the demosaicing module embeds a double line-buffer, 2 x 2688 components wide. As such, it limits the maximum width of a converted demosaicing buffer to 2688 components per pixels wide. Wider camera sensor resolution can be connected and goes through the first decimation stage in front of the demosaicing operation.

Figure 26. Demosaicing module



Note: This demosaicing conversion algorithm can create some artifacts such as zipper. These artifacts are reduced by a later downsizing or even suppressed with a 2 x 2 downsizing.

3.2.1.9 Contrast enhancement

This module concludes the image-processing pipeline by enhancing the frame contrast through nonlinear emphasis of pixel luminance. It increases the number of luminance quantification steps in frequently used luminance ranges (typically central grays) while reducing steps in less used ranges (usually very dark and very light pixels). The purpose of this approach is to address parts of the scene that may be too dark or too light.

The contrast module nonlinearly amplifies the luminance of RGB pixels to improve visibility. It extracts pixel luminance based on local R, G, and B values and applies an amplification factor dependent on the luminance, interpolated from a table with nine entries. This process amplifies or reduces per-pixel luminance and proportionally adjusts the R, G, and B values.

3.2.1.10 Color conversion

This module implements color conversion with flexible and fully programmable coefficients. The goal of this module is to perform color correction to compensate for ambient illumination (such as sunlight, incandescent, or fluorescent lighting) and sensor distortions. It also maps colors to a neutral RGB.

It also supports YUV sensors by converting YUV data to standard RGB.

The module features the multiplication of input R, G, B (or YUV) values by a 3 x 3 matrix and the addition of an offset. It is fully configurable via software and can be clamped to a reduced dynamic range if necessary.

3.2.2 Pipe1: Main pipe, postprocessing part

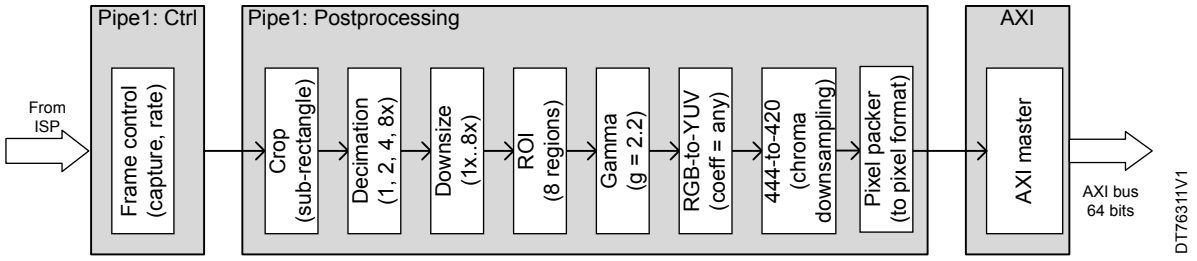
The postprocessing Pipe1 (illustrated in Figure 27) is a pixel pipeline in which postprocessing functions are integrated. The input is from ISP pipe. The applicative Pipe1 is the most featured pixel pipeline, even if the image-processing functions can be shared with Pipe2.

Once the input source is chosen, the data enters Pipe1: Control, where the frame control module manages the capture rate. Then, the data proceeds to Pipe1 postprocessing, which includes several modules. After these processing steps, the data is passed to the pixel packer, which prepares it for transmission to the final output. The AXI master module then sends the processed data over a 64-bit AXI bus with a 2 µs latency.

Pipe1 integrates postprocessing modules to relieve the software from time and memory-consuming operations. These functions include cropping to retain only the necessary part of the image for the targeted application, decimation as a cost-effective preprocessing step before downsizing, and downsizing to reduce image resolution.

Additionally, it handles ROI signaling that helps object detection applications, gamma correction before sending data to memory for display operations, and YUV conversion with optional down-sampling to optimize memory usage or match a YUV420 video encoder.

Figure 27. Data flow and processing modules in Pipe1



3.2.3 Pixel 2D cropping

The pixel 2D cropping aims at extracting and forwarding a rectangle of pixels. The cropping assigns an X and Y location to the input pixels, selects a configured rectangle among the pixels, and forwards the pixels of this rectangle farther in the pixel pipe. The pixels around the selected rectangle are discarded.

3.2.4 Regions of interest (ROI)

Pixel pipes can define up to eight different ROIs to highlight areas in a picture. The purpose of this feature is to draw one rectangle with user size defined for each enabled ROI.

There is no hardware processing within the defined area applied on the image.

As the aim is to draw rectangles on a display, the ROI is supposed to be used when the data flow is RGB at the output of the image-processing blocks.

3.2.5 Gamma conversion

This feature implements a gamma compression on each R, G, B component, using a static gamma exponent 2.2. It stores gamma-compressed pictures, as defined in IEC 61966-2-1.

3.2.6 YUV conversion

This module converts the pixels to YUV for JPEG or video encoding applications.

3.2.7 Chroma-down sampling

The chroma-down sampling module reduces the chroma resolution from the YUV444 format (arriving) to the potential YUV420 output and sends them to a different channel.

3.2.8 Pixel packing

This module operates on the dump pipe (Pipe0) and pixel pipes (Pipe1 to N), organizing the incoming pixels into memory words.

It maps the RGB or YUV data pipeline to standard memory words, supporting various pixel formats.

For RGB:

- ARGB8888

- RGBA8888
- RGB888p
- RGB565

For YUV:

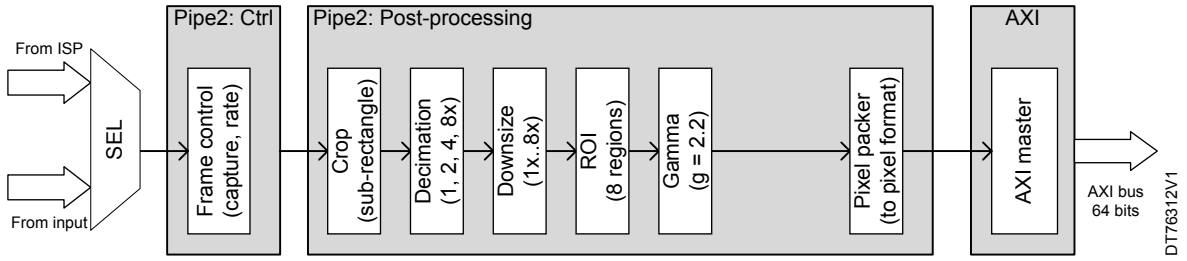
- 8 bpp for monochrome
- 1-buffer (444, 422)
- 2-buffer (422, 420)
- 3-buffer (420)

Additionally, it offers the capability to swap R and B (and Cr and Cb) components for more flexibility.

3.3 Pipe2: Postprocessing part

The main components of Pipe2, including the controller and general postprocessing functions, are almost identical to those of Pipe1. However, Pipe2 is a lighter pixel pipeline. It can operate in standalone mode or share image-processing functions with Pipe1. As illustrated in Figure 28, this pipeline manages the input and output flow within the image-processing system. The input can be selected from either the ISP (image signal processor) or a direct input virtual channel through a selector (SEL). The AXI master module then sends the processed data over a 64-bit AXI bus.

Figure 28. Data flow and processing modules in Pipe2



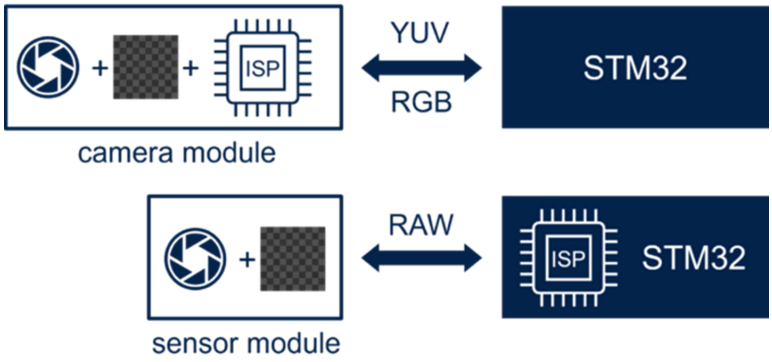
The set of postprocessing blocks is reduced compared to Pipe1, as neither the YUV conversion nor the down-sampling YUV444 to YUV420 are integrated in Pipe2. This pipe can have its own image crop area, dedicated downsizing, and it takes benefit of the gamma correction, similar to Pipe1.

4 ISP calibration and IQTune tool

In contrast to a camera module that includes the lens, image sensor, and ISP, the raw sensor module only includes the lens and image sensor

The ISP processes the raw data from the sensor to produce a viewable image. It handles tasks such as white balance and color correction.

Figure 29. Raw sensor module integration



4.1 The STM32 ISP IQTune tool

The STM32 ISP IQTune is a comprehensive STM32 ISP tuning software application offering a wide range of features and services to STM32 embedded ISP users.

It consists of two components:

- A desktop application, running on a host computer (Linux® Ubuntu®, Windows® or macOS®).
- An embedded application, running on the STM32 device to be tuned.

Linux® is a registered trademark of Linus Torvalds.

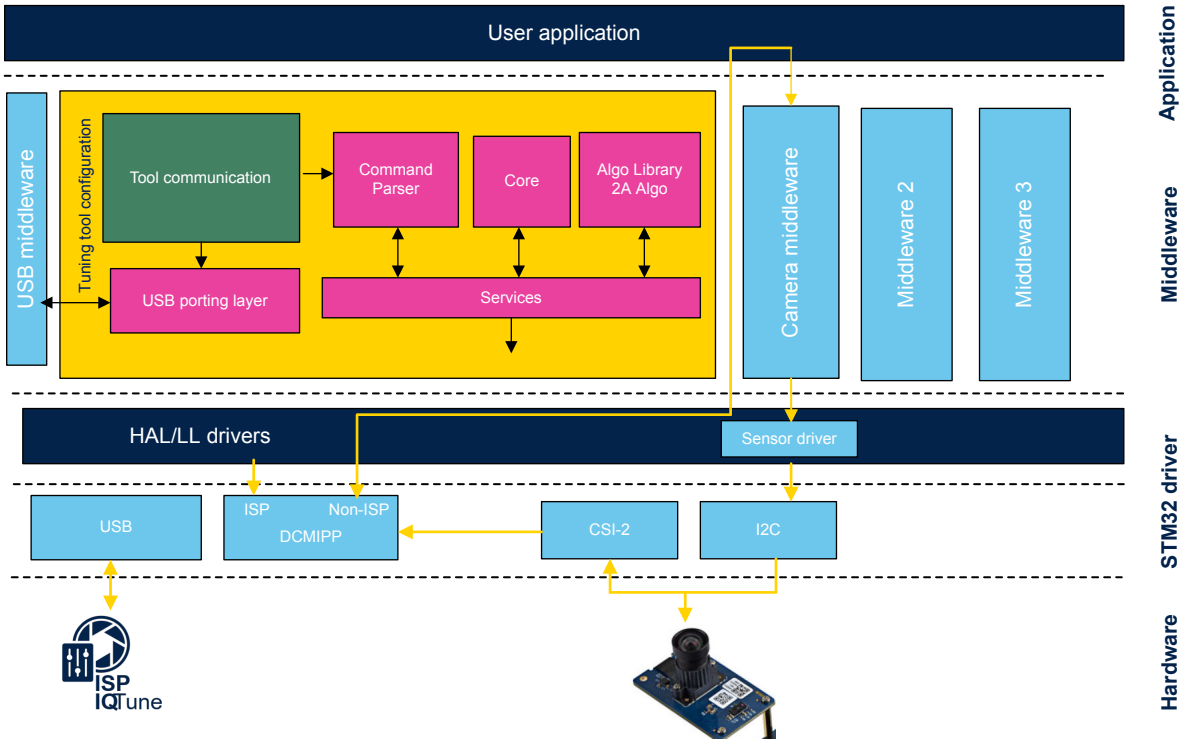
Ubuntu® is a registered trademark of Canonical Ltd.

Windows is a trademark of the Microsoft group of companies.

Ubuntu® is a registered trademark of Canonical Ltd.

The embedded application architecture includes two middleware components to interface with USB and the IMX335 camera sensor. The user application, running on the target device, enables interaction between the STM32 ISP IQTune software (executed on the host) and the target device, allowing users to control and adjust ISP parameters.

Figure 30. ISP IQTune application architecture



DT76329V1

STM32 ISP IQTune provides all the services for users to tune the STM32 ISP for an RGB raw Bayer sensor and its associated lens. It provides image quality analyzing capabilities to easily tune the STM32 ISP for any lighting conditions.

For a white-balance profile, the image quality analyzing services provide:

- The possibility to define the white-balance ISP gain.
- The possibility to define the color correction matrix.
- Information and metrics about the color accuracy according to the ISP configuration defined by the user.

4.2 ISP for a raw sensor

Illuminants, also called different light sources, have varying spectral distributions and color temperatures (measured in Kelvin) that can affect how the colors are captured by the camera sensor. The white balance adjustment (ISP gain) ensures that the whites are accurately represented, and the color adjustment (color correction matrix) ensures that the colors in the captured image closely match the actual scene.

ISP parameters for different lighting conditions must be tuned during the camera application development process. It ensures that the sensor and lens connected to the device can produce good image quality in terms of white balance and color accuracy under different lighting conditions, meeting user application needs. The results of the tuning procedure are a set of parameters programmed in the ISP and controlled by ISP algorithms.

Note: For more details about how to tune ISP using the STM32 ISP IQTune, users can check How to tune ISP using the STM32 ISP IQTune wiki article.

The ISP needs to be tuned when used with an RGB raw sensor. With a monochrome raw sensor, the ISP does not need to be tuned.

5 DCMIPP pipeline configuration for various sensor types

This section includes configurations for different types of sensors, including RGB, raw Bayer, monochrome (grayscale), YUV, and JPEG sensors. Each section provides specific instructions for setting up and capturing data from these sensors using the DCMIPP pipes of the STM32 microcontroller.

5.1 Configuration for RGB sensors

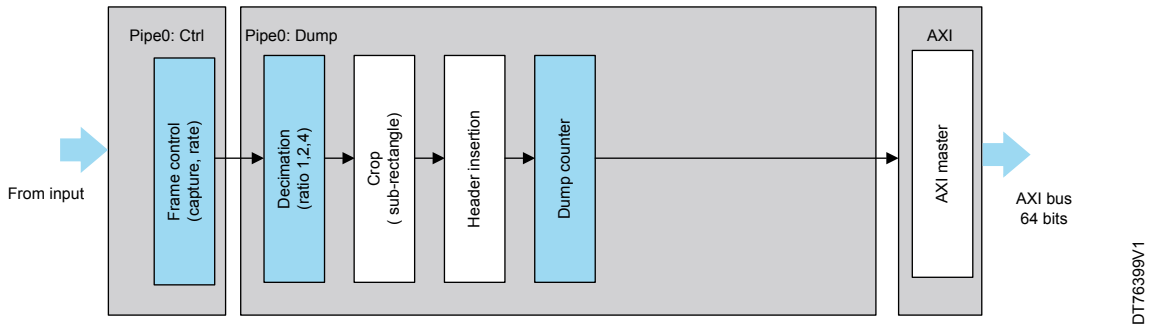
Incoming RGB pixels from the sensor can be captured by Pipe0, Pipe1, or Pipe2, depending on the DCMIPP capabilities of the STM32 microcontroller.

Pipe0 configuration

The dump pipe supports more formats than the pixel pipe. One of these formats is the RGB data type. Before Pipe0 retrieves data through the 16-bit parallel interface or the CSI-2 interface connected to the RGB sensor module, the flow selection module must be configured to handle the choice of virtual channels, data, and the acquisition interface.

To correctly capture RGB pixels and write them to the memory using Pipe0, the sensor should be set to output in the adequate RGB format, and Pipe0 needs to be configured to capture frames. Additionally, 2D-cropping and decimation operations can be configured by the software if needed. A dump counter is available to handle unknown lengths of data or to limit the amount of data within a frame. The pixel packer stage packs the pixels into the desired format and buffer, which serves as the storage area for the data.

Figure 31. Capturing RGB pixels using Pipe0

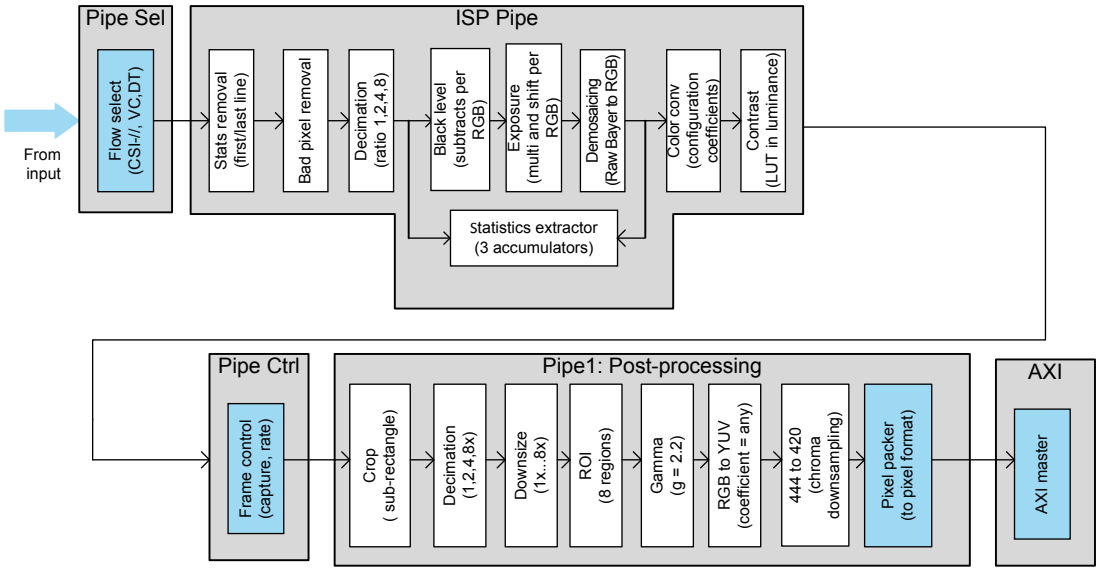


Pipe1 configuration

The same applies to Pipe1. The flow selection module and frame control are replicated for each dump and pixel pipe, managing the same parameters (CSI-//, VC, DT) along with the capture mode (continuous, snapshot) and rate settings of frame control. The pixel packer module then works on the dump pipe (Pipe0) and pixel pipe (Pipe1), setting the arriving pixels into memory words.

To capture RGB pixels and write them to memory using Pipe1 (which additionally handles image processing (ISP) and post-processing alongside pixel dumping), the modules highlighted in Figure 32 should be enabled. The remaining modules, such as decimation and downsize, are optional and can be utilized for additional image-processing functions.

Figure 32. Capturing RGB pixels using Pipe1

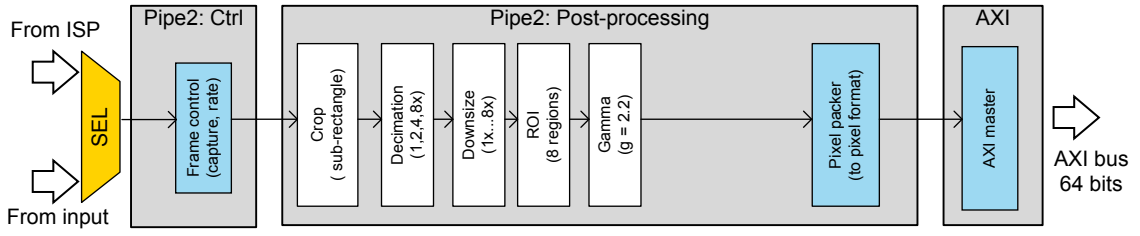


Pipe2 configuration

To capture RGB pixels and write them to memory using Pipe2, the user needs to configure the pixel packer registers with the following settings:

- Specify the pixel format
- Set the width of the image in pixels
- Define the line stride (pitch) in bytes, which is the number of bytes between the start of one line and the start of the next line (that is, 5760 bytes for RGB888 format)
- Set the height of the image in pixels

Figure 33. Capturing RGB pixels using Pipe2



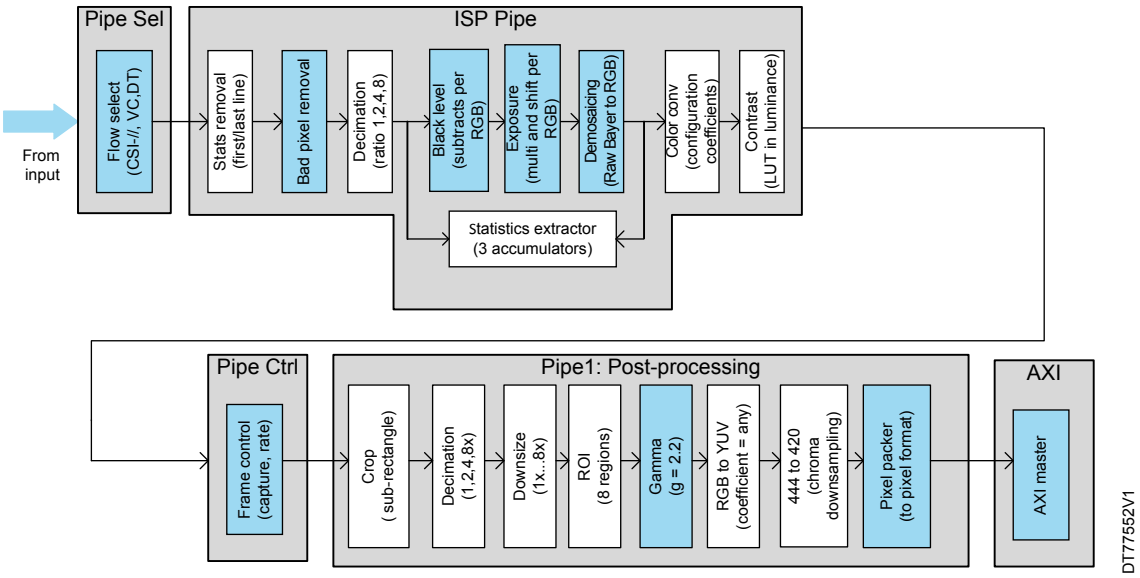
5.2 Configuration for raw Bayer sensors

This section focuses on capturing raw Bayer pixels from the sensor and transferring the processed data to the memory. The raw Bayer pixels must pass through the highlighted modules in Figure 34 to be converted into RGB pixels:

1. Bad pixel removal: enable this module to detect and correct artifacts from the sensor.
2. Black Level: this module adjusts the black level offset, which occurs when the sensor sends non-null pixels while capturing a black picture. R, G, and B component offsets are subtracted from the R, Gr, Gb, and B Bayer components.
3. Exposure: this module multiplies the R, G, and B components by a configurable gain such as to recenter the colors to more accurate white tones.
4. Demosaicing: this module implements a mid-quality conversion from raw Bayer 8 bpp to RGB 24 bpp. The output RGB pixel rate matches the input component rate.
5. Gamma: this module applies gamma correction to each R, G, and B color component as defined in the sRGB standard.
6. Pixel packer: once data is converted, this stage packs the processed pixels into the desired format.

The remaining modules are optional and can be used for additional image-processing functions.

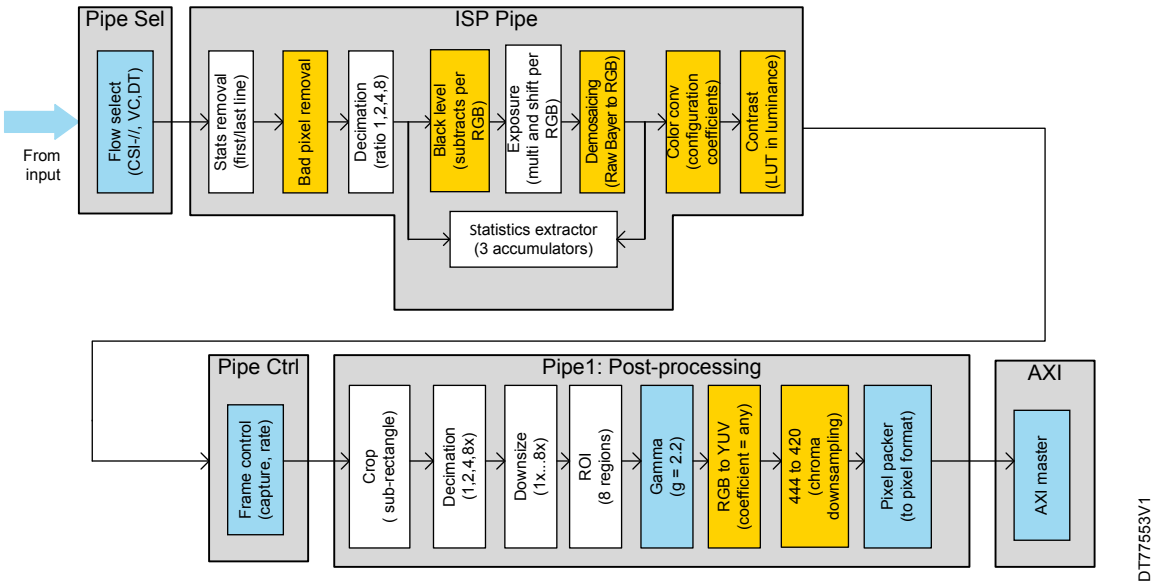
Figure 34. Capturing raw Bayer pixels using Pipe1



5.3 Configuration for monochrome (grayscale) sensors

To capture and process grayscale image data from a monochrome sensor, the user can utilize either the dump pipe or the pixel pipe (Pipe1) for further processing. For incoming grayscale pixels, it is necessary to enable the flow selection to direct the data flow through Pipe1. In this configuration, the following modules should be disabled: bad pixel removal, black level, demosaicing, color conversion and contrast.

Figure 35. **Capturing monochrome pixels using Pipe1**



5.4 Configuration for YUV sensors

To capture YUV pixels from YUV sensors and process them, either dump pipe (Pipe0) or pixel pipes (Pipe1, Pipe2) can be used. Dumping through Pipe0 is ideal for applications that require luminance and chrominance information. Alternatively, using Pipe1 to process YUV data and convert it to RGB using the ColorConv stage.

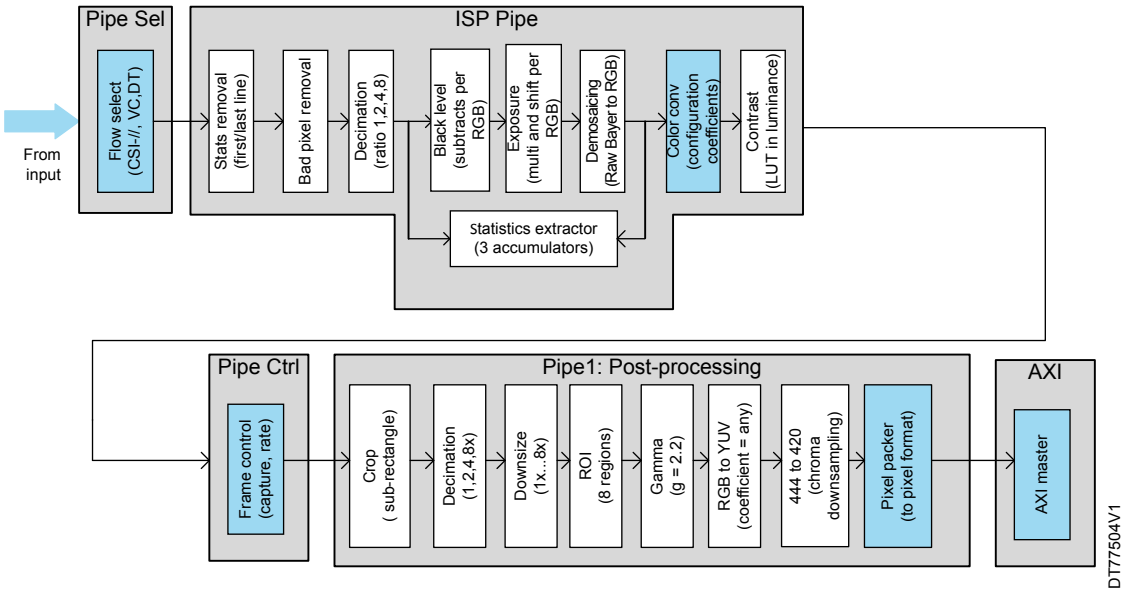
Pipe0 and Pipe1 configuration

In the YUV configuration with either Pipe0 or Pipe1, the same modules are activated as for RGB sensors, with the only difference being the change of data type in the module flow selection.

Pipe1 configuration

To process the YUV pixels and convert them to RGB format using the ColorConv module, enable the highlighted modules, namely flow select, Colorconv, frame control, and pixel packer, as they are essential. The rest of the post-processing modules are optional such as the downsize stage for minimizing resolution or the chroma down-sampling module for reducing chroma resolution from the incoming YUV444 format to a potential YUV420 output.

Figure 36. **Capturing YUV pixels using Pipe1**

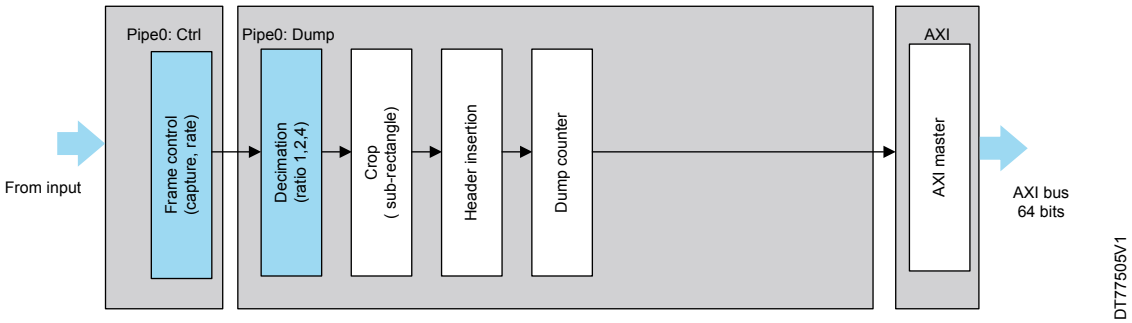


5.5 Configuration for JPEG sensors

To capture a JPEG-compressed stream from a JPEG sensor via the parallel or CSI-2 interfaces, the user must use Pipe0 by enabling the modules highlighted in Figure 37. When selecting a parallel interface, the DCMIPP Pipe0 should be configured with byte pixel format, with swap cycles disabled, and hardware synchronization enabled.

Using the CSI-2 camera sensor, the user needs to configure the virtual channel and data types to JPEG byte stream mode for Pipe0 via the flow selection module.

Figure 37. **Capturing JPEG pixels using Pipe0**



The user must ensure that the decimation and crop modules are disabled, as their functionalities are not supported when the pipe is processing JPEG format.

Note: In byte stream mode, the bytes are dumped consecutively with only a 32-bit padding (if needed) at the end of the frame (or JPEG stream).

There is no 32-bit padding operation inside the stream until the VSYNC deassertion (end of frame, or end of JPEG stream).

6 DCMIPP application examples

This section details how to use the DCMIPP and provides step-by-step implementation examples.

6.1 DCMIPP use cases

The DCMIPP and other STM32 peripherals can significantly enhance computer vision applications on the edge. Here are some application examples:

Table 6. DCMIPP application examples

Category	Application
Building	• People counting • Occupancy estimation
Home	• Home appliances • Home security • Door phone and home security
Retail	• Customer behavior tracking • Shelf storage anomalies
City	• Crowd behavior • Parking spot occupancy
Automotive	Driver monitoring
Farming	• Disease detection • Livestock health and position monitoring
Industrial	• Defect detection • Visual anomaly detection • Industrial monitoring systems and automated inspection
Security and surveillance	• Security and video surveillance • System control • Access control systems
Video streaming	• Real-time video streaming • Battery-powered video camera

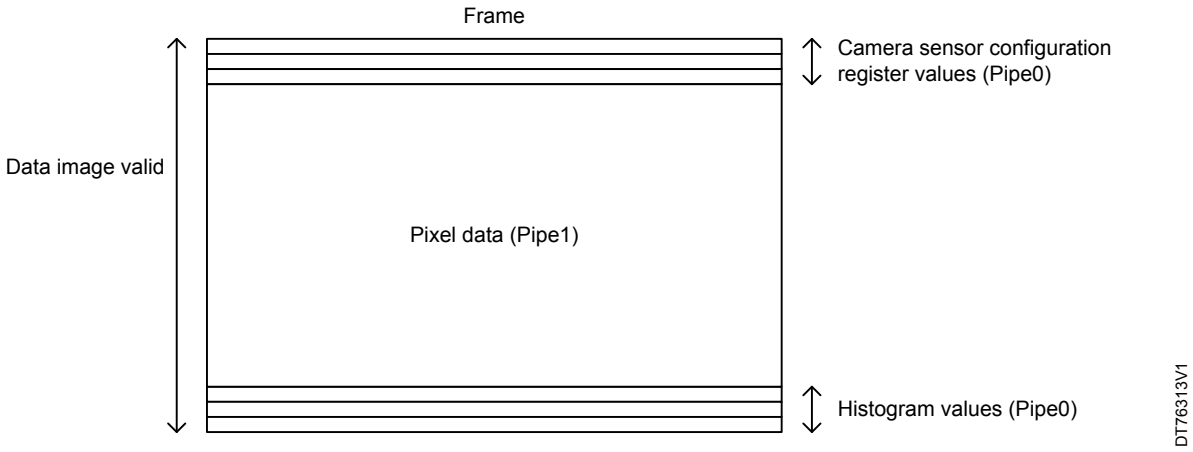
6.1.1 Parallel interface camera sensor module

A parallel interface camera sensor can transfer images and statistics. It is easier to manage the software processing by splitting the statistics and the image data on different memory storages. Two pipes are used in such cases, with Pipe0 extracting the statistics and Pipe1 dealing with the image data (whatever the data format supported by the DCMIPP).

Both Pipe0 and Pipe1 work in parallel on the same frame, frame by frame.

Each output FIFO attached to the pipe conveys the data on the targeted memory for further processing. The camera sensor module may use some lines of the frame to transfer statistics or sensor configuration data. These lines can be placed at the beginning and at the end of the frame (see Figure 38), even if there is no real standard.

Figure 38. Use case: Pipe0 (statistics), Pipe1 (pixels)



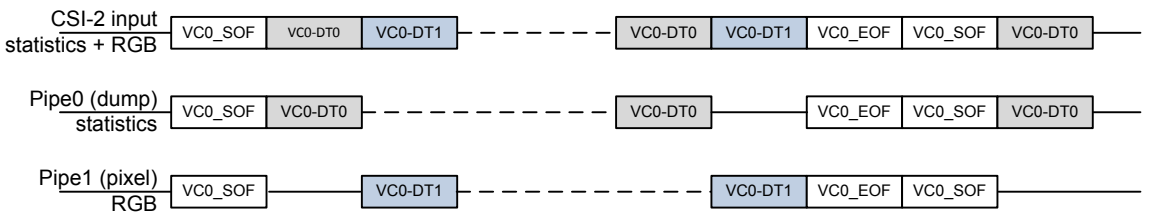
6.1.2 CSI-2 camera sensor module

The CSI-2 protocol uses data types to distinguish between different data categories within a single virtual channel, involving two pipes (that is, Pipe0, and Pipe1). In Figure 39, the CSI provides two parallel data flows: RGB (or raw Bayer) and its sideband statistics.

The statistics are handled by the dump pipe (Pipe0), while the RGB data is sent to the pixel pipe (Pipe1).

The data flow starts with a virtual channel 0 start of frame (VCO_SOF), followed by data for statistics (VCO_DT0) and RGB data (VCO_DT1), and ends with a virtual channel 0 end of frame (VCO_EOF). The CSI-2 protocol distinguishes these data types using specific data type identifiers (DT) within the data stream, predefined codes representing different data types, and frame markers such as VCO_SOF and VCO_EOF.

Figure 39. Use case with CSI-2 interface: Pipe0 (statistics) and Pipe1 (pixels)



6.1.3 Example of ISP processing

The ISP processing example illustrates the image-processing pipeline for a raw Bayer. The process begins with **raw Bayer demosaicing**, which is performed in the raw Bayer module to convert the raw image data into a full-color image. Then, **Exposure Control** is applied, where the luminance average (via R, G, B) is extracted from statistics. This information is fed back into the exposure module, using a median value of 64 to adjust the exposure settings.

Following this, **White Balance** is conducted by extracting the R, G, B averages from statistics and feeding this data back into the exposure module for each RGB component, after the **Gray World algorithm (or similar)** algorithm to ensure neutral color balance.

Finally, **Contrast Enhancement** is performed by extracting luminance bins from statistics and feeding this information into the contrast module, using a median value of 128 to enhance the contrast of the image. This step-by-step processing ensures that the final image is well-exposed, color-balanced, and has enhanced contrast for better visual quality.

Figure 40. Example of ISP processing

RawBayer picture

RawBayer democaising
- Done in RawBayer module

Exposure control
- Luminance average (via R,G,B) extracted from Stats.
- Feedback onto Exposure Module (64 as Median)

White balance
- R,G,B averages extracted from Stats.
- Feedback onto Exposure Module (Grey World)

Contrast enhancement
- Luminance bins extracted from Stats.
- Feedback onto Contrast Module (128 as Median)

DT76314V1

6.1.4 Concurrent processing using multiple pipelines (capture and AI inference on STM32N6x5/6x7)

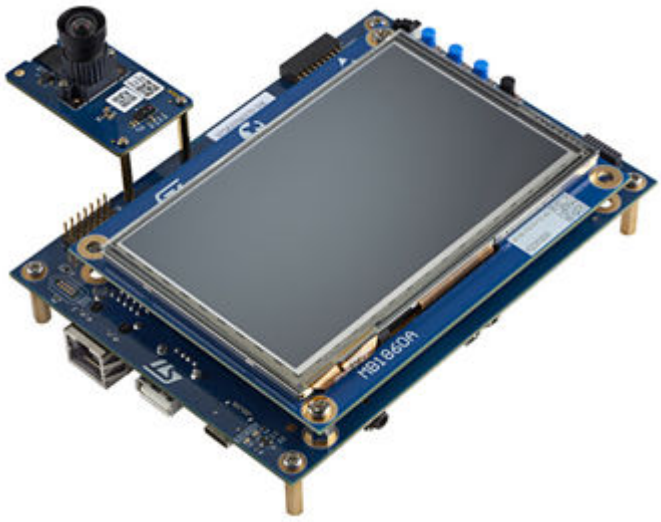
6.1.4.1 Hardware description

The hardware is composed of the MB1939 STM32N6570-DK development kit and the MB1854B IMX335 camera module (5Mpx/30fps), which is the default camera provided with the MB1939 board.

The STM32N6570-DK features the STM32N6570 microcontroller, designed for high-performance applications requiring advanced image processing and AI capabilities.

The MB1854B IMX335 camera module captures high-resolution images and video with its high-quality image sensor.

Figure 41. STM32N6570-DK development kit



DT76325V1

The characteristics of the pins are as follows:

- Pipe 1 is enabled to continuously transmit images from the imx335 sensor to the DISPLAY_BUFFER buffered.
- Pipe 2 is enabled using to continuously transmit images from the imx335 sensor to the double-buffered.

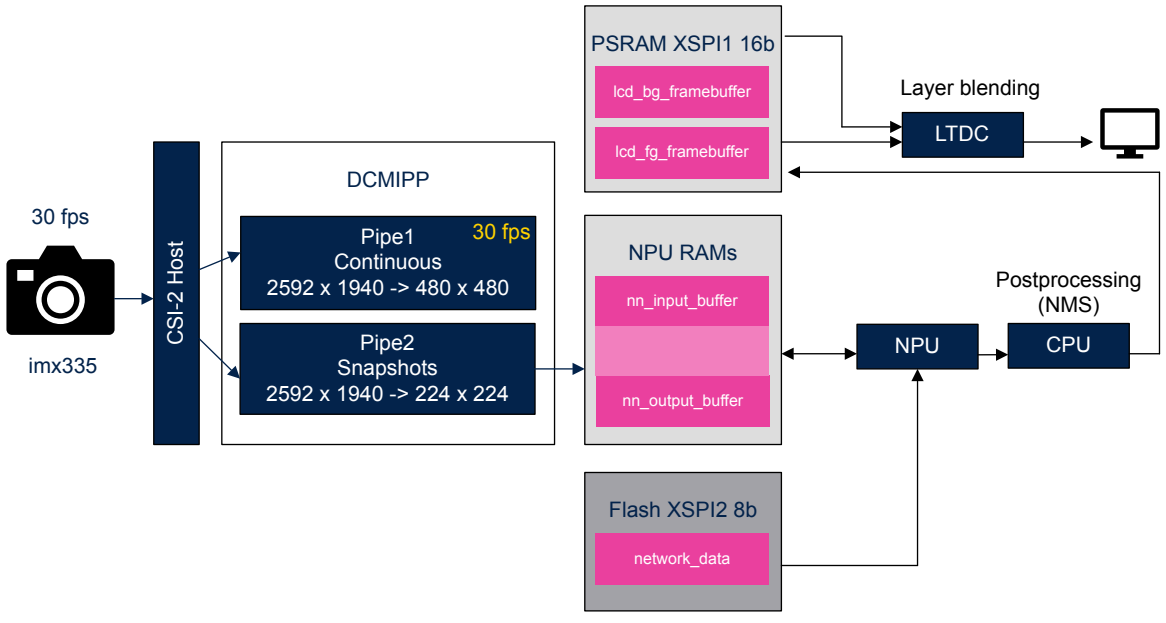
The following table provides information about different framebuffers used in capture and inference in STM32N6x5/6x7.

Table 7. Framebuffer specifications and locations

Framebuffer name	Size	Format	Location
LCD foreground buffer	4608 KB	RGB888 (800x480x3) x 4	External XSPI1 PSRAM
LCD background buffer	1500 KB	ARGB4444 (800x480x2) x 2	External XSPI1 PSRAM
Input neural network buffer	221 KB	RGB88 (192x192x3) x 2	External XSPI1 PSRAM

The DCMIPP interface features two pipelines: Pipe 1 for continuous image capture at a resolution of 2592 x 1940 downscaled to 480 x 480, and Pipe 2 for snapshots at the same resolution downscaled to 224 x 224.

Figure 42. DCMIPP use case example



6.2 STM32Cube examples

The STM32CubeH7RS and STM32CubeN6 MCU Packages offer a large set of examples implemented and tested on the corresponding boards.

The table below gives an overview of the DCMIPP examples and applications across various STM32Cube. All these examples are developed to capture RGB data. For most of the examples, the user can select one of the following resolutions: 800 x 480, 2592 x 1944.

Table 8. STM32Cube DCMIPP examples

MCU Package	Project name	Board
STM32CubeH7RS	DCMIPP_ContinuousDBM	STM32H7S78-DK
	DCMIPP_ContinuousMode	
	DCMIPP_JPEGSnapshotMode	
STM32CubeN6	DCMIPP_ContinuousDBM	STM32N6570-DK

6.3 DCMIPP examples based on STM32CubeMX

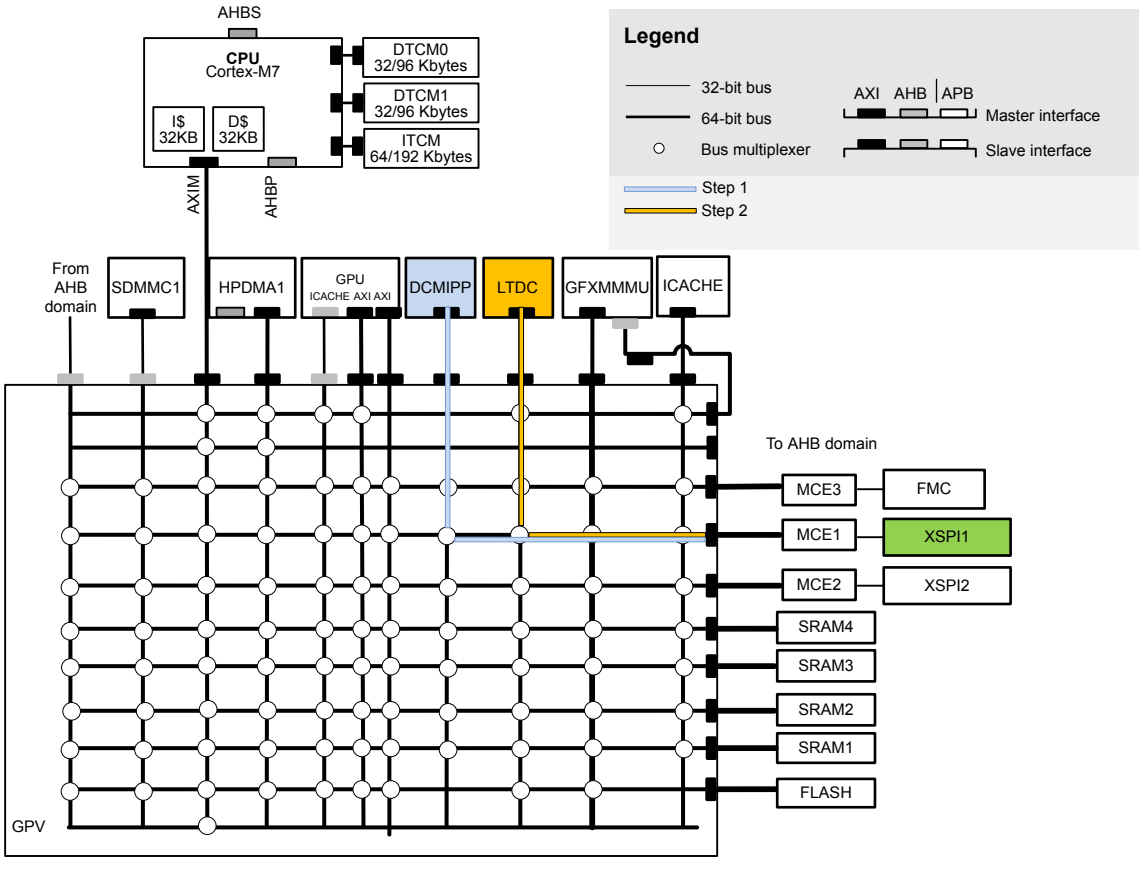
This section provides detailed examples of typical DCMIPP configuration, focusing on capturing and displaying RGB data. It demonstrates how to configure the camera in continuous mode with a resolution of 480 x 272, using the STM32H7S78-DK board from STMicroelectronics connected to the 5-megapixel OV5640 camera sensor.

The DCMIPP Pipe0 is configured for RGB565, with swap cycles enabled, framerate set to DCMIPP_FRAME_RATE_ALL, and hardware synchronization enabled. All frames are stored in the camera framebuffer, and the LTDC is configured to continuously display these captured frames from the camera framebuffer.

As illustrated in Figure 43, the application consists of two main steps:

1. DCMIPP dumps to a single RGB buffer. It has a single pipe (Pipe0) active, where all the AXI capabilities are focused on the single active pipe, which gets all the FIFO space. With this setting, a camera dump of 200 Mpixel/s peak on 32 bpp (800 MB/s bandwidth) benefits from 5 Kbytes FIFO.
2. Import data from the external memory to be displayed on the LCD-TFT, only for RGB data format. For YCbCr or JPEG data format, the user must convert the received data to RGB to be displayed.

Figure 43. Data path in capture and display application



6.3.1 Hardware description

The example uses the STM32H7S78-DK board from STMicroelectronics, connecting to the 5 megapixel OV5640 camera sensor, as illustrated in Figure 44.

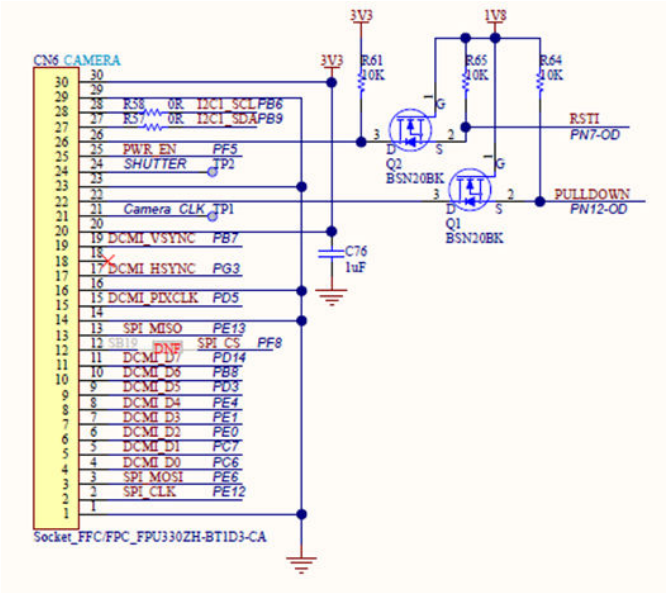
The camera module with a resolution of 480x272 is connected to the STM32H7S78-DK board through the following signals.

The configuration of this example can be done by following the steps described in Section 6.3.2: Configuration of common examples.

This camera module is interfaced with the DCMIPP:

- Control signals: DCMIPP_PIXCLK, DCMIPP_VSYNC, DCMIPP_HSYNC
- Image data signals: DCMIPP_D[0..7]

Figure 44. **Camera connector on STM32H7S78-DK**



DT76316V1

For more details on the STM32H7S78-DK board, refer to the document [8] available on the STMicroelectronics website.

6.3.2 Configuration of common examples

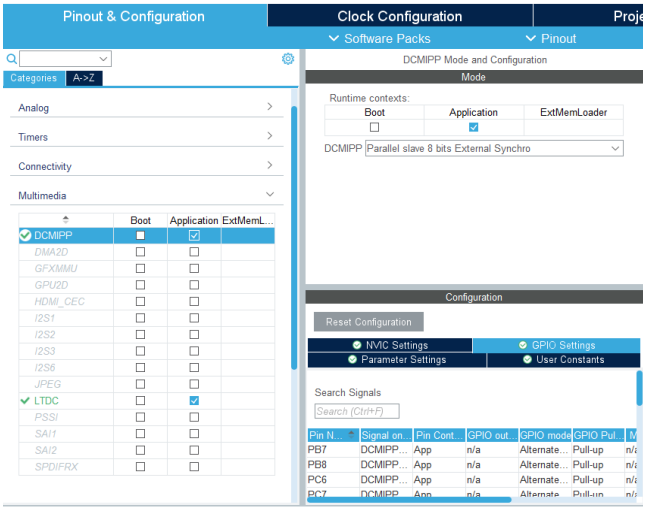
When starting with STM32CubeMX, the first step is to configure the project location and the corresponding toolchain or IDE (menu Project/Settings).

6.3.2.1 *STM32CubeMX - Configuration of DCMIPP GPIOs*

DCMIPP GPIOs can be configured following these steps:

1. Select the DCMIPP and choose “Slave 8 bits External Synchro” in the pinout and configuration multimedia tab to configure the DCMIPP in parallel slave 8-bit external synchronization. After this step, nine pins must be highlighted in green (D[0..7] , DCMIPP_VSYNC, DCMIPP_HSYNC and DCMIPP_PIXCLK).

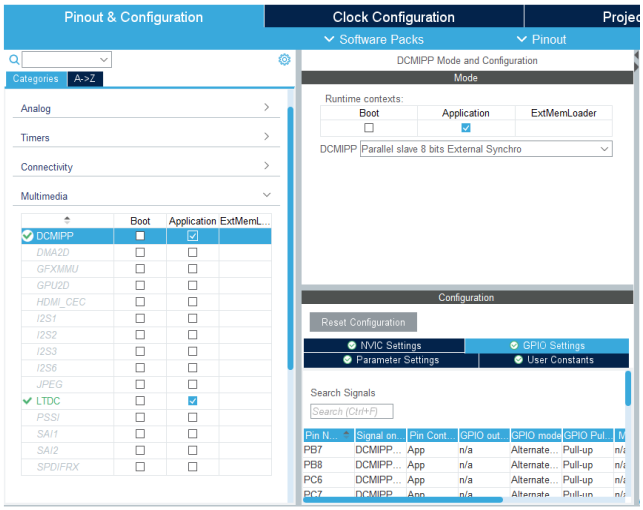
Figure 45. STM32CubeMX - DCMIPP synchronization mode selection



DT76317V1

2. Select the "Configuration tab" to configure the GPIOs mode.
3. When the DCMIPP configuration window appears, select the GPIO Settings tab.

Figure 46. STM32CubeMX - GPIO settings selection



DT76318V1

4. Select all the DCMIPP pins.

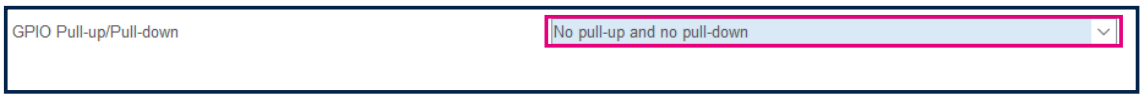
Figure 47. STM32CubeMX - DCMIPP pin selection

Pin Na...	Signal on Pin	Pin Cont...	GPIO ou...	GPIO m...	GPIO P...	Maximu...	User La...	Modifi
PB7	DCMIPP_VSYNC	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PB8	DCMIPP_D6	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PC6	DCMIPP_D0	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PC7	DCMIPP_D1	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PD3	DCMIPP_D5	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PD5	DCMIPP_PIXCLK	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PD14	DCMIPP_D7	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PE0	DCMIPP_D2	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PE1	DCMIPP_D3	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PE4	DCMIPP_D4	Applicati...	n/a	Alternate...	Pull-up	n/a		☑
PG3	DCMIPP_HSYNC	Applicati...	n/a	Alternate...	Pull-up	n/a		☑

DT76319V1

5. Set the GPIO pull-up/pull-down.

Figure 48. STM32CubeMX - GPIO no pull-up and no pull-down selection

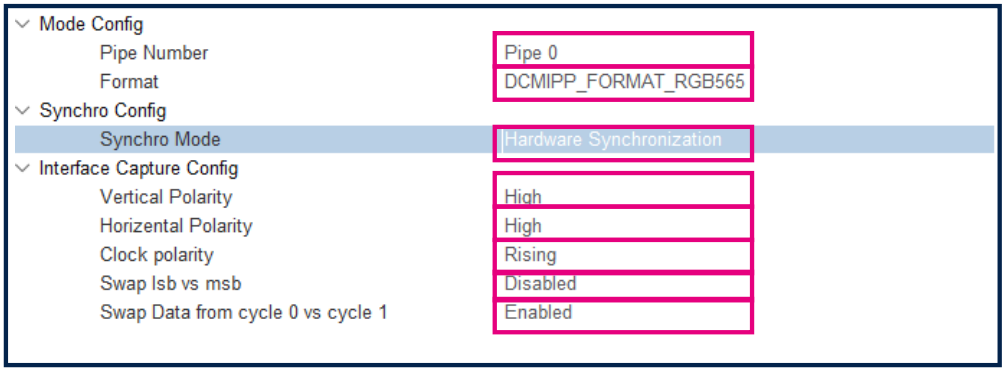


DT76320V1

6.3.2.2 *STM32CubeMX - Configuration of DCMIPP control signals and capture mode*

1. Click on Parameter Settings tab in DCMIPP Configuration window

Figure 49. STM32CubeMX - DCMIPP control signals and capture mode



DT76321V1

2. Set the different parameters, including frame rate, pipe number, data format, synchronization configuration, and pixel clock polarities, according to the camera module configuration.

Note: The vertical synchronization polarity must be active high, and the horizontal synchronization polarity must be active low. They must not be inverted for this configuration of the camera module.

6.3.2.3 *STM32CubeMX - Enable DCMIPP interrupts*

1. Select NVIC Settings tab in DCMIPP Configuration window and check the DCMIPP global interrupt.

Figure 50. STM32CubeMX - Configuration of DCMIPP interrupts

DT76322V1

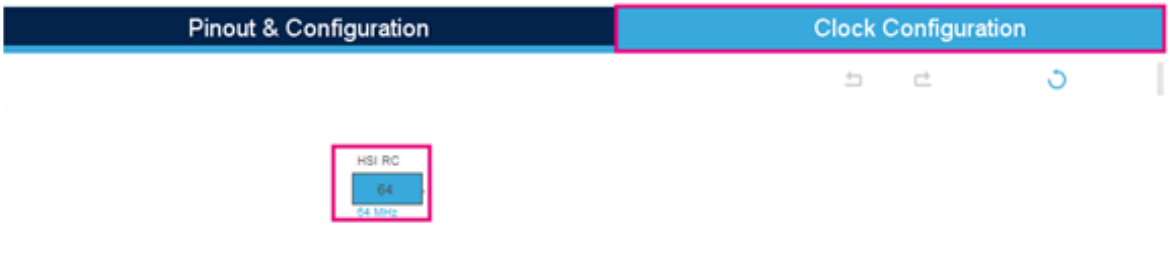
6.3.2.4 *STM32CubeMX - System clock configuration*

In this example the system clock is configured as follows:

- Use of internal HSI clock, where the main PLL is used as system source clock
- HCLK at 300 MHz, so the Cortex-M7 running at 600 MHz

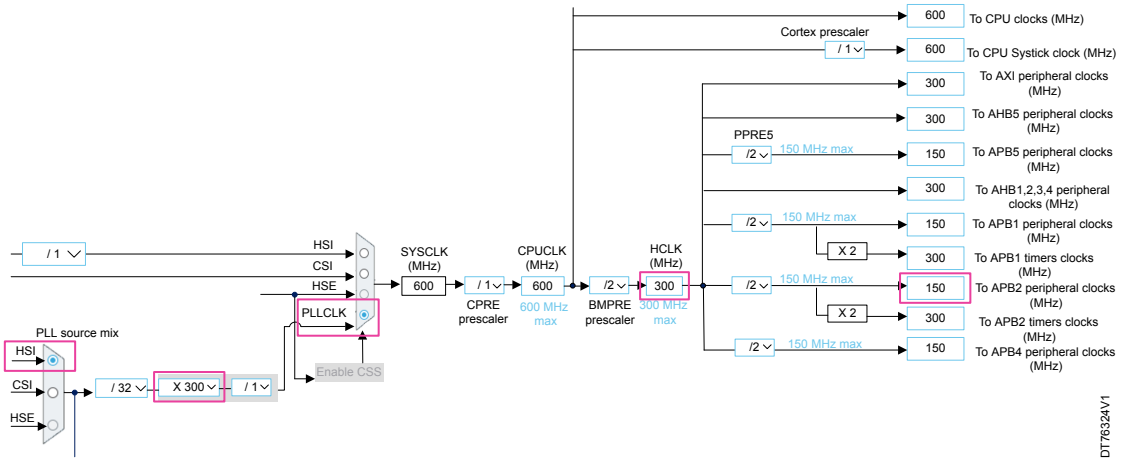
1. Select the Clock Configuration tab.

Figure 51. STM32CubeMX - HSI configuration



2. Set the PLLs and the prescalers in the Clock Configuration tab, to get the system clock HCLK at 300 MHz.

Figure 52. STM32CubeMX - HCLK configuration



6.3.2.5 *Adding files to the project*

Generate the code and open the generated project using the preferred toolchain. Then follow these steps:

- Copy ov5640.c from the Components folder to the Src folder.
- Copy ov5640_reg.c from the Components folder to the Src folder.
- Copy ov5640.h from the Components folder to the Inc folder.

6.3.2.6 *Modifications in main.h file*

Update main.h by inserting some instructions to include the needed files in the adequate space This task provides the project modification and regeneration without losing the user code:

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "camera.h"
#include "ov5640.h"
#include "string.h"
#include "stm32h7s78_discovery.h"
#include "stm32h7s78_discovery_bus.h"
/* USER CODE END Includes */
```

Some variable declarations must then be inserted in the adequate space:

```
/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */
#define FRAME_BUFFER_SIZE    (480*272*2 / 4)
#define CAMERA_OV5640_ADDRESS 0x78
/* USER CODE END EC */
```

6.3.2.7 *Modifications in main.c file*

Update main.c by inserting some instructions to include the needed variables:

```
/* USER CODE BEGIN PV */
static CAMERA_Drv_t *Camera_Drv = NULL;
uint32_t CAMERA_FRAME_BUFFER[FRAME_BUFFER_SIZE] __attribute__((aligned(16)));
uint32_t xpos = (800 - 480)/2;
uint32_t ypos = (480 - 272) /2;
uint32_t ImageWidth = 480;
uint32_t ImageHeight = 272;
/* USER CODE END PV */
```

The function prototypes must also be inserted in the adequate space:

```
/* USER CODE BEGIN PFP */
static uint32_t OV5640_Config(uint32_t Resolution, uint32_t PixelFormat);
/* USER CODE END PFP */
```

Update main() function by inserting some functions in the adequate space:

- OV5640_Config allows the configuration of the camera module, the DCMI registers, and the DCMI parameters.
- HAL_DCMIPP_PIPE_Start enables DCMIPP capture on Pipe0 by specifying the destination address and the continuous mode.

```
/* USER CODE BEGIN 2 */
/* Configure OV5640 camera Module */
if(OV5640_Config(OV5640_R480x272, OV5640_RGB565) != OV5640_OK)
{
/* Camera Module Config KO */
Error_Handler();
}
/* Start Camera w/ continuous Mode */
if(HAL_DCMIPP_PIPE_Start(&phdcmipp, DCMIPP_PIPE0, (uint32_t)CAMERA_FRAME_BUFFER , DCMIPP_MODE
_CONTINUOUS) != HAL_OK)
{
Error_Handler();
}
/* USER CODE END 2 */
```

Insert the implementation of the new functions (called in the main() function), out of the main function, in the adequate space:

```
static uint32_t OV5640_Config(uint32_t Resolution, uint32_t PixelFormat)
{
OV5640_IO_t IOCtx;
uint32_t id;
uint32_t ret = OV5640_OK;
static OV5640_Object_t OV5640Obj;

/* Configure the Camera driver */
IOCtx.Address = CAMERA_OV5640_ADDRESS;
IOCtx.Init = BSP_I2C1_Init;
IOCtx.DeInit = BSP_I2C1_DeInit;
IOCtx.ReadReg = BSP_I2C1_ReadReg16;
IOCtx.WriteReg = BSP_I2C1_WriteReg16;
IOCtx.GetTick = BSP_GetTick;

/* Register Bus IO */
If(OV5640_RegisterBusIO (&OV5640Obj, &IOCtx) != OV5640_OK)
{
ret = OV5640_ERROR;
}
```

```
/* Read ID */
if(OV5640_ReadID(&OV5640Obj, &id) != OV5640_OK)
{
ret = OV5640_ERROR;
}
if(id == OV5640_ID)
{

/* Initialize the camera Module */
Camera_Drv = (CAMERA_Drv_t *) &OV5640_CAMERA_Driver;

OV5640_DeInit(&OV5640Obj);
if(Camera_Drv->Init(&OV5640Obj, Resolution, PixelFormat) != OV5640_OK)
{
ret = OV5640_ERROR;
}
else if(Camera_Drv->MirrorFlipConfig(&OV5640Obj, OV5640_MIRROR_FLIP_NONE) != OV5640_OK)
{
ret = OV5640_ERROR;
}
}
return ret;
}
void HAL_DCMIPP_PIPE_FrameEventCallback(DCMIPP_HandleTypeDef *pHdcmipp, uint32_t Pipe)
{

/* Assert display enable LCD_DISP_EN pin */
HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, GPIO_PIN_SET);
if(Pipe == DCMIPP_PIPE0)
{
BSP_LED_Toggle(LD1);
}
}
/* USER CODE END 4 */
```

At this stage, the user can build, debug, and run the project.

Note: For LTDC configuration, the user can configure LTDC according to document Section 1: General information.

6.3.3 RGB data capture and display

To simplify this example, data are captured and displayed in RGB565 format (2 bpp). The image resolution is 480 x 272. The framebuffer is placed in the external flash. Camera and LCD data are in the same framebuffer. The LCD displays then directly the data captured through the DCMIPP without any processing. The camera module is configured to output RGB565 data, 480 x 272.

The configuration of this example can be done by following the steps described in Section 6.3.2: Configuration of common examples.

7 Conclusion

The DCMIPP offers an efficient interface to connect the camera modules to the STM32 MCUs supporting high speed, high resolutions, and a variety of pixel formats/widths.

The STM32 MCUs, equipped with the DCMIPP, are capable of handling extensive computer vision applications due to the integrated pipeline that manages image processing.

This application note gives the STM32 users some basic concepts, with easy-to-understand explanations of the features, architecture, and configuration of the DCMIPP.

Revision history

Table 9. **Document revision history**

Date	Version	Changes
08-Apr-2025	1	Initial release.

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics International NV and its affiliates (“ST”) reserve the right to make changes corrections, enhancements, modifications, and improvements to ST products and/or to this document any time without notice.

This document is provided solely for the purpose of obtaining general information relating to an ST product. Accordingly, you hereby agree to make use of this document solely for the purpose of obtaining general information relating to the ST product. You further acknowledge and agree that this document may not be used in or in connection with any legal or administrative proceeding in any court, arbitration, agency, commission or other tribunal or in connection with any action, cause of action, litigation, claim, allegation, demand or dispute of any kind. You further acknowledge and agree that this document shall not be construed as an admission, acknowledgment or evidence of any kind, including, without limitation, as to the liability, fault or responsibility whatsoever of ST or any of its affiliates, or as to the accuracy or validity of the information contained herein, or concerning any alleged product issue, failure, or defect. ST does not promise that this document is accurate or error free and specifically disclaims all warranties, express or implied, as to the accuracy of the information contained herein. Accordingly, you agree that in no event will ST or its affiliates be liable to you for any direct, indirect, consequential, exemplary, incidental, punitive, or other damages, including lost profits, arising from or relating to your reliance upon or use of this document.

Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment, including, without limitation, the warranty provisions thereunder.

In that respect, note that ST products are not designed for use in some specific applications or environments described in above mentioned terms and conditions.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

Information furnished is believed to be accurate and reliable. However, ST assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.