UG10215

i.MX 95 Camera Porting Guide

Rev. 2.1 — 25 September 2025 **User guide**

Document information

Information	Content
Keywords	UG10215, i.MX 95, ISP, sensor porting guide, IPA, camera, NEO-ISP
Abstract	This guide describes the steps to enable a RAW Bayer camera sensor into the i.MX 95 applications processor internal image signal processing on top of Linux camera software stack.

1 Introduction

This guide describes the steps to enable a RAW Bayer camera sensor into the i.MX 95 applications processor internal image signal processing (ISP) on top of the Linux camera software stack.

2 Overview

This document provides an overview of the i.MX 95 applications processor hardware camera subsystem, and then describes its corresponding software architecture within the Linux Kernel and User Space. It also describes the porting instructions and how to configure the 3A algorithms.

3 i.MX 95 applications processor camera domain hardware architecture

The camera subsystem is known as the camera domain in the *i.MX 95 Applications Processor Reference Manual* (document IMX95RM[1]).

The domain provides the following functions:

- Camera interface and capture from two CSI interfaces
- Image signal processing (ISP) on the camera stream

Figure 1 shows the camera domain block diagram.

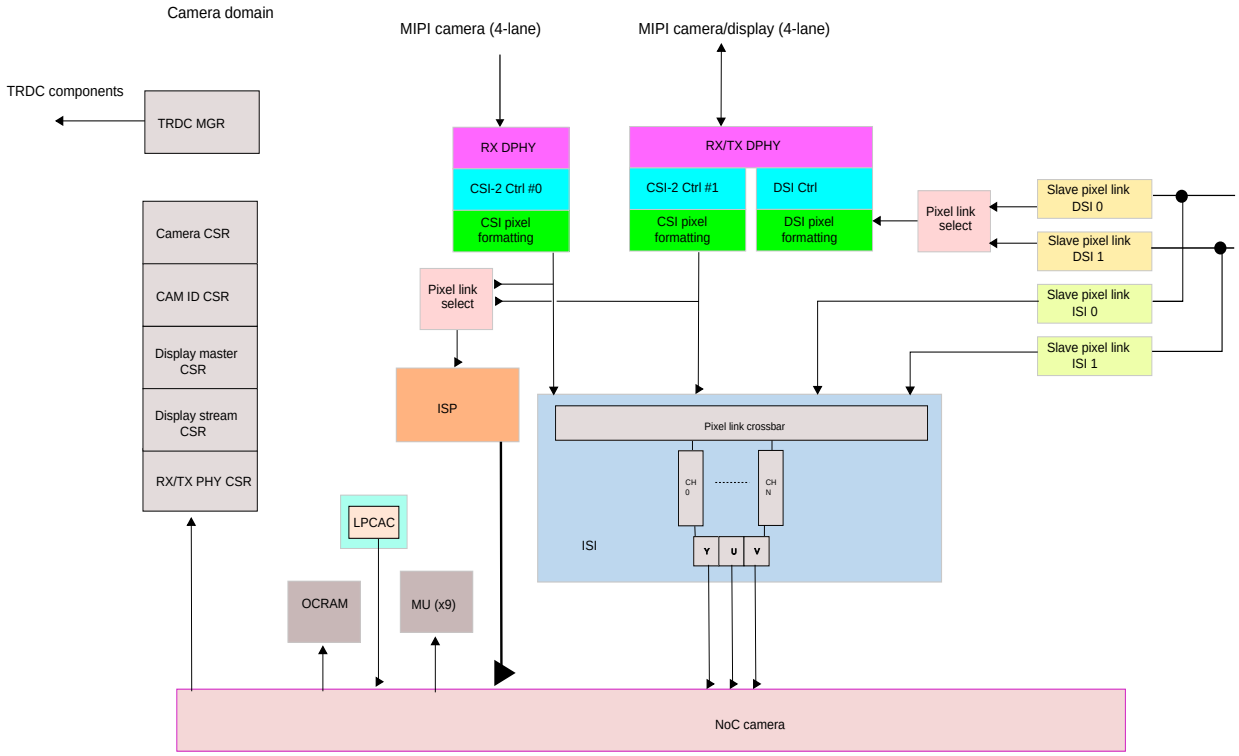

Figure 1. Camera domain hardware architecture

For hardware details of the camera domain modules involved in the Linux camera stack, see "Chapter 12 Camera Domain" in the *i.MX 95 Applications Processor Reference Manual* (document IMX95RM[2]).

1 Contact a local field applications engineer (FAE) or sales representative.
2 Contact a local field applications engineer (FAE) or sales representative.

3.1 i.MX 95 EVK reference camera modules

Table 1 provides NXP reference RAW camera modules used to implement and validate the i.MX 95 applications processor Linux camera software stack.

Table 1. List of camera sensors modules supported

Sensor	Description	Comments
Omnivision OX03C10	• 2.5 Megapixels • 3 exposure HDR (HDR merge performed inside the sensor) • Lenses: 180° H FOV	Connected to i.MX 95 EVK through a MAXIM deserializer.
Omnivision OX05B1S	• 5 Megapixels • RGB-IR4x4 • Single exposure 10 bit • Lenses: 160° H FOV	-
Omnivision OS08A20	• 8 Megapixels • 2 exposure HDR • Lenses: 120° H FOV to 130° H FOV	-

4 i.MX 95 applications processor camera software architecture

A Linux kernel driver is available for each of the hardware domain modules listed in Figure 1.

These drivers follow the Linux V4L2/media controller framework. For more details, see Section 4.1 "Linux kernel architecture".

To hide the complexity of the subsystem to the end user, the libcamera library has been adopted. It is responsible for setting up and managing the camera stream pipelines from the sensor to the user application, through the camera domain hardware modules. For more details, see Section 4.2 "Libcamera architecture".

The i.MX 95 applications processor camera software architecture has been designed with compliance with modern approaches for managing complex camera subsystems in mind, such as:

- Avoid deviation from the Linux kernel V4L2 API, design, or behavior whenever possible.
- Use of a recognized Linux community library, libcamera.
- Avoid deviation from the libcamera API, design, or behavior.

To achieve the best image quality required for a product, a camera sensor must be tuned and calibrated. A production quality tuning and calibration toolset is provided to perform such a task. This tool connects to the libcamera library to exchange data frames and metadata, enabling effective image processing of the final product.

4.1 Linux kernel architecture

This section describes the drivers for each of the hardware modules involved in the software stack.

This section covers the media controller entities involved in the imaging pipeline. The green boxes, with round corners, represent the V4L2 media subdevices:

- SINK pads (input) are at the top.
- The subdevice name and node path are in the middle.
- SOURCE pads (output) are at the bottom.

Note:

- *The V4L2 API uses the notation of SINK and SOURCE, when referring to pad connection. A SINK pad represents the input pad, relative to the entity. Input pads sink data and are targets of links. A SOURCE pad represents an output pad, relative to the entity. Output pads the source data and are the origins of links. In other words, an entity takes its data from a SINK pad and outputs it on a SOURCE pad.*
- *In addition, when referring to video device nodes, the V4L2 API uses the notation of CAPTURE, OUTPUT, and M2M. CAPTURE video nodes are nodes that output capture data streams from cameras or other sources. OUTPUT video nodes are nodes that send streams to the output of the system. M2M video nodes are nodes that use input streams from the user-space, returning them back to the user-space.*

4.1.1 Camera sensor driver

This section describes the camera sensor driver design, when connected and not connected to the EVK through a GMSL deserializer.

4.1.1.1 V4L2 camera subdevice

Table 2 is the representation of a camera sensor as a Linux kernel media entity.

Table 2. Camera sensor as a Linux kernel media entity

Type	MEDIA_ENT_F_CAM_SENSOR
Flags	V4L2_SUBDEV_FL_HAS_DEVNODE
SINK pads	0
SOURCE pads	1 (num: 0)

Figure 2 shows a simple representation of a camera sensor. The only pad represents the device entry-point for connecting with other subdevices. The example above also includes a serializer on the physical camera board, but for simplicity, this serializer is not included into the V4L2 subsystem representation.

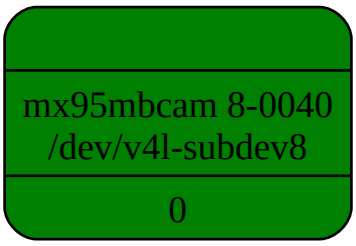

Figure 2. Camera sensor as a Linux kernel media entity

4.1.2 V4L2 GMSL deserializer subdevice

Gigabit multimedia serial link (GMSL) is the line driver and receiver designed for video applications.

An expansion card has been built to support up to four GMSL2 based OX03C10 cameras to the i.MX 95 EVK, through a Maxim MAX96724 GMSL2 deserializer.

The MAX96724 deserializer converts four GMSL2 inputs to 1, 2, or 4 MIPI D-PHY or C-PHY outputs.

Table 3 is the representation of the MAX96724 deserializer as a Linux kernel media entity.

Table 3. Maxim96724 deserializer as a Linux kernel media entity

Type	MEDIA_ENT_F_VID_IF_BRIDGE
Flags	V4L2_SUBDEV_FL_HAS_DEVNODE
SINK pads	4 (num: 0..3)

Table 3. Maxim96724 deserializer as a Linux kernel media entity*...continued*

Type	MEDIA_ENT_F_VID_IF_BRIDGE
SOURCE pads	1 (num: 4)

This device is required to obtain multiple streams from various camera sensors using the CSI protocol. A CSI receiver has only one physical port. However, at protocol level, it can carry up to four streams using Virtual Channels (VCs). Therefore, the analog serializer/deserializer pairs can stream up to four camera streams.

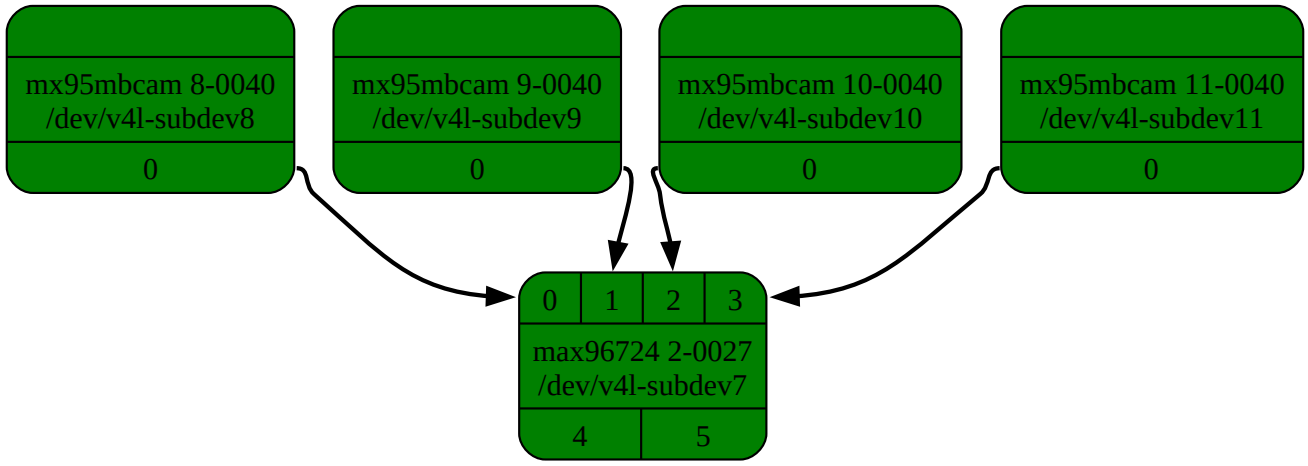

Figure 3. Four OX03C10 and GMSL2 deserializer media topology

The SINK pads connect up to four camera sensor subdevices, while the SOURCE pad connects the deserializer to its corresponding CSI receiver.

4.1.3 MIPI CSI-2 driver

This section describes the i.MX 95 MIPI CSI-2 Linux kernel driver design.

First, it focuses on describing the hardware block. Then, it focuses on the design choices to represent it as a Linux kernel V4L2/media controller driver.

4.1.3.1 MIPI CSI-2 hardware block

Figure 4 shows the hardware (HW) description of the MIPI CSI-2 receiver, as described in *i.MX 95 Applications Processor Reference Manual* (document IMX95RM[3]).

3 Contact a local field applications engineer (FAE) or sales representative.

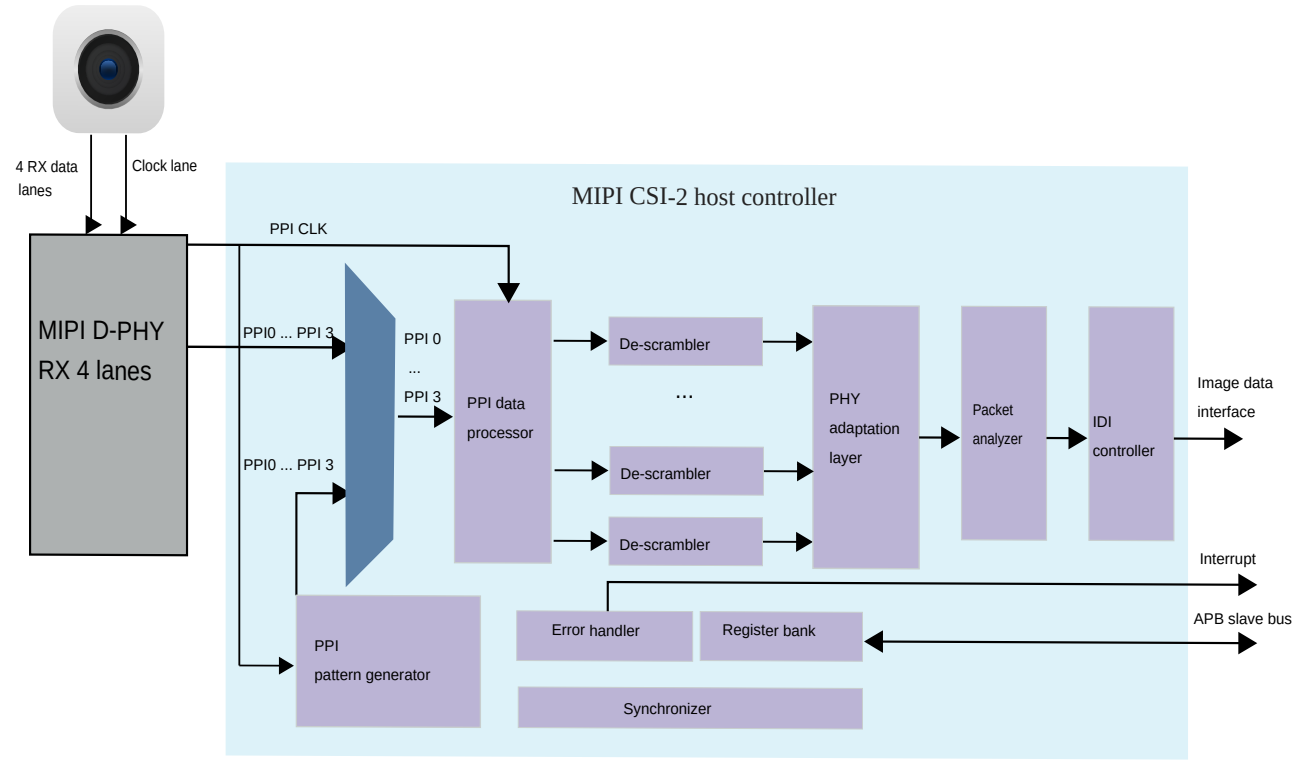

Figure 4. MIPI CSI-2 host controller block diagram

As shown in Figure 5, it outputs the data through the image data interface (IDI) block. After this block, there is another block, called CSI Pixel Formatter. This formatter converts data from the CSI-2 format to the Pixel Link (PL) format. This conversion is necessary because PL is the internal data BUS used to transfer camera pixels through the system.

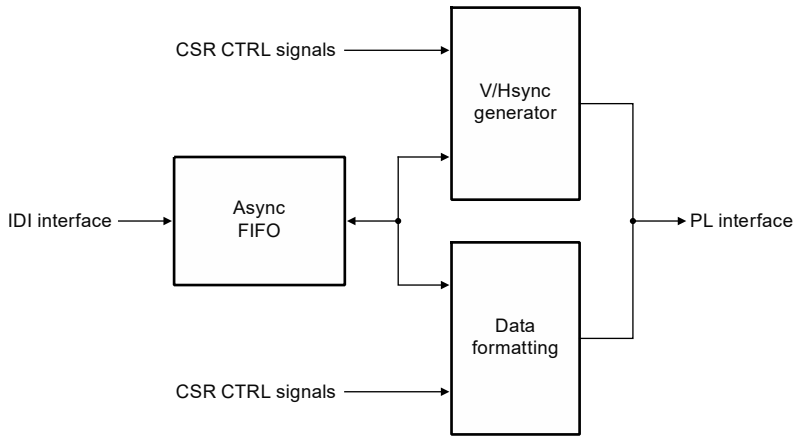

Figure 5. CSI-2 pixel formatting block diagram

To understand the capabilities of the CSI Pixel Formatter better, Figure 6 shows three cameras streaming various data types and formats through a CSI-2 deserializer.

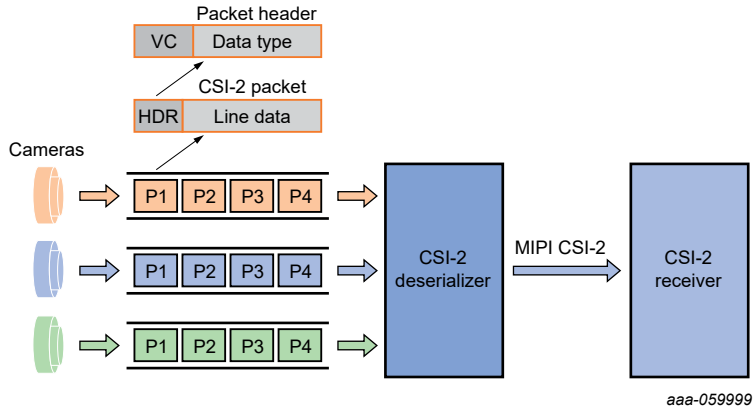

Figure 6. MIPI CSI-2 streaming example

As shown in Figure 7, the camera sends the data through a CSI-2 packet. This packet has a header, which identifies what type of data is carried on that packet. A VC and a data type (DT) help identify the data. All the packets are deserialized on the same CSI-2 port by the deserializer. Once deserialized, the CSI-2 data is processed based on the VC and DT. In summary, the function of the CSI Pixel Formatter is to analyze the VC and DT of a packet. Based on its programming, it can theoretically reroute the packet by altering the VC.

Figure 7 is an example diagram of such rerouting.

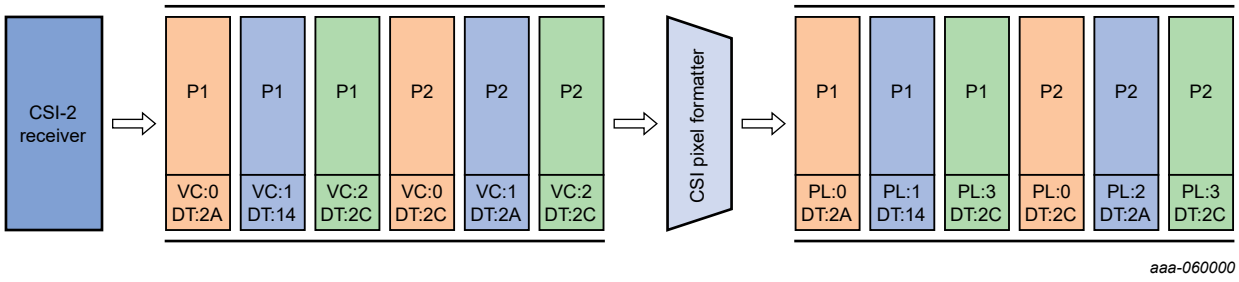

Figure 7. i.MX 95 MIPI CSI Formatter routing example

The diagram in Figure 7 has a new level of packet representation:

- The camera highlighted in orange is sending RAW16 (DT: 0x2E) interleaved with RAW12 (DT: 0x2C) on VC0.
- The camera highlighted in blue is sending embedded data (DT: 0x14 - user-defined data type) and RAW16 (DT: 0x2A) on VC1.
- The camera highlighted in green is sending RAW12 (DT: 0x2C) only on VC2.

The output from the CSI Pixel Formatter shows the effect of the desired routing:

- RAW16 (DT: 0x2A) and RAW12 (DT: 0x2C) on VC0 goes to Pixel Link VC0 (PL0). In this case, the dual-exposure frame remains interleaved on VC0.
- Embedded data (DT: 0x14) on VC1 goes to PL VC1.
- RAW16 (DT: 0x2A) on VC1 goes to PL VC2.
- RAW12 (DT: 0x2C) on VC2 goes to PL VC3.

The example above shows how CSI-2 data can be multiplexed or demultiplexed:

- Multiplexed: Multiple data types from multiple CSI-2 virtual channels can go to the same PL virtual channel.
- Demultiplexed: Multiple data types from the same CSI-2 virtual channel can go to another individual PL virtual channel, demonstrated in the above example.

4.1.3.2 V4L2 MIPI CSI-2 subdevice

This driver manages the HW block that handles the CSI-2 D-PHY and CSI-2 receiver interface in the system.

Table 4. i.MX 95 MIPI CSI-2 Linux kernel media entity

Type	MEDIA_ENT_F_VID_IF_BRIDGE
Flags	V4L2_SUBDEV_FL_HAS_DEVNODE
SINK pads	1 (num: 0)
SOURCE pads	1 (num: 1)

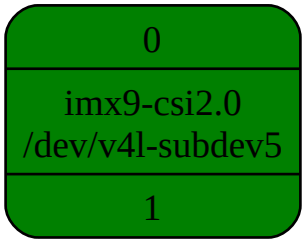

Figure 8. i.MX 95 MIPI CSI-2 Linux kernel media entity

The SINK pad (0) connects to a camera streaming device that is either a camera subdevice or a deserializer bridge. The SOURCE pad (1) connects to the next pixel processor in the system. The VCs identify the camera streams, which are treated as streams in the V4L2 subsystem.

4.1.3.3 V4L2 CSI pixel formatter subdevice

This driver manages the CSI Pixel Formatter block, which is placed after the IDI of the CSI-2 receiver. The CSI-2 pixel formatting module uses packet information, pixel, and non-pixel data from the CSI-2 host controller and reformat them to match PL definition. It can also optionally route non-pixel data or a pixel data type to another PL virtual channel.

Table 5. i.MX 95 MIPI CSI-2 Pixel Formatter Linux kernel media entity

Type	MEDIA_ENT_F_VID_MUX
Flags	V4L2_SUBDEV_FL_HAS_DEVNODE V4L2_SUBDEV_FL_MULTIPLEXED
SINK pads	1 (num: 0)
SOURCE pads	1 (num: 1)

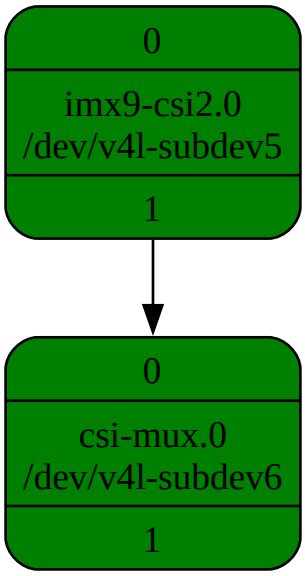

Figure 9. i.MX 95 MIPI CSI-2 subsystem media topology

The csi-mux subdevice multiplexes CSI virtual channels from the CSI-2 output to the PL bus. This device can also split camera streams into multiple streams.

For example:

- Use case 1: CSI-VC0 is routed to PL-VC1
  Here, the csi-mux reroutes a camera stream from one virtual channel to another on the PL bus.
- Use case 2: CSI-VC0 is routed to PL-VC0, PL-VC1, and PL-VC2
  Here, the csi-mux identifies data types in a camera stream, such as embedded-data, long-exposure frames, and short-exposure frames. It then reroutes them to individual virtual channels on the PL.

To achieve the example use cases above, use the V4L2 streams API routing operations.

4.1.3.4 V4L2 CSI-2 pipeline topology

Now that all the devices have been explained, Figure 10 shows the media topology. It features four cameras connected to a deserializer that streams to the CSI-2 receiver.

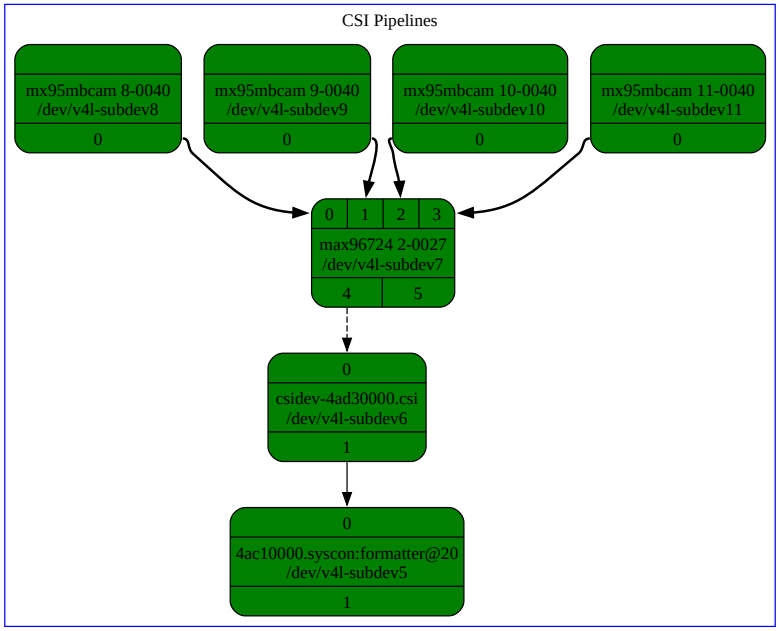

Figure 10. i.MX 95 camera sensor and MIPI CSI-2 subsystem media topology

4.1.4 Image sensing interface driver

This section describes the i.MX 95 image sensing interface (ISI) driver design.

First, it focuses on describing the hardware block. Then, it focuses on the design choices to represent it as a Linux kernel V4L2/media controller driver.

4.1.4.1 i.MX 95 ISI hardware overview

The ISI IP block can process up to 8 pixel streams using eight simultaneous processing channels. Each processing pipeline or channel can be assigned to the same or different pixel input source.

Input sources supported are as follows:

- Two 4-lane MIPI CSI-2 (CSI-2 #0 and CSI-2 #1 PL interface)
- Display controller 0 output (DC #0 PL interface)
- Display controller 1 output (DC #1 PL interface)
- System memory (AXI master; internally converted to PL interface)

Figure 11 shows the HW diagram of the ISI block.

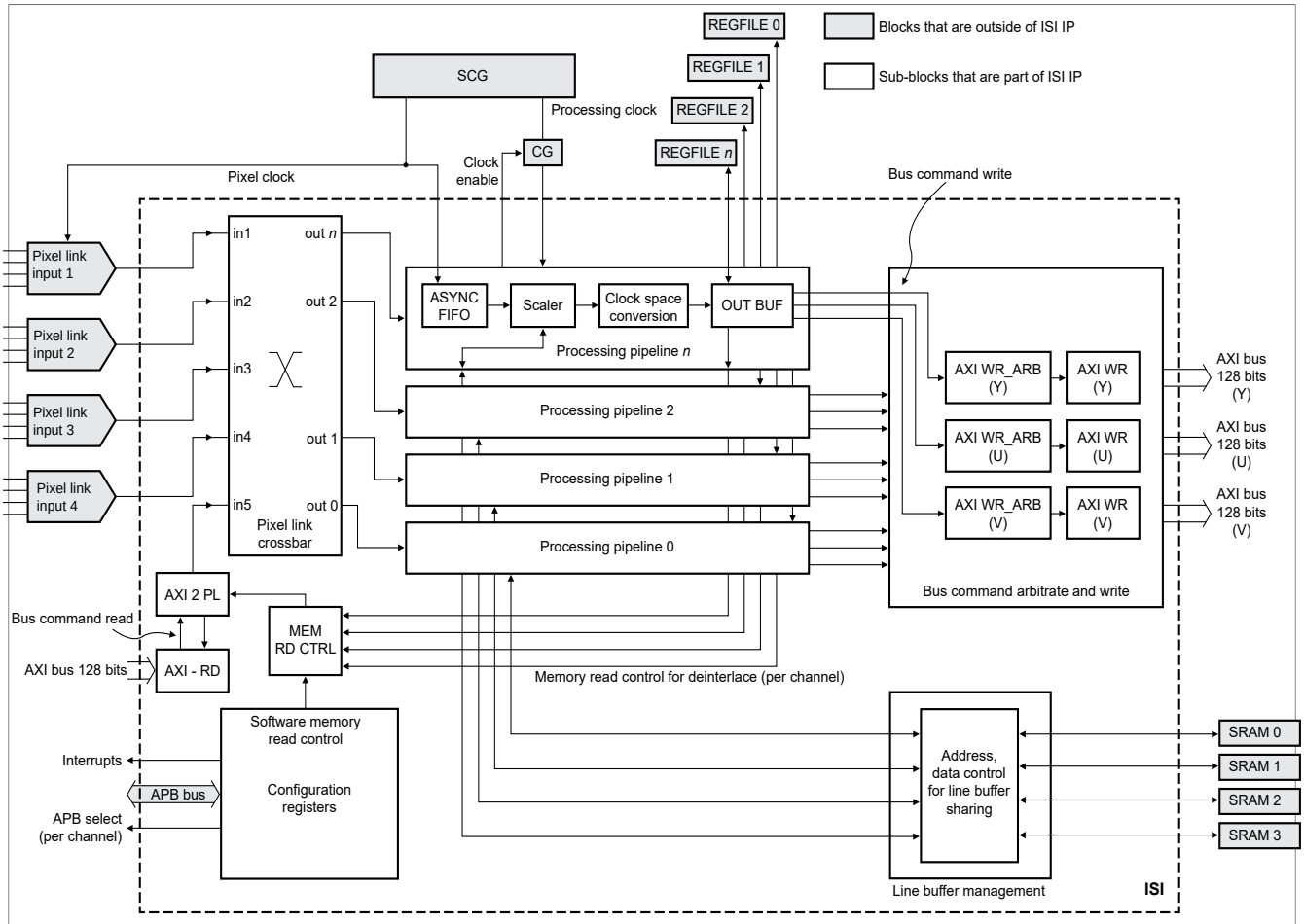

Figure 11. ISI block diagram

The focus is solely on the MIPI CSI-2 and system memory as input sources, because they are of interest for using the imaging subsystem.

4.1.4.2 i.MX 95 V4L2 ISI driver

The ISI driver includes an ISI crossbar subdevice and an ISI pipe subdevice, corresponding to the HW ISI crossbar and ISI processing pipelines.

4.1.4.3 V4L2 ISI crossbar subdevice

The crossbar subdevice is created to allow all the possible connections between input types to an ISI channel. For input pads, the crossbar can be configured with up to eight inputs (one for each processing pipeline). It also has one pad for the system memory input.

Table 6. i.MX 95 ISI crossbar Linux kernel media entity

Type	MEDIA_ENT_F_VID_MUX
Flags	V4L2_SUBDEV_FL_HAS_DEVNODE V4L2_SUBDEV_FL_MULTIPLEXED
SINK pads	5 (num: 0..4)
SOURCE pads	8 (num: 5..12)

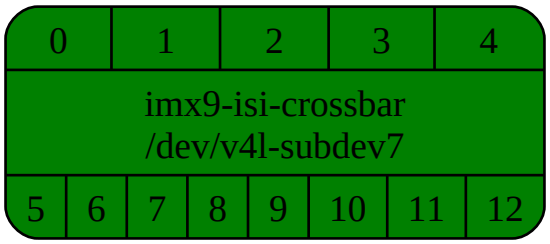

Figure 12. i.MX 95 MIPI ISI crossbar Linux kernel media entity

The SINK/SOURCE pads are used to interpret the HW representation of the ISI crossbar, as defined in the ISI schematics.

Note: *The crossbar is a multiplexer of five inputs and eight outputs. It is possible to configure each output port to connect to any of the five inputs. The five inputs are numbered from 0 to 4, where input 5 is always a memory location. A single input can be assigned to multiple outputs on the crossbar.*

The ISI block inputs ports connect i.MX95 camera domain physical links to the ISI internal pixel link crossbar, which are in the following order:

- Pixel Link input 0, connected to display controller (DC) #0
- Pixel Link input 1, connected to DC #1
- Pixel Link input 2, connected to MIPI CSI-2 controller #0
- Pixel Link input 3, connected to MIPI CSI-2 controller #1
- Memory input, connected to AXI bus

Note: *The CSI type inputs can carry multiple streams using virtual channels. These streams must be routed to individual crossbar outputs, so that one ISI pipe can process each stream.*

4.1.4.4 V4L2 ISI pipe subdevice

The ISI pipe subdevice represents a physical ISI processing channel and it gets connected to an output of the ISI crossbar.

Table 7. i.MX 95 ISI pipe Linux kernel media entity

Function	MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER
Capabilities	V4L2_SUBDEV_FL_HAS_DEVNODE
SINK pads	1 (num: 0)
SOURCE pads	1 (num: 1)

Note: *The output gets connected automatically to an ISI video device node.*

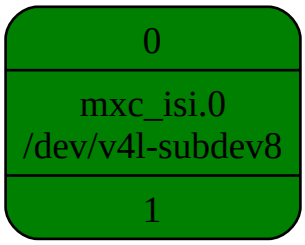

Figure 13. i.MX 95 MIPI ISI pipe Linux kernel media entity

4.1.4.5 V4L2 ISI video node

The video subdevice creates V4L2 video device nodes and manages the CAPTURE buffer queue through it.

Table 8. i.MX 95 ISI video node Linux kernel media entity

Function	MEDIA_ENT_F_PROC_VIDEO_SCALER
Capabilities	V4L2_CAP_STREAMING V4L2_CAP_VIDEO_CAPTURE_MPLANE V4L2_CAP_IO_MC
SINK pads	1 (num: 0)

4.1.4.6 V4L2 ISI memory-to-memory node

The memory-to-memory (M2M) subdevices creates V4L2 video device nodes and manages CAPTURE and OUTPUT buffer queues through it. This subdevice is automatically added to crossbar SINK pad 4 (memory input).

Table 9. i.MX 95 ISI M2M node Linux kernel media entity

Function	MEDIA_ENT_F_PROC_VIDEO_SCALER
Capabilities	V4L2_CAP_STREAMING V4L2_CAP_VIDEO_M2M_MPLANE
SOURCE pads	1 (num: 0)

4.1.4.7 V4L2 ISI pipeline topology

Now, that all the devices are explained, Figure 14 represents the media topology, with four cameras connected to a deserializer, streaming to ISI pipeline channels. It represents the physical connection but the streaming configuration and routing to ISI channels must be done in the user space.

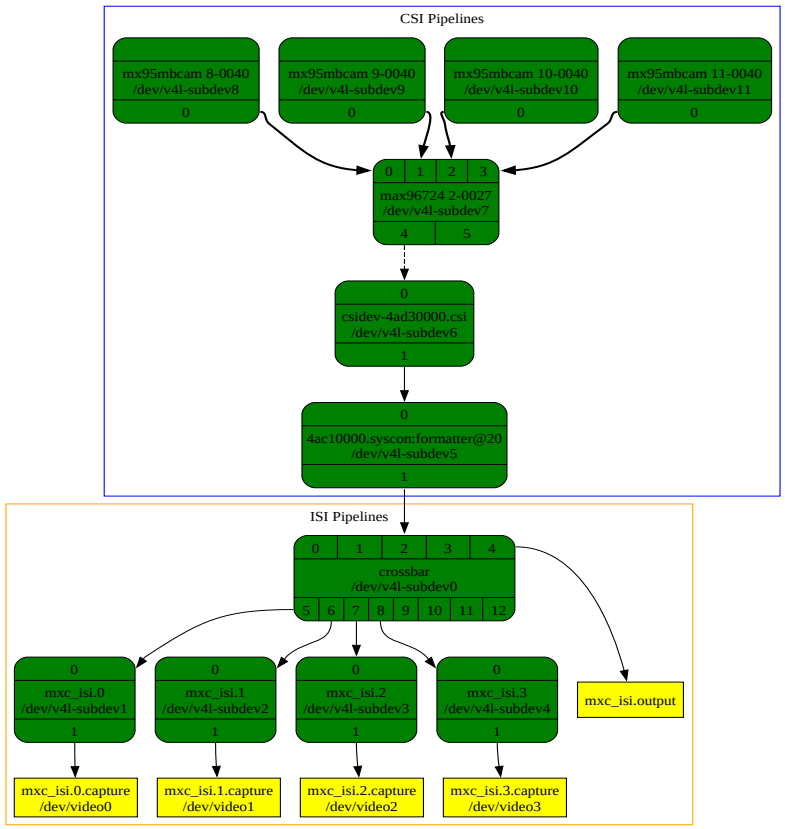

Figure 14. i.MX 95 camera subsystem media topology from camera sensors to ISI video nodes

4.1.4.8 V4L2 ISI camera scenarios

Lets consider a few examples to expose some possible scenarios involving the ISI pipelines.

4.1.4.8.1 Single YUV camera streaming on CSI0:VC0

In this case, the user space:

- Enables and configures the link between csi-mux and isi-crossbar
- Sets the routing in isi-crossbar to route the [input:2, stream:0] from SINK pad 0 to channel 0 on SOURCE pad 5
- From this point, frame buffers can be obtained through the video node of mxc_isi.0

4.1.4.8.2 Two YUV/RGB cameras streaming on CSI0:VC0, VC1

In this case, the user space:

- Enables and configures the link between csi-mux and isi-crossbar
- Sets the routing in isi-crossbar to route the [input:2, stream:0] from SINK pad 0 to channel 0 on SOURCE pad 5
- Sets the routing in isi-crossbar to route the [input:2, stream:1] from SINK pad 0 to channel 1 on SOURCE pad 6
- From this point, frame buffers can be obtained through the video nodes of mxc_isi.0 and mxc_isi.1

4.1.4.8.3 Single RAW camera streaming single-exposure on VC0

The RAW data is RAW12.

In this case, the user space:

- Enables and configures the link between csi-mux and isi-crossbar
- Configures the csi-mux to enable data monitors on VC0 for RAW12
- Sets the routing on csi-mux to route [input:0, stream:0] (CSI:VC0) to [output:0, stream:0] (PL:VC0, Pass-through mode)
- Sets the routing in isi-crossbar to route the [input:2, stream:0] from SINK pad 2 to channel 0 on SOURCE pad 5 (mxc_isi.0)
- Configures the mxc_isi.0 to route RAW12
  Note: *Frame size is established based on the number of lines sent by the sensor*

4.1.4.8.4 Single RAW camera streaming dual-exposure and metadata on same VC

The dual-exposure is represented as:

- Data type 1: RAW16
- Data type 2: RAW12

The metadata is RAW8, a custom user data type. All the data is streamed in CSI:VC0.

In this case, the user space:

- Enables and configures the link between csi-mux and isi-crossbar
- Configures the csi-mux to enable data monitors on VC0 for embedded data, RAW16, and RAW12
- Configures the csi-mux to reroute embedded data to VC0, RAW16 to VC1, and RAW12 to VC2
- Sets the routing on csi-mux to route [input:0, stream:0] (CSI:VC0) to [output:0, stream:0,1,2] (PL:VC0, VC1, VC2; stream demultiplexing)
- Sets the routing in isi-crossbar to route the [input:2, stream:0] from SINK pad 2 to channel 0, on SOURCE pad 5 (mxc_isi.0)
- Sets the routing in isi-crossbar to route the [input:2, stream:1] from SINK pad 2 to channel 1, on SOURCE pad 6 (mxc_isi.1)
- Sets the routing in isi-crossbar to route the [input:2, stream:2] from SINK pad 2 to channel 2, on SOURCE pad 7 (mxc_isi.2)
- Configures the mxc_isi.0 to process RAW8
  Note: *Frame size is established based on the number of lines sent by the sensor*
- Configures the mxc_isi.1 to process RAW16
- Configures the mxc_isi.2 to process RAW12

This method demultiplexed an interleaved dual-exposure stream with embedded data sent across the same MIPI-CSI VC. This method simplifies the process by separating each data type sent by the camera sensor, which allows independent storage in memory. However, a drawback is that for each camera supporting HDR and embedded data, the three ISI channels are already consumed. As a result, it is evident that too many HDR cameras cannot be supported simultaneously. The processing of RAW frames involves an ISP block; for more details on processing of RAW frames using ISP, see Section 4.1.5 "Image signal processing driver".

4.1.4.8.5 Single RAW camera streaming dual-exposure and metadata on several VCs

The dual-exposure is represented as:

- Data type 1: RAW16 sent on VC0
- Data type 2: RAW12 sent on VC1

The metadata is RAW8, a custom user data type sent on VC0 as pre-pixel and post-pixel data.

In this case, the user-space:

- Enables and configures the link between csi-mux and isi-crossbar
- Configures the csi-mux to enable data monitors on VC0 for embedded data and RAW16, and on VC1 for RAW12
- Configures the csi-mux to reroute embedded data to VC0, RAW16 to VC0, and RAW12 to VC0
- Sets the routing on csi-mux to route [input:2, stream:0, 1] (CSI:VC0, VC1) to [output:0, stream:0] (PL:VC0; stream multiplexing)
- Sets the routing in isi-crossbar to route the [input:2, stream:0] from SINK pad 2 to channel 0 on SOURCE pad 5 (mxc_isi.0)
- Configures the mxc_isi.0 to process RAW8
  Note: *Frame size is established based on the number of lines sent by the sensor*

This use case is a complex theoretical one that implies interleaving multiple data types into the same stream. It is essential to ensure that the order of the incoming data is correct. Otherwise, it is impossible to determine which line corresponds to which data type. In this case, the frame looks as follows:

```
+-----------------------------------------------+
| Line 0  : Meta-data front line 1              |
| Line 1  : Meta-data front line 2              |
| Line 2  : Data Type 1 (RAW16), Image line 0   |
| Line 3  : Data Type 2 (RAW12), Image line 0   |
| Line 4  : Data Type 1 (RAW16), Image line 1   |
| Line 5  : Data Type 2 (RAW12), Image line 1   |
|         :     ...                             |
| Line n-2: Data Type 1 (RAW16), Image line h-1 |
| Line n-1: Data Type 2 (RAW12), Image line h   |
| Line n  : Meta-data back line 1               |
+-----------------------------------------------+
```

Ensure proper handling of this structure and process the frame buffer accordingly. This method is useful when dealing with multiple HDR cameras, especially when ISI channels are insufficient to process each data type individually.

4.1.5 Image signal processing driver

This section describes the i.MX 95 image signal processing (ISP) driver design.

It first focuses on describing the hardware block. Then focuses on the design choices to represent it as a Linux kernel V4L2/media controller driver.

4.1.5.1 ISP IP block

ISP IP Block (NEO-ISP) performs a set of image-processing tasks on the RAW camera stream. The input camera stream and the NEO-ISP processed output image are stored in DDR or another sufficiently fast system memory to keep up with NEO-ISP processing.

The NEO-ISP operation is based on frames, where one complete image frame is read and output pixel by pixel, line by line. The raw pixel processing pipeline can handle up to 500 megapixels per second.

In M2M mode, it processes multiple camera streams in a time-multiplexed manner by switching the ISP contexts, including register and internal memory content. For example, 2x8MP@30fps, 8x2MP@30fps, or other sensor combinations.

4.1.5.2 V4L2 ISP topology

The ISP subdevice is a standalone media device acting like a STREAMING or M2M device.

4.1.5.2.1 M2M mode

Figure 15 shows the ISP device nodes topology.

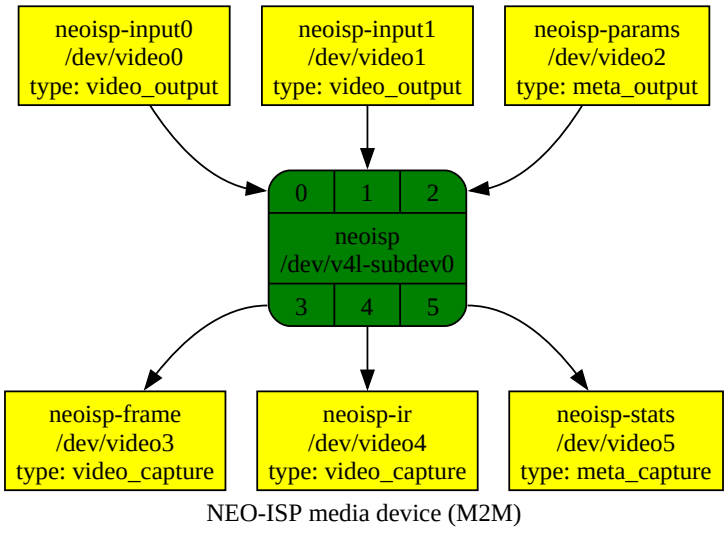

Figure 15. i.MX 95 ISP Linux kernel media entity in M2M mode

In M2M mode, the device takes its data from the special M2M video node.

Table 10. Description of i.MX 95 ISP Linux kernel media entity in M2M mode

Function	MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER
Capabilities	V4L2_SUBDEV_FL_HAS_DEVNODE
SINK pads	3 (num: 0..2)
SOURCE pads	3 (num: 3..5)

The pads are used as follows:

- Pad 0 (input) to connect to ISP "neoisp-input0" output device for RAW frames to be submitted to the ISP for processing
- Pad 1 (input) to connect to ISP "neoisp-input1" output device for RAW frames short capture in HDR mode
- Pad 2 (input) to connect to ISP "neoisp-params" output meta device for parameters provided by user space 3A algorithms
- Pad 3 (output) to connect to ISP "neoisp-frame" capture device for processed images
- Pad 4 (output) to connect to ISP "neoisp-ir" capture device for infra-red (IR) image
- Pad 5 (output) to connect to ISP "neoisp-stats" capture meta device for generated image statistics for user space 3A algorithms

neoisp_input0, neoisp_input1

Images processed by NEO-ISP are queued to the neoisp-input0 and neoisp-input1 (if HDR mode is enabled) output device nodes. NEO-ISP inputs are limited in width and height to be multiple of 16 and 2 respectively.

neoisp_params

The neoisp_params output meta device receives configuration data that is written to NEO-ISP registers and internal memory for the desired input image processing. This V4L2 device only accepts the V4L2_META_FMT_NEO_ISP_PARAMS format.

neoisp_frame

The capture device writes the processed image by NEO-ISP to memory.

neoisp_ir

For infrared pixels camera sensors, this capture device provides infrared pixels image processed by NEO-ISP.

neoisp_stats

The neoisp_stats capture meta device provides statistic data generated by NEO-ISP hardware while processing the input image. This V4L2 device accepts only V4L2_META_FMT_NEO_ISP_STATS format.

The ISP can process up to eight pixel streams sequentially. Therefore, eight instances of the ISP subdevice are created with each instance managing a one-pixel stream, as shown in Figure 15. When a video device is opened, the corresponding ISP instance is initialized to handle both video buffers and context buffers. A dedicated driver called "neoisp" manages the internal operations of the ISP. It handles tasks such as resetting the ISP, managing streaming nodes, performing IOCTLs, and more.

4.1.5.2.2 Streaming mode

The device is connected directly to a CSI-specific subdevice without going through the ISI.

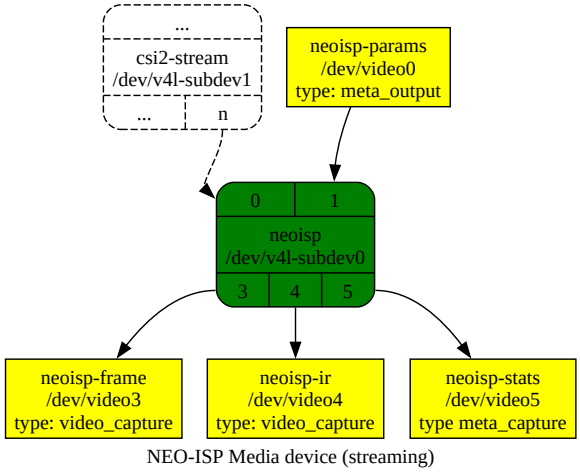

Figure 16. Representation of i.MX 95 ISP Linux kernel media entity in Streaming mode

Note: *This mode is not supported in NXP Linux BSP for i.MX 95 yet.*

Now, that all the pieces are in place, see Figure 17 for an overview of all the involved nodes.

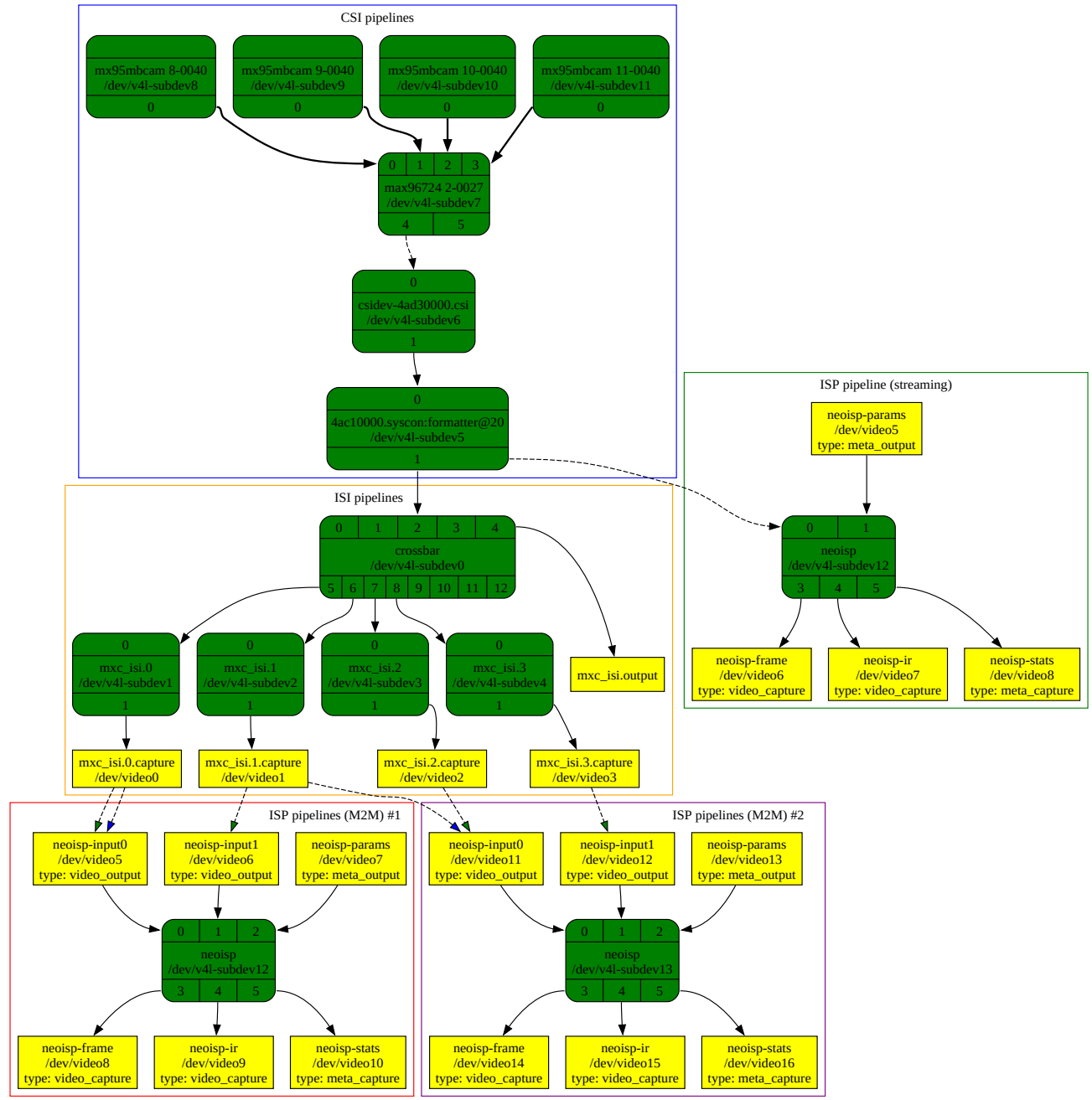

Figure 17. i.MX 95 camera subsystem media topology with four camera sensors

Note: *The link connections between the CSI multiplexer, ISI crossbar, and ISP are dashed because these links exist but are disabled. The user enables these links based on its needs.*

This topology shows only two instances of M2M ISP media entity for the sake of readability. The green sub-graph represents the STREAMING mode ISP (not supported yet). Both M2M and STREAMING modes cannot coexist, as shown in Figure 18.

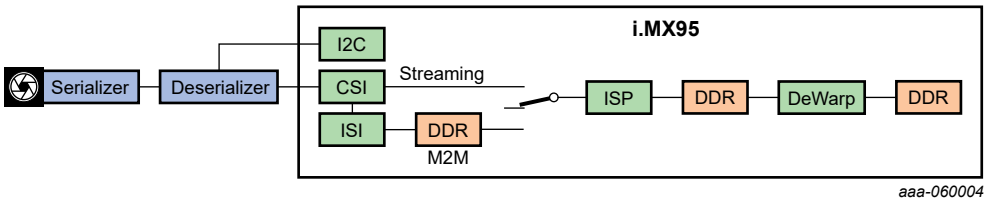

Figure 18. Logical representation of i.MX 95 camera capture stream

The links between ISI capture (0 and 1) nodes and ISP output (input0 and input1) nodes depend on the input type:

- If the input is either single exposure or multiple exposures combined in the sensor, then the input1 pad is not linked.
- If dual exposure input is used, then both input0 and input1 are used.

The blue dashed links depict the first input type listed above.

4.1.5.3 V4L2 ISP driver formats and events

This section lists the Linux kernel V4L2 video outputs and metadata formats supported by the driver.

4.1.5.3.1 Meta formats

Table 11. List of ISP driver metadata supported format

V4L2 pixel format	Description
V4L2_META_FMT_NEO_ISP_PARAMS	NXP NEO-ISP 3A parameters (new format)
V4L2_META_FMT_NEO_ISP_STATS	NXP NEO-ISP 3A statistics (new format)

4.1.5.3.2 Video output formats

Table 12. List of ISP driver supported video output formats

V4L2 pixel format	Description
V4L2_PIX_FMT_SRGGB8	8-bit Bayer RGRG/GBGB
V4L2_PIX_FMT_SBGGR8	8-bit Bayer BGBG/RGRG
V4L2_PIX_FMT_SGBRG8	8-bit Bayer GBGB/RGRG
V4L2_PIX_FMT_SGRBG8	8-bit Bayer GRGR/BGBG
V4L2_PIX_FMT_SRGGB10	10-bit Bayer RGRG/GBGB
V4L2_PIX_FMT_SBGGR10	10-bit Bayer BGBG/RGRG
V4L2_PIX_FMT_SGBRG10	10-bit Bayer GBGB/RGRG
V4L2_PIX_FMT_SGRBG10	10-bit Bayer GRGR/BGBG
V4L2_PIX_FMT_SRGGB12	12-bit Bayer RGRG/GBGB
V4L2_PIX_FMT_SBGGR12	12-bit Bayer BGBG/RGRG
V4L2_PIX_FMT_SGBRG12	12-bit Bayer GBGB/RGRG
V4L2_PIX_FMT_SGRBG12	12-bit Bayer GRGR/BGBG
V4L2_PIX_FMT_SRGGB14	14-bit Bayer RGRG/GBGB
V4L2_PIX_FMT_SBGGR14	14-bit Bayer BGBG/RGRG

Table 12. List of ISP driver supported video output formats*...continued*

V4L2 pixel format	Description
V4L2_PIX_FMT_SGBRG14	14-bit Bayer GBGB/RGRG
V4L2_PIX_FMT_SGRBG14	14-bit Bayer GRGR/BGBG
V4L2_PIX_FMT_SRGGB16	16-bit Bayer RGRG/GBGB
V4L2_PIX_FMT_SBGGR16	16-bit Bayer BGBG/RGRG
V4L2_PIX_FMT_SGBRG16	16-bit Bayer GBGB/RGRG
V4L2_PIX_FMT_SGRBG16	16-bit Bayer GRGR/BGBG
V4L2_PIX_FMT_GREY	8-bit Greyscale
V4L2_PIX_FMT_Y10	10-bit Greyscale
V4L2_PIX_FMT_Y12	12-bit Greyscale
V4L2_PIX_FMT_Y14	14-bit Greyscale
V4L2_PIX_FMT_Y16	16-bit Greyscale

4.1.5.3.3 Video capture formats

Table 13. List of ISP driver supported video capture formats

V4L2 pixel format	Description
V4L2_PIX_FMT_BGR24	24-bit BGR 8-8-8
V4L2_PIX_FMT_RGB24	24-bit RGB 8-8-8
V4L2_PIX_FMT_BGRX32	32-bit XBGR 8-8-8-8
V4L2_PIX_FMT_RGBX32	32-bit RGBX 8-8-8-8
V4L2_PIX_FMT_NV12	12-bit Y/CbCr 4:2:0
V4L2_PIX_FMT_NV21	12-bit Y/CrCb 4:2:0
V4L2_PIX_FMT_NV16	16-bit Y/CbCr 4:2:2
V4L2_PIX_FMT_NV61	16-bit Y/CrCb 4:2:2
V4L2_PIX_FMT_YUYV	16-bit YUYV 4:2:2
V4L2_PIX_FMT_UYVY	16-bit UYVY 4:2:2
V4L2_PIX_FMT_VYUY	16-bit VYUY 4:2:2
V4L2_PIX_FMT_YUV24	24-bit YUV 4:4:4 8-8-8
V4L2_PIX_FMT_YUVX32	32-bit YUVX 8-8-8-8
V4L2_PIX_FMT_VUYX32	32-bit VUYX 8-8-8-8
V4L2_PIX_FMT_GREY	8-bit Greyscale
V4L2_PIX_FMT_Y10	10-bit Greyscale
V4L2_PIX_FMT_Y12	12-bit Greyscale
V4L2_PIX_FMT_Y16	16-bit Greyscale
V4L2_PIX_FMT_Y16_BE	16-bit big-endian Greyscale

All formats marked as new format have been added to the Linux kernel's common V4L2 API.

4.1.5.3.4 V4L2 supported events

Table 14. List of ISP driver supported V4L2 events

V4L2 event type	Description
V4L2_EVENT_FRAME_SYNC	Frame start notification

4.1.5.4 V4L2 ISP camera scenarios

The i.MX 95 camera subsystem is flexible enough to support a significant number of routing and camera combinations scenarios.

This section focuses on some of the most common routing and camera combinations scenarios.

4.1.5.4.1 Single RAW camera, streaming on CSI0: VC0

In this use case, the camera stream goes through the ISI, which represents the M2M mode. As the STREAMING mode is not implemented currently, this path is presented.

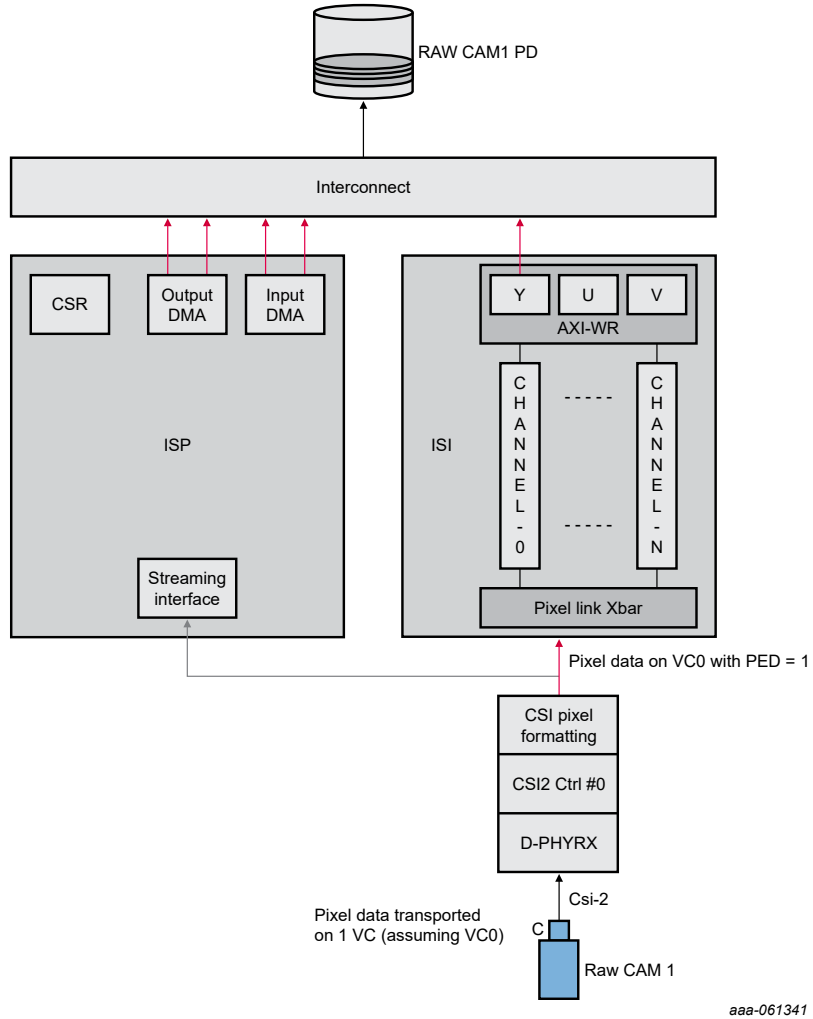

Figure 19. Single RAW camera capture: i.MX 95 hardware routing options

4.1.5.4.2 Three RAW cameras streaming on CSI0: VC0, VC1, VC2

Figure 20 shows that the path to ISP (STREAMING mode) is not enabled currently. Therefore, all camera streams route through ISI.

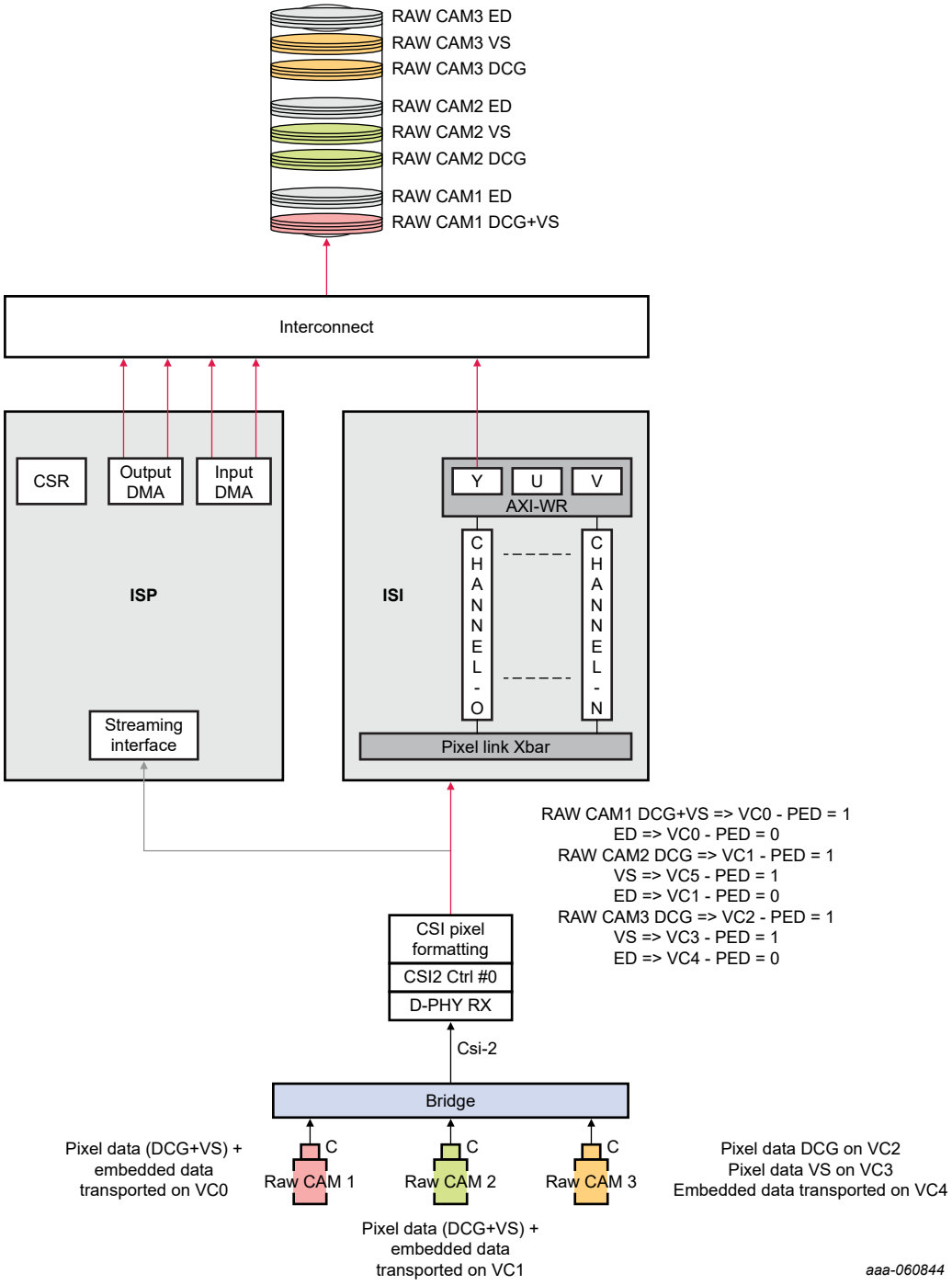

Figure 20. Three RAW camera capture data stream in the i.MX 95 camera hardware subsystem

Similar to the steps outlined in the Section 4.1.4.8.4 "Single RAW camera streaming dual-exposure and metadata on same VC", the user space must follow the same steps for all camera streams.

Lets assume that:

- Camera 1 is streaming metadata and interleaved dual-exposure frames on VC0
- Camera 2 is streaming metadata and interleaved dual-exposure frames on VC1
- Camera 3 is streaming DCG (RAW16) pixel data on VC2, VS (RAW12) pixel data on VC3 and metadata on VC4

In this case, the user space:

- Enables the link between csi-mux and isi-crossbar
- Configures the csi-mux to enable data monitors on VC0 for embedded data, RAW16, and RAW12
- Configures the csi-mux to enable data monitors on VC1 for embedded data, RAW16, and RAW12
- Configures the csi-mux to enable data monitors on VC2 for RAW16
- Configures the csi-mux to enable data monitors on VC3 for RAW12
- Configures the csi-mux to enable data monitors on VC4 for embedded data
- Configures the csi-mux to reroute VC0 embedded data, RAW16, and RAW12 to PL:VC0 (stream pass through)
- Configures the csi-mux to reroute VC1 embedded data, RAW16 to PL:VC1 (substream pass through), and RAW12 to PL:VC5 (substream reroute)
- Configures the csi-mux to reroute VC2 RAW16 to PL:VC2 (substream pass through), VC3 RAW12 to PL:VC3 (substream passthrough) and VC4 embedded data to PL:VC4 (substream pass through)
  At this point, we have streams on six virtual channels: VC0-5
- Sets the routing in isi-crossbar to route the [input:0, stream:0] from SINK pad 0 to channel 0, on SOURCE pad 5 (mxc_isi.0)
- Sets the routing in isi-crossbar to route the [input:0, stream:1] from SINK pad 0 to channel 1, on SOURCE pad 6 (mxc_isi.1)
- Sets the routing in isi-crossbar to route the [input:0, stream:5] from SINK pad 0 to channel 5, on SOURCE pad 10 (mxc_isi.5)
- Sets the routing in isi-crossbar to route the [input:0, stream:2] from SINK pad 0 to channel 2, on SOURCE pad 7 (mxc_isi.2)
- Sets the routing in isi-crossbar to route the [input:0, stream:3] from SINK pad 0 to channel 3, on SOURCE pad 8 (mxc_isi.3)
- Sets the routing in isi-crossbar to route the [input:0, stream:4] from SINK pad 0 to channel 4, on SOURCE pad 9 (mxc_isi.4)

Clearly, the pipeline is complex enough to cause understanding difficulties, highlighting why the use of libcamera greatly assists the camera application developers.

4.1.5.5 V4L2 ISP UAPI interface

User space API (UAPI) is the Linux kernel interface to user space applications. The UAPI simplifies the complex interdependencies between headers that are partly exported to user space. Also, it simplifies and reduces the size of the kernel-only headers. It helps to track changes to the APIs that the kernel presents to user space.

Moreover, a new UAPI definition must respect the following aspects:

- Portability across different architectures.
- Memory management abstraction such as cache, barriers, and byte and bit order.
- Abstraction of hardware physical address map.
- The same interface must be able to handle multiple device revisions/evolutions (stability).
- Compliant to Linux community coding rules and common usage.

ISP UAPI header file defines metadata structures that are shared between ISP and 3A algorithms operating in the user space. Metadata consists of statistics generated by the ISP hardware and configuration parameters fine-tuned by 3A algorithms to be applied to ISP hardware. This UAPI header file must define both metadata types.

In addition to the metadata, this UAPI introduces a "feature update flag" structure designed to minimize the number of configuration parameters sent to the ISP. This is achieved by setting an "update" flag for each ISP sub-block. For any given ISP feature block, the ISP driver updates its parameters only if the corresponding "update" flag is set by the 3A algorithms.

4.1.5.5.1 Metadata flow

ISP metadata includes configuration parameters and statistical data generated by different hardware sub-blocks.

This metadata shared between the ISP hardware block and the 3A algorithms libraries interface (in user space):

- From ISP to 3A: Only statistics are provided where all buffers are copied.
- From 3A to ISP: Only parameters are involved and not all buffers are copied; only the parameters with the "update flag" set.

In the worst-case scenario, all parameters can be updated, but typically only some are adjusted in real-world applications. Update flags are placed at the beginning of the parameter structures, with each feature block of the ISP pipeline having its own dedicated flag.

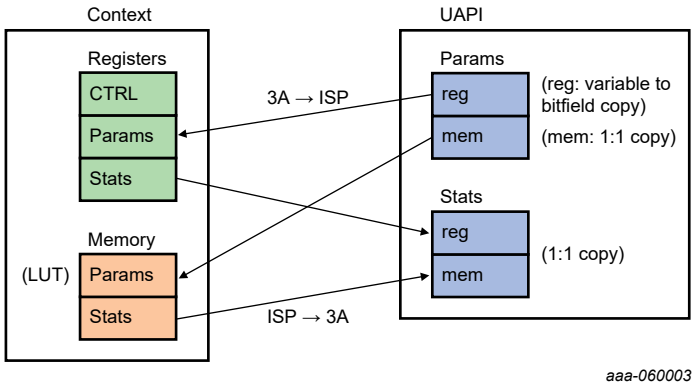

Figure 21. V4L2 ISP driver internal metadata to UAPI conversion

The context is the concatenation of ISP registers and internal memory snapshots. It is saved under a UAPI structure, which saves and restores the ISP context without any data handling.

In multi-context use case (for example, multiple cameras), each camera has a dedicated context structure stored in DDR (saved context).

A dashed arrow for parameters means that not all structures contents are copied; only the ones with "update flag" enabled are transferred to the ISP.

4.2 Libcamera architecture

For libcamera architecture, see the *i.MX Linux Users Guide* (document UG10163).

4.2.1 Libcamera overview

Libcamera is an open source camera stack and framework developed by the Linux media community in collaboration with the industry.

- It is a user space library that relies on the existing Linux kernel drivers and API.
- It aims to abstract the complexity of the camera subsystem and provides the users with a unified and intuitive interface for the camera operation.
- It supports single and multi-camera use cases. Supported cameras can be either a camera sensor typically connected through a MIPI CSI bus, or a USB camera exposing a UVC class.

Native libcamera applications make direct use of the interfaces exported by the library. The framework supports a GStreamer adapter; it implements a GStreamer plugin available to applications to use as a source element for a GStreamer pipeline.

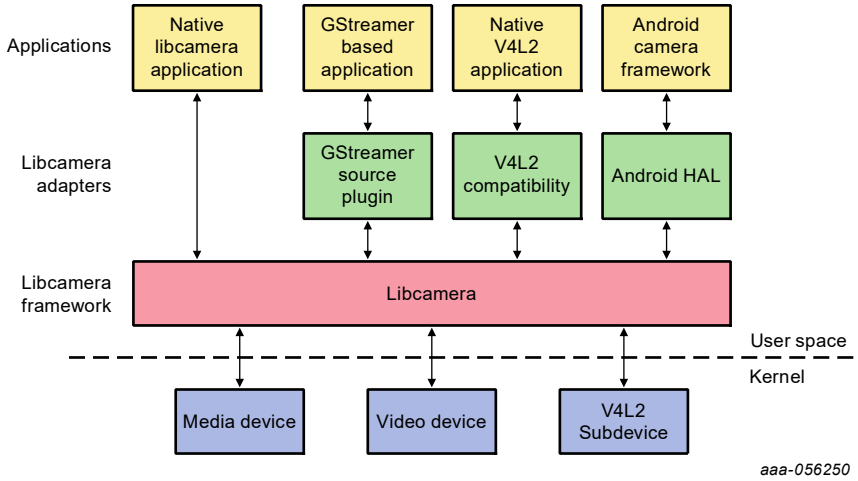

Figure 22. Libcamera high-level architecture overview

4.2.2 Libcamera i.MX 95 support overview

The i.MX 95 supports both raw and smart cameras:

- Raw cameras require the use of NEO-ISP, to decode and post process the raw image output by the sensor. In the libcamera framework, a specific ISP is supported through the dedicated NXP NEO-ISP pipeline handler and its associated image-processing algorithm (IPA).
- Smart cameras provide a decoded image in various formats. In the libcamera framework, the NXP ISI pipeline handler aims to handle such cameras.

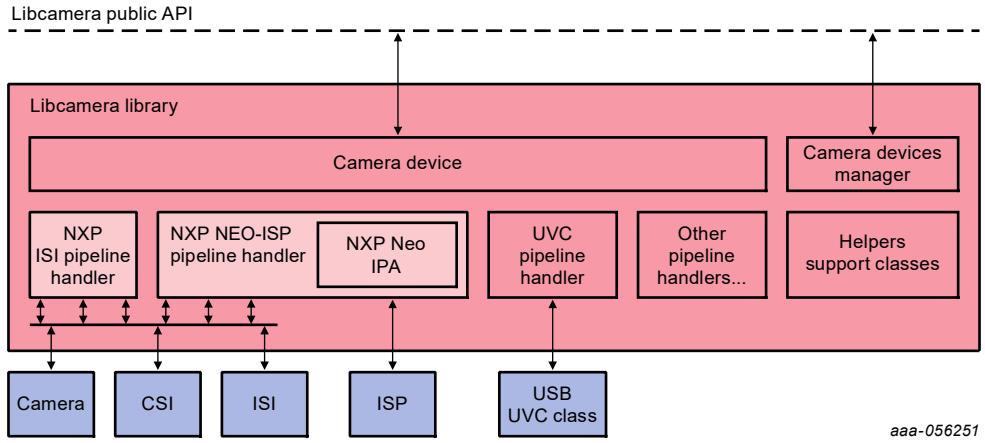

Figure 23. i.MX 95 libcamera architecture overview

The pipeline handlers implement the platform-specific handling of the media devices from the hardware pipeline:

- For the NEO-ISP pipeline case, it corresponds to the camera, CSI, ISI, and ISP devices.
- For the ISI pipeline case, it corresponds to the camera, CSI, and ISI devices.

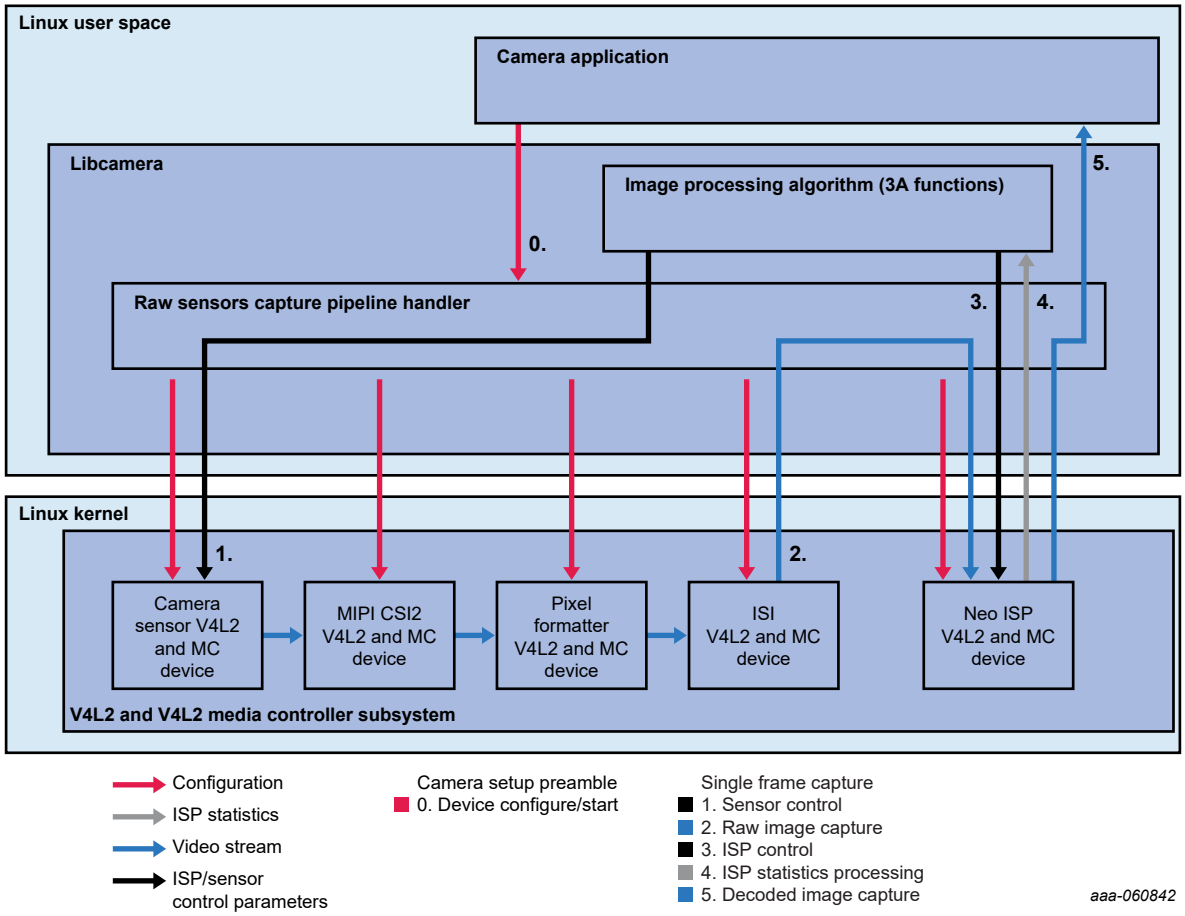

Figure 24. Pipeline handler for RAW camera sensors

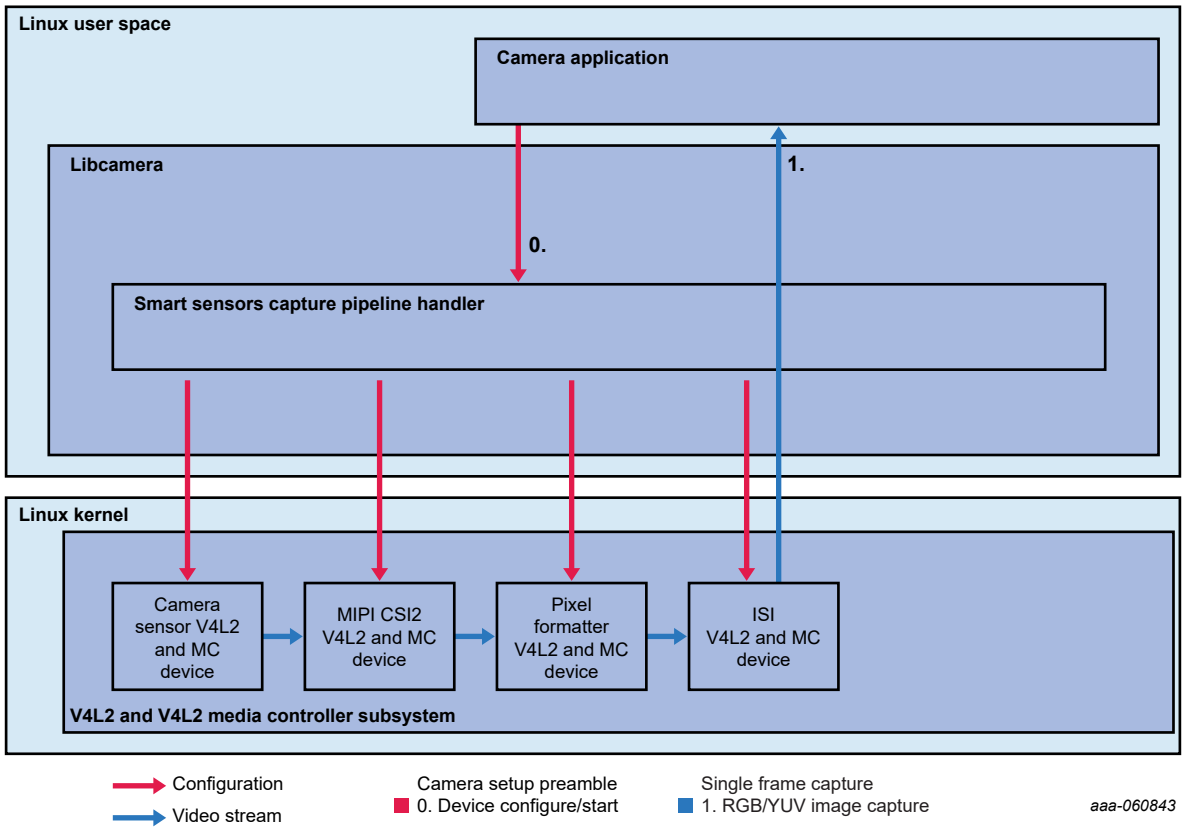

Figure 25. Pipeline handler for smart camera sensors

The pipeline handlers are responsible for the enumeration and configuration of those devices and circulating the image and metadata buffers between the hardware and software parties.

The IPA is the component implementing the camera control algorithms (white balance, auto exposure control, and so on). Algorithms from the IPA process on a camera frame based on the statistics metatdata output by the ISP. They compute and reconfigure the necessary parameters in real time to optimize the ISP and camera operation.

UVC camera support is standard. Its pipeline handler comes with the libcamera framework.

4.2.3 NEO-ISP image-processing algorithm

NEO-ISP libcamera pipeline is used when its pipeline handler is configured as the matching pipeline for the camera enumeration procedure. See section "libcamera configuration" in the *i.MX Linux Users Guide* (document UG10163). During its operation, the NEO-ISP pipeline handler interacts with an IPA relevant to that specific pipeline and ISP. Such an IPA module embeds the 3A control algorithms in charge of the real-time programming of the ISP and the sensor configuration.

In the libcamera framework, an IPA module is a shared library and can be built in-tree (open source).

It can be bound to the pipeline handler and executed in the same process, but in a different thread from libcamera. Alternatively, an IPA implementation can be located out-of-tree giving the option to be licensed as a closed-source. In that case, IPA can be bound to the pipeline handler but runs in isolation in a separate process. In both cases, the pipeline handler and IPA interwork through the same interprocessor communication (IPC) interface. For more details, see libcamera IPA Writer's Guide.

Two IPAs are delivered in the i.MX 95 applications processor libcamera SW stack:

- uGuzzi IPA: A production quality software in closed-source format.
- NXP IPA: An open source IPA implementation, which can be used for enablement and demonstration purposes.

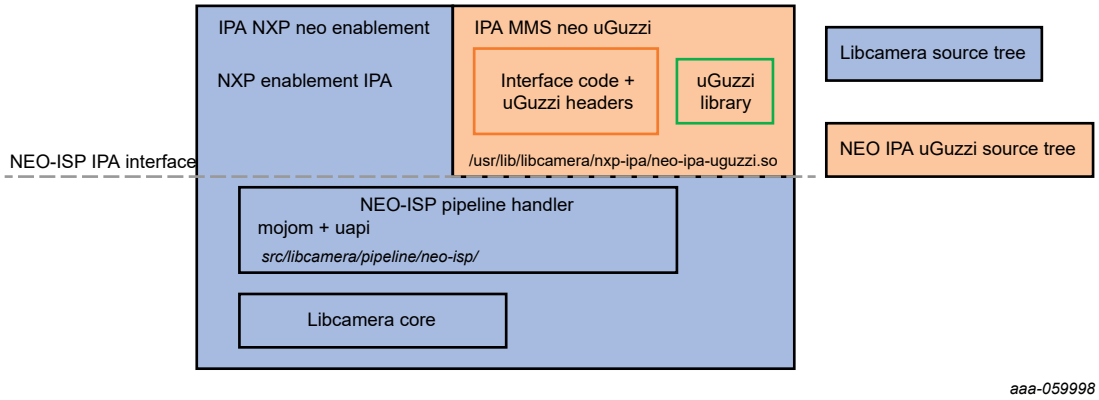

Figure 26. i.MX 95 libcamera IPA's architecture overview

4.2.4 Libcamera with uGuzzi IPA

The uGuzzi IPA is based on the imaging algorithms framework delivered by MMS. This framework aims to achieve good performance and image quality and is provided as a separate library `libuguzzi.so` dynamically linked with the interface code of the IPA.

The uGuzzi algorithms apply static and dynamic configuration. This uGuzzi setting writes the whole ISP configuration exposed by the UAPI. Therefore, the ISP driver setting is overwritten.

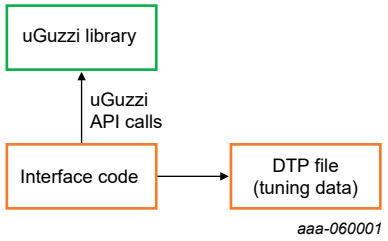

Figure 27. i.MX 95 uGuzzi libcamera IPA architecture overview

Tuning data specific to a sensor is available in a dynamic tuning parameters (DTP) file. To get the ISP tuning data, the IPA extracts the DTP. This file is part of the uGuzzi IPA source tree.

This uGuzzi IPA is built out of the libcamera tree, as it includes a third-party library. As per the libcamera framework, such IPA execution runs in a separate isolated process.

4.2.5 Libcamera with NXP IPA

IPA consists of several algorithms:

- Some algorithms are used for static configuration. They only execute once at the first frame to configure an ISP block in a fixed way for the whole duration of the camera stream.
- Some algorithms are used for dynamic configuration. They process the ISP statistics of each camera stream frame to produce updated configurations for the ISP, or sensor, or both, which are applied to the subsequent frames.

Table 15 lists the NXP NEO-ISP algorithms currently supported.

Table 15. List of algorithms supported in NXP open source IPA

Algorithm	Description	Type
Automatic gain/Exposure control (AGC/AEC)	Controls the sensor's gain and exposure based on ISP histograms (STAT unit)	Dynamic
Automatic white balance (AWB)	Controls the ISP white balance correction (OB WB unit) based on the color temperature unit statistics.	Dynamic
Black Level Correction (BLC)	Configures the ISP offset (OB WB unit) to apply to reach the zero value for the black pixel color.	Static
Color Correction Matrix (CCM)	Configures the ISP RGB to YUV unit according to the CCM and to the color temperature.	Dynamic
Dynamic range compression (DRC)	Configures the DRC unit Global LUT and gain.	Static/Dynamic
Gamma Output Correction control (GOC)	Configures the GCM block of the ISP.	Static
HDR decompression	Configures the HDR decompression unit, necessary with the sensors companding the data	Static
HDR merge	Configures the pixels combination of the two images of line path 0 and line path 1 into a single output.	Static
Lens Shading Correction control (LSC)	Configures the vignetting LUT of the ISP used to compensate for the lens shading effect.	Dynamic
Pipe Conf	Configures the subset of parameters INALIGN and LPALIGN from the Pipeline Configuration ISP block.	Static
RGBIr	Configures the RGBIR and the IR compression units, required with sensors having a RGBIr matrix	Static

Note: *For detailed ISP hardware specification information, contact a local field applications engineer (FAE) or sales representative.*

4.2.6 Embedded data support

The sensor configuration is essential for the image algorithms to provide optimized settings for both the ISP and the sensor.

This sensor configuration is acquired in two ways:

- Embedded data: The embedded data refers to metadata providing information about the sensor settings used to capture the image, such as sensor exposure and gain.
- Sensor control history: The sensor control history maintains a record of the controls applied on a frame, according to the specified delay required by the control to take effect. If available, embedded data is used.

Depending on the sensor capability and configuration, embedded data can be transmitted as part of the image, as pixel data, typically at the top lines of the image. The image buffer size in the camera drivers and at the kernel space includes the additional lines. ISP does not decode the lines, the libcamera NXP NEO pipeline handler crops the image buffer to remove the lines and adjust the size accordingly. The libcamera NXP NEO pipeline handler supports the embedded data of the image, only when embedded data are in the top lines.

Alternatively, embedded data can be provided as a separate data type, or via a virtual channel. Currently, the camera stack does not support this method.

Figure 28 shows the Ox03c10 sensor, where the embedded data is prepended to the image frame and is in the top two lines.

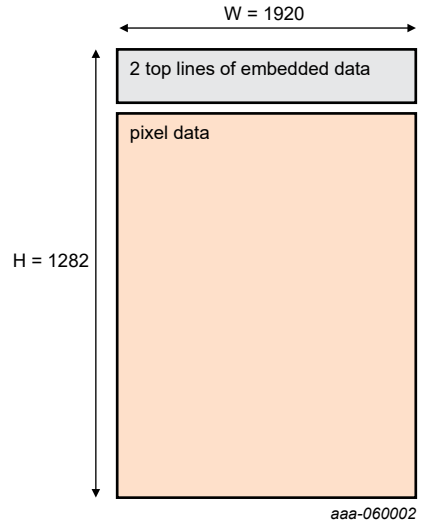

Figure 28. "In-Frame" embedded data representation on OX03C10

4.2.7 Operating libcamera for image capture and video streaming

For instructions to run libcamera on i.MX 95, see *i.MX Linux Users Guide* (document UG10163).

5 Sensor porting guide

Integrating a new sensor into the i.MX 95 camera software stack consists essentially of the following tasks:

- Writing a Linux kernel V4L2/media controller driver for this sensor.
  - Add device tree node for sensor and integration within the board and SoC architecture.
- Implement a CamHelper class for the new sensor.
- Provide a calibration file that the neo IPA consumes to manage IPA algorithms.
- Declare the sensor in the configuration file `config_ipa_uguzzi.yaml` with its associated configuration.

5.1 Writing a Linux kernel driver for a new sensor

For instructions on writing a capture sensor driver, refer to the The Linux Kernel.

The sensor driver must use the V4L2 sub-device framework, and the media control API. The Linux kernel provides examples for multiple camera sensor drivers in the `<linux_kernel>/drivers/media/i2c` folder.

Also, libcamera outlines specific requirements that the kernel driver for the sensor must adhere to. For further details, see Sensor Driver Requirements.

5.2 Implementing a libcamera CameraHelper for a new sensor

CameraHelper is a helper class used by each IPA to abstract camera specifics.

Each IPA dynamic library is built with a CameraHelper implementation. Therefore, the CameraHelper code must be part of each IPA source tree.

For the in-tree IPA, the relevant files are in the `<libcamera>/src/ipa/nxp/cam_helper` source tree directory.

For the uGuzzi IPA, the relevant files are in the `<neo-ipa-uguzzi>/cam_helper` source tree directory.

It contains essentially the following:

- The base class definition (`camera_helper.h`) and implementation (`camera_helper.cpp`).
- The camera-specific subclasses (`camera_helper_<camera model>.cpp`) that can override some methods of the base class for customization purposes.
- The meson configuration file (`meson.build`) that lists the source files and dependencies to be included for the build.

Table 16. List of CameraHelper methods

Method	Description
setControls	Configure the camera helper with sensor control values. This function passes the sensor control list populated with the actual control values read from the sensor. The usage from CameraHelper can be for instance to access the sensor calibrated values in OTP.
setCameraMode	Configure the camera helper for the current sensor mode of operation (camera sensor information)
controlListSetAGC	This function abstracts the AGC control for sensors with a proprietary programming model, converting the computed exposure and gain into specific sensor AGC controls.
controlInfoMapGetExposureRange	Retrieve min, max, and default values for exposure. At least one value is reported in each vector corresponding to the main (long) exposure. If it supports multiple captures (short and/or very short), the implementation can append extra values.
controlInfoMapGetAnalogGainRange	Retrieve min, max, and default values for analog gain. At least one value is reported in each vector corresponding to the main (long) analog gain. If it supports multiple captures (short and/or very short), the implementation can append extra values.
controlListSetAWB	Configure the sensor with white balance gain. This method is an optional that is used when the following conditions are met: • IPA can control white balance gains in sensors rather than in the ISP and this option is enabled. • The sensor driver actually provides a control for white balance gain configuration. If the ISP controls the white balance gains, then this method is ineffective.
parseEmbedded	To extract metadata, parse the embedded data buffer.
sensorControlsToMetaData	The purpose of this function is to build a metadata control list matching a sensor control list. It can be used when the sensor does not actually report embedded data to represent the sensor's state in the metadata format.
lineDuration	Line duration is computed by dividing the line length in pixels by the pixel rate. By default, the line length is configured to its minimum value, so use that value.
gainCode	Compute gain code from the analog gain absolute value.
gain	Compute the real gain from the V4L2 sub-device control gain code.

Table 17. CameraHelper sensor definition structure

Structure member	Description
Attributes	Structure holding the characteristics of a camera sensor
Attributes::MdParams	Metadata parameters describing the optional metadata support of that sensor.
Attributes::MdParams::topLines	Metadata can be prepended as the first lines of the image. This field reports the number of embedded lines, zero if none.
Attributes::MdParams::controls	ControlIdMap of the controls reported by the metadata parser,
Attributes::delayedControlParams	The map of (delay, priority) pair definitions for the camera controls handled by 3A. It is used to initialize the DelayedControls object instantiated by the pipeline. The delay corresponds to the latency in the number of frames for the control value to be applied. Priority indicates that the control must be applied ahead of, and separately from other controls.

To support a new camera sensor, the associated CameraHelper subclass must be implemented. This implementation typically involves the creation of a dedicated source file and updating the `meson.build` accordingly.

An example of a CameraHelper subclass for the OX03C10 can be found here: `<ipa_root_dir>/cam_helper/camera_helper_mx95mbcam.cpp`

For tuning a new sensor with a tuning tool, the CameraHelper subclass for this new sensor must be implemented with at least the following APIs:

- `parseEmbedded()`: If the sensor-embedded data are available; otherwise `sensorControlsToMetaData()`
- `controlListSetAGC()`
- `controlListSetAWB()`: Depending on the conditions as described in the list of methods

5.3 Sensor porting guide – calibration file generation

5.3.1 Calibration file generation for NXP IPA

The tuning data of each sensor are described in yaml files located under `<libcamera>/src/ipa/nxp/neo/data/`.

The yaml files are:

- Named `<sensor>.yaml`
- Installed under `/usr/share/libcamera/ipa/nxp/neo/`

As an example for the Ox03c10 sensor, the associated yaml file is called `mx95mbcam.yaml`.

5.3.2 Calibration file generation for uGuzzi IPA

The tuning data of each sensor is generated using the MMS tuning tool in a binary DTP file located in the uGuzzi IPA source tree under `<neo-ipa-uguzzi>/data/`.

The DTP files are:

- Named as `database_<sensor_name>.bin`
- Installed under `/usr/share/libcamera/ipa/nxp/neo/uguzzi/`

As an example for the Ox03c10 sensor, the associated DTP file is called `database_mx95mbcam.bin`.

For optimized calibration/tuning operation, relying on embedded data.

The DTP for Ox03c10 sensor `database_mx95mbcam.bin` can be used as a template for tuning a new sensor. Some parameters from this template DTP must be set according to the sensor resolution, pattern, and bit depth.

Note:

This template does not provide an initial setting for HDR and RGB-IR.

The Pipeline configuration `PIPE_CONF` (except for the subset of parameters `INALIGN` and `LPALIGN`) and the Packetizer configuration available from the tuning tool are not propagated to the ISP driver. Indeed, the ISP driver sets the appropriate settings to these modules according to the video nodes configuration.

An NXP tuning tool is required to perform tuning and calibration. To access the tools and related document, contact a local field applications engineer (FAE) or sales representative.

5.4 Adding a new sensor in the configuration file

The new sensor model or entity entry should be defined in the configuration file `config_ipa_uguzzi.yaml`. See Section 6.2 for more details.

6 uGuzzi IPA usage details

By default, the IPA executed for the NXP neo pipeline is the NXP enabled IPA.

To run the uGuzzi IPA, set the following environment variable to specify the path where the IPA is located:

```
$ export LIBCAMERA_IPA_MODULE_PATH="/usr/lib/libcamera/ipa-nxp-neo-uguzzi"
```

If this environment variable is not set, the default NXP enabled IPA is loaded and executed.

Once the uGuzzi IPA is loaded and initialized, the following logs get displayed:

```
INFO IPAProxyNxpNeoWorker nxpneo_ipa_proxy_worker.cpp:537 Starting worker for
 IPA module /usr/lib/libcamera/ipa-nxp-neo- uguzzi/neo-ipa-uguzzi.so with IPC fd
 = 42
...
INFO NxpNeoUguzziIPA neo.cpp:946 IPANxpNeo UGUZZI_IPA_v0.2.0+16-9d9936a1
```

For information on using GStreamer pipelines and the cam application to capture frames while the uGuzzi IPA is configured, refer to the "GStreamer Pipelines" and "Cam Test Application" sections in the *i.MX Linux Users Guide* (document UG10163).

6.1 uGuzzi IPA configuration

The IPA can be configured using the configuration file located in the uGuzzi IPA source tree at `data/config_ipa_uguzzi.yaml`. The configuration file is installed under `/usr/share/libcamera/ipa/nxp/neo/uguzzi/config_ipa_uguzzi.yaml`. This file provides configuration, such as specific parameters for each connected camera.

Each camera mode including its resolution and bit depth should be specified in the configuration file with its associated:

- DTP file
- Tuning ID

- Tuning mode

If the camera mode entry does not exist, the IPA will fail. Refer to the description from `data/config_ipa_uguzzi.yaml` for more details.

6.2 uGuzzi IPA isolation

By default, the uGuzzi IPA runs in Isolated mode. However, for tuning and development purposes (connection with the tuning tool), disable the Isolation mode by setting the following environment variable:

```
$ export LIBCAMERA_IPA_DISABLE_ISOLATION="yes"
```

The "uGuzzi Connect" library is a software component that sits between uGuzzi and the tuning tool.

It acts as a listener on the target, handling protocol communication between the tuning tool and the IPA. It provides support to populate predefined memory regions to share data with a tuning tool.

In the non-isolated mode, the Live Tuning library can only operate on a single camera, which is, by default, the first one initialized by libcamera. This single camera can also by changed explicitly by the user.

For that purpose, the IPA configuration file `data/config_ipa_uguzzi.yaml` can be used to specify:

- The single camera to run: this camera should be used by the application.
- The socket port to use for the IP connection between the uGuzzi IPA and the Tuning Tool: if not specified, the port 50000 is used by default.

6.3 uGuzzi and tuning tool considerations

It is assumed that the camera sensor tuning and IPA tuning are performed in a lab before the final product is released to the market. Based on this assumption, the "uGuzzi Connect library" is not optimized for memory when deployed in the product's final software or firmware.

Therefore, ensure to disable live tuning support in the uGuzzi IPA package prior to building the final product software. This action removes the "uGuzzi Connect library" from the final software and firmware.

To disable Live Tuning/Control support in the uGuzzi IPA, set the Meson build option "live_control" to "disabled".

6.4 uGuzzi IPA logs

By default, the uGuzzi IPA logs are displayed in the debug console.

To configure the IPA to redirect the logs into a file, set the following environment variable:

```
$ export LIBCAMERA_IPA_UGUZZI_LOG_DIR=”/tmp/”
```

With this configuration, the files generated are called with the PID of the IPA `<ipa_pid>.log`.

7 Limitations

7.1 Camera software stack

Table 18 lists the limitations associated with camera software stack.

Table 18. Camera software stack limitations

Items	Description
Ox03c10 support	When using Ox03c10 sensors, the four Ox03c10 cameras must be connected to the deserializer.

Table 18. Camera software stack limitations*...continued*

Items	Description
HDR merges in the ISP	The HDR merge in ISP is not supported; current camera stack limitation.
Embedded data support	Embedded data is not available when capturing raw frame concurrently with a decoded stream.
Embedded data support	Embedded data is supported as pixel data in the top of the sensor image; a current camera stack limitation.
Sensor control history	Sensor control history is unreliable during frame loss events, especially when no embedded data is available or when capture is saved to a file.
Color space support	The color space supported is sRGB only.

7.2 uGuzzi IPA

Table 19 lists the limitations associated with uGuzzi IPA.

Table 19. uGuzzi IPA limitations

Items	Description
uGuzzi IPA	Output buffers access is not supported for debug purpose using the Tuning tool. Access to the RAW bayer input0 buffer is available.
Tuning tool use case	The tuning tool requests socket access, therefore, it must run in non-isolated mode. In non-isolated mode, only a single camera can be operated. As a result, tuning is performed for a single camera. By default, this single camera is the first one initialized by libcamera. It can also be changed explicitly with the configuration file `config_ipa_uguzzi.yaml`.
Multi cameras support	Multi cameras are supported only for IPA in Isolated mode. In this mode, a different single uGuzzi instance operates each camera.
Surround view support	Surround view with harmonized-multiple cameras is not supported.
User controls	User controls are not supported.
uGuzzi library	The "sensor_id" parameter for the uGuzzi initialization is hardcoded to the value "2010" in the neo IPA. However, the DTP generated for the Ox03c10 sensor allows for any value to be selected for this parameter, except for the following invalid options: • 2020 • 2021 **These specific values ("2020", "2021") are reserved and must not be used when generating a DTP with the tuning tool**.

8 Known issues

8.1 uGuzzi IPA

Table 20 lists the known issues related to uGuzzi IPA.

Table 20. uGuzzi IPA known issues

Items	Description
Ox03c10 and WB gains	Saturation issues can be observed when WB gains are not applied before HDR merge in the sensor due to the LCG exposure of the sensor. This occurs when WB gains location is configured in the ISP.
Invalid first startup frames	At capture startup, the first frames can be incorrect due to the uGuzzi algorithms convergence time and sensor delay applying controls

Table 20. uGuzzi IPA known issues*...continued*

Items	Description
Dark artifacts in red area	Dark artifacts can be observed in the red areas of the image under low light conditions
Brightness flickering	Brightness flickering can be observed with sudden changes from dark to very bright scene. A stabilization issue is coming from the uGuzzi AGLBCE algorithm.

9 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Revision history

Table 21 summarizes the revisions to this document.

Table 21. Revision history

Document ID	Release date	Description
UG10215 v.2.1	26 September 2025	Updated for the Linux BSP release LF6.12.34_2.1.0.
UG10215 v.2.0	26 June 2025	First revision to be delivered with the Linux BSP release LF6.12.20_2.0.0.
UG10215 v.1.0	26 March 2025	Initial NDA release.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents