

UG10215

i.MX 95 Camera Porting Guide

Rev. 2.4 — 25 June 2026

User guide

Document information

Information	Content
Keywords	UG10215, i.MX 95, ISP, sensor porting guide, IPA, camera, NEO-ISP, LF6.18.20_2.0.0
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).

The domain provides the following functions:

- MIPI Camera Serial Interface (MIPI CSI)
- CSI Pixel Formatter
- Image Sensor Interface (ISI)
- Image Sensor Processing (ISP)

[Figure 1](#) shows the camera domain block diagram.

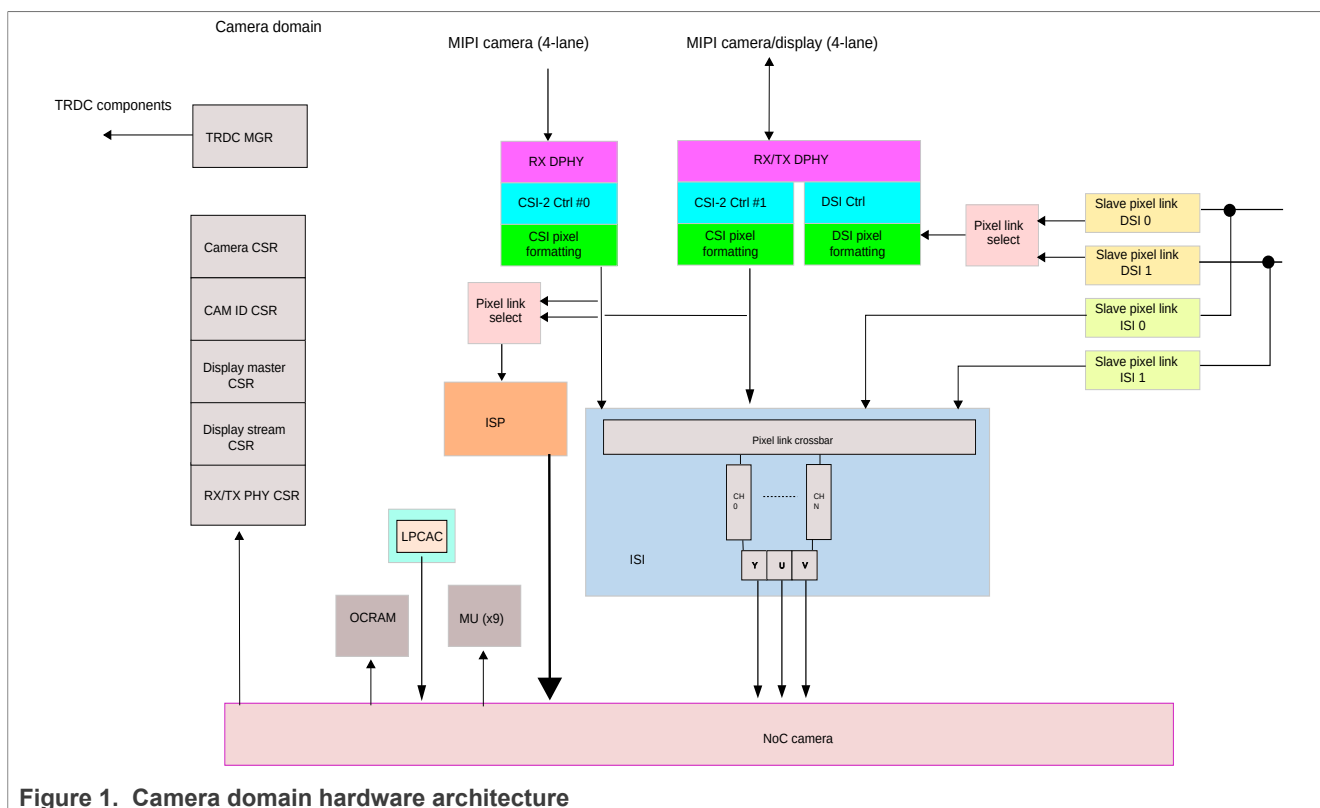


Figure 1. Camera domain hardware architecture

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).

Note: Contact a local field applications engineer (FAE) or sales representative to get the i.MX 95 Applications Processor Reference Manual (document IMX95RM).

i.MX 95 ISP supports both streaming and memory-to-memory (M2M) modes, and generic image sensor data processing pipeline.

For NXP Libcamera and uGuzzi IPA repositories, see [neo-ipa-uguzzi](#) and [libcamera](#).

Note: Current Libcamera stack for i.MX 95 ISP only supports the memory-to-memory mode operation.

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.

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 4.1 "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.

[Figure 2](#) shows the i.MX 95 camera subsystem software architecture with runtime and tools components.

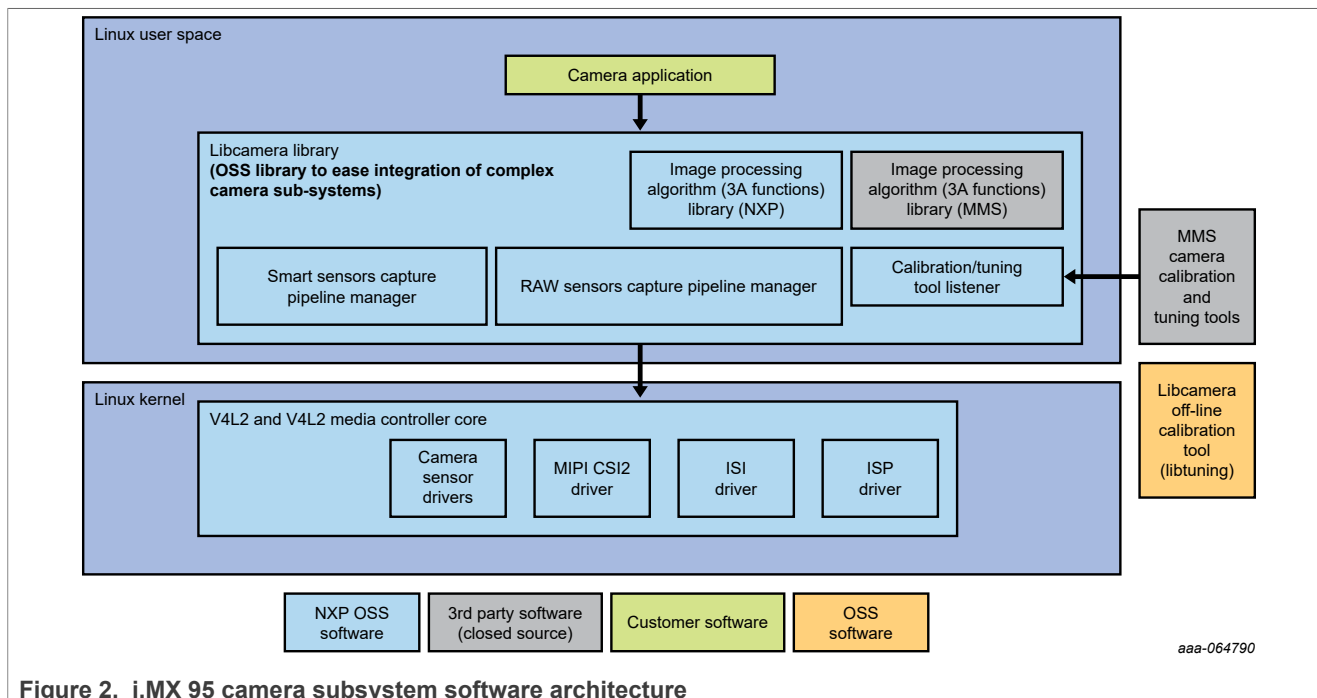


Figure 2. i.MX 95 camera subsystem software architecture

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.

Note: To obtain the tuning tool, contact your local field applications engineer (FAE) or sales representative.

4.1 Libcamera architecture

For Libcamera architecture, see the *i.MX Linux User's Guide* (UG10163).

4.1.1 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.

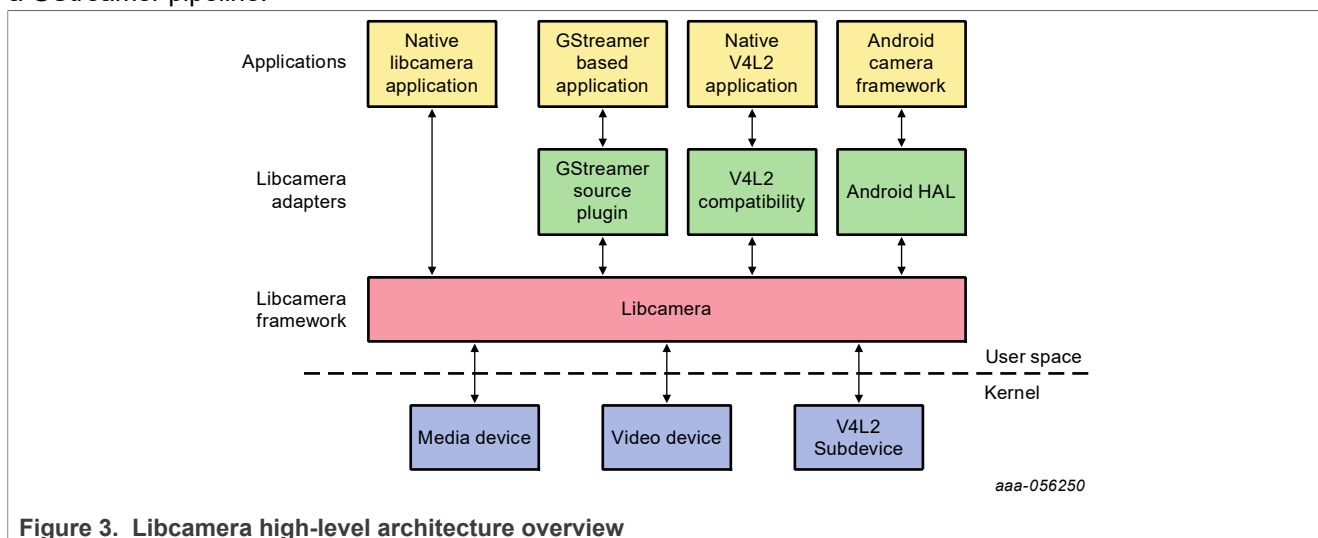


Figure 3. Libcamera high-level architecture overview

4.1.2 Libcamera i.MX 95 support overview

The i.MX 95 supports raw 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).

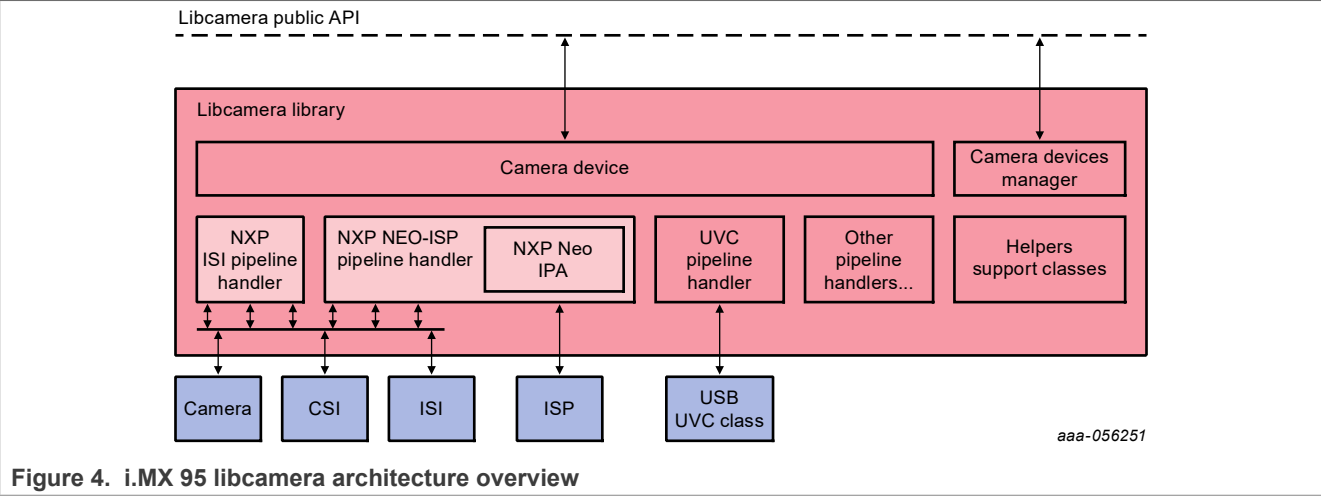


Figure 4. i.MX 95 libcamera architecture overview

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.

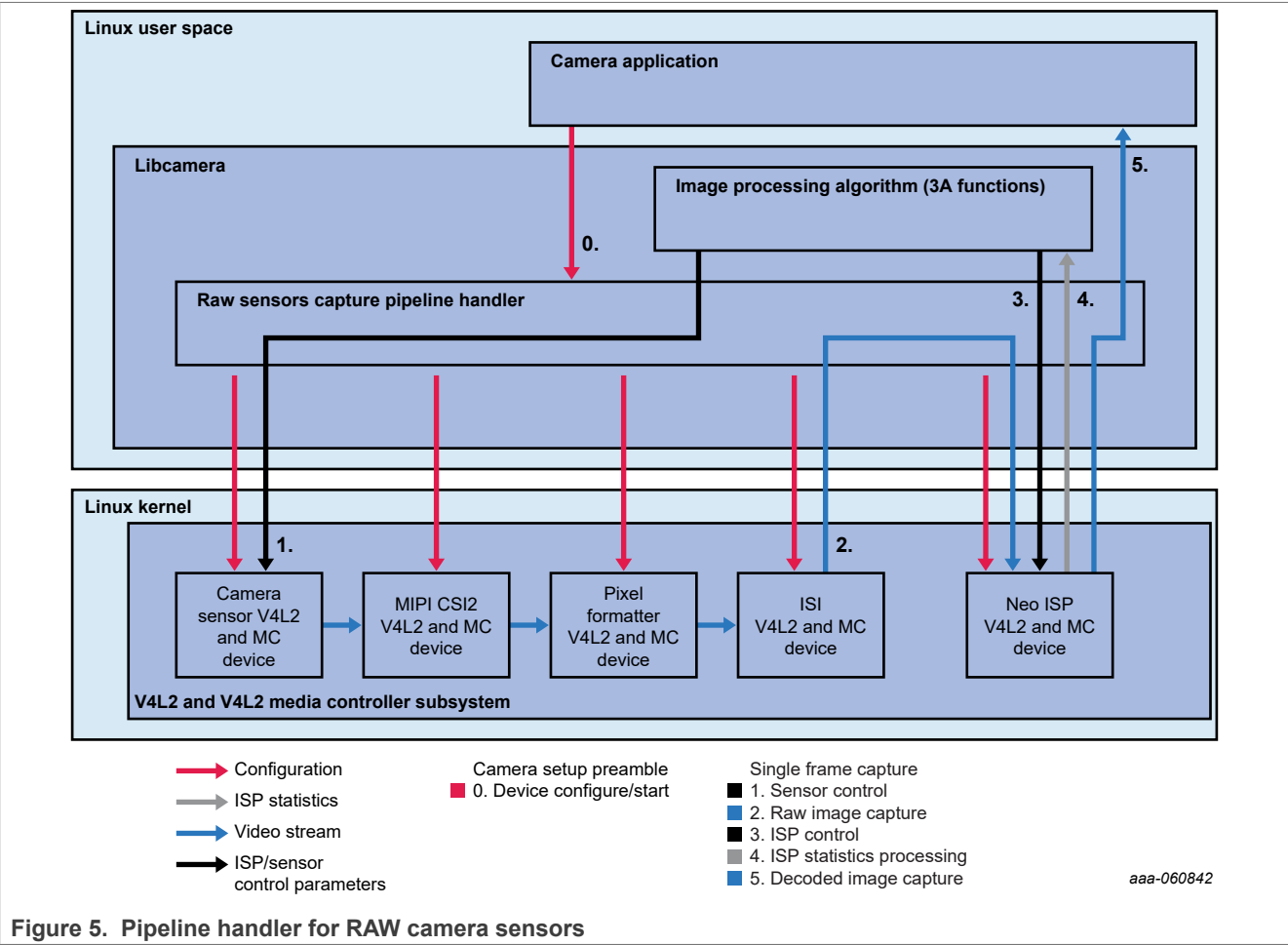


Figure 5. Pipeline handler for RAW camera sensors

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 metadata 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.

4.1.3 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 User's Guide* (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.

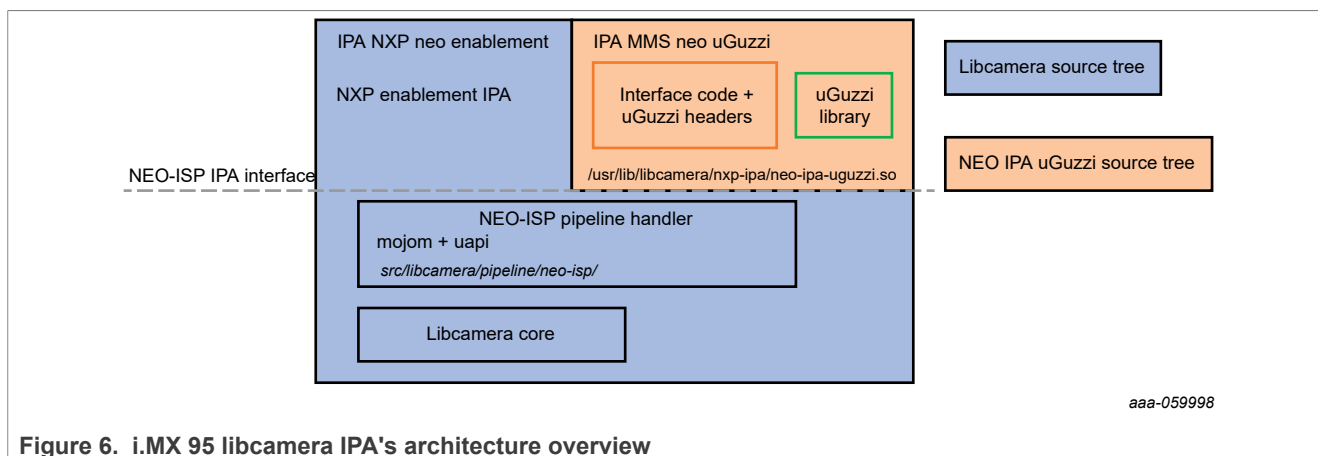
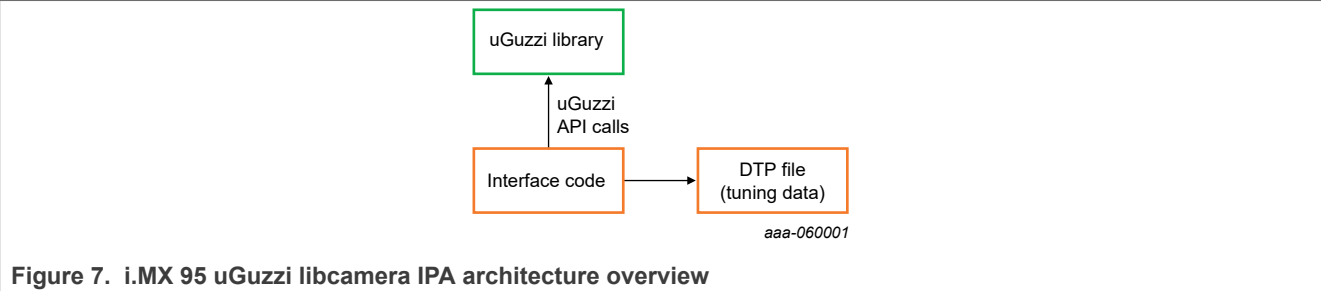


Figure 6. i.MX 95 libcamera IPA's architecture overview

4.1.4 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.



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.

4.1.5 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 1 lists the NXP NEO-ISP algorithms currently supported.

Table 1. List of algorithms supported in NXP open source IPA

Algorithm	Description	Type
Auto focus (AF)	Controls the lens position to adjust the focus (AF unit)	Dynamic
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

Table 1. List of algorithms supported in NXP open source IPA...continued

Algorithm	Description	Type
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.

4.1.6 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.

Figure 8 shows the OX03C10 sensor, where the embedded data is prepended to the image frame and is in the top two lines.

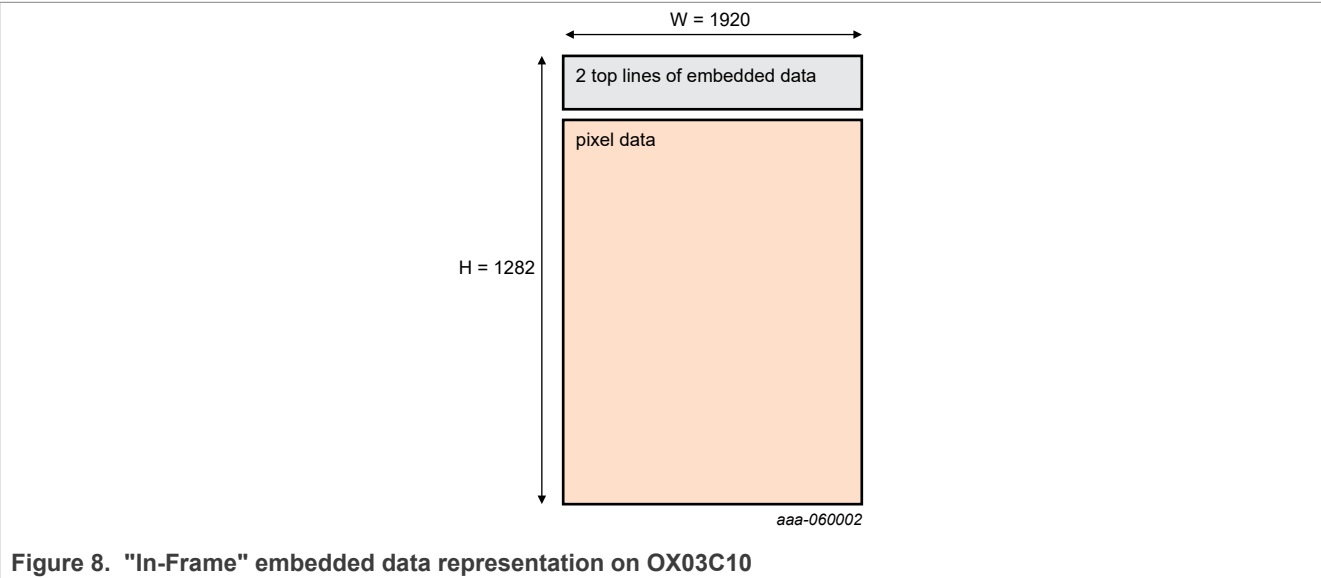


Figure 8. "In-Frame" embedded data representation on OX03C10

4.1.7 Operating libcamera for image capture and video streaming

For instructions to run libcamera on i.MX 95, see *i.MX Linux User's Guide* (UG10163).

5 Sensor porting guide

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

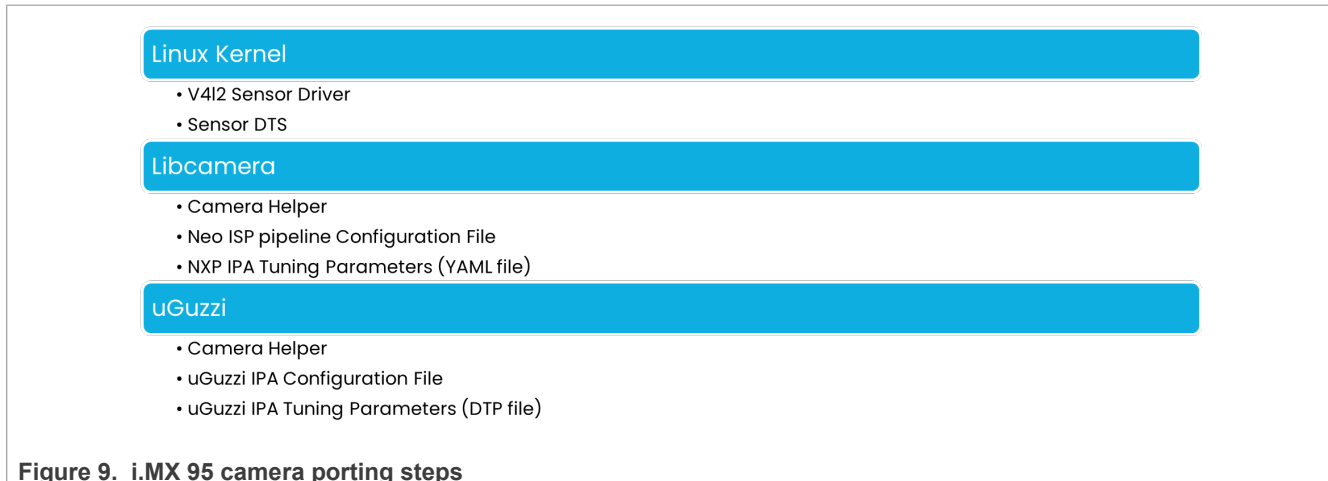


Figure 9. i.MX 95 camera porting steps

All porting activities are organized into three key components: Linux kernel enablement, Libcamera enablement, and ISP parameters calibration and tuning. This document describes each component in details in the subsequent sections.

5.1 i.MX 95 camera bandwidth calculation

Before porting a new sensor to the i.MX 95 platform, calculate the bandwidth required by the sensor and verify that the i.MX 95 platform supports it.

The i.MX 95 platform supports two CSI interfaces: a standalone CSI D-PHY and a combo DSI/CSI D-PHY, each providing bandwidth of up to 10 Gbps. In addition to the interface bandwidth, the overall capture capability of the i.MX 95 is constrained by the maximum pixel clock frequency supported within the camera domain.

On the i.MX 95 platform, the pixel clocks for the Pixel Formatter, ISI, and ISP modules are derived from the `cam_isi_clk_root` clock source.

Table 2. i.MX 95 pixel clock constraint

Root clock	Maximum Frequency (MHz)		
	Low drive mode	Nominal mode	Override mode
<code>cam_isi_clk_root</code>	333.33	500	666.67

Assuming that the camera module runs at the $W \times H @ BPP \& FPS$ setup, calculate the bandwidth required by the TX side as follows:

- Required DPHY bandwidth:

$$DPHY \text{ Bandwidth} = W * H * BPP * FPS * (1 + horizontal_blanking) * (1 + vertical_blanking) = W * H * BPP * FPS * (1 + total_blanking)$$
- Required Rx pixel clock:
 - The i.MX 95 supports only one pixel per clock cycle for the YUV/RGB inputs.
 - The i.MX 95 supports multiple pixels per clock cycle for the RAW format input, with the RAW32 mode enabled:

$$Rx \text{ Pixel Clock} \geq DPHY \text{ Bandwidth} / BPP / (pixels_per_clock)$$

Table 3. i.MX 95 RAW32 mode support

Format	RAW32 mode	Pixels per Clock Cycle
YUV/RGB	Not supported	1
RAW6/7/8	Enabled	4
	Disabled	1
RAW10/12/14/16	Enabled	2
	Disabled	1

It is recommended to use a 20% total blanking rate for the assessment. For detailed requirements, see the sensor data sheet.

5.2 Linux kernel enablement

5.2.1 Writing a V4L2 driver for the linear sensor

The sensor driver must be fully compliant with the V4L2 subdevice framework. We provide several examples for both single-camera and multiple-camera use cases in the `<linux_kernel>/drivers/media/i2c` folder.

The Libcamera interfaces with the camera sensors through the V4L2 control API. It uses the subdevices that are exposed to the user space by the corresponding sensor drivers. Therefore, all sensor drivers must implement and support the required V4L2 controls listed below. The i.MX 95 RX D-PHY driver requires that the sensor driver provides a valid link frequency control to correctly configure the `HS_SETTLE` timing. Configure the `link_freq` property to match the data rate of the connected sensor to ensure proper operation.

```
link_freq = hs_clk_freq = data_rate_per_lane / 2
```

- V4L2_CID_LINK_FREQ
- V4L2_CID_ANALOGUE_GAIN
- V4L2_CID_EXPOSURE
- V4L2_CID_HBLANK
- V4L2_CID_PIXEL_RATE
- V4L2_CID_VBLANK
- V4L2_CID_CAMERA_ORIENTATION
- V4L2_CID_CAMERA_SENSOR_ROTATION

You can implement the digital gain control to achieve smoother gain transitions. However, it may also introduce additional image noise.

5.2.2 Writing a V4L2 driver for the HDR sensor

To achieve a higher dynamic range, the HDR sensors must capture multiple gain and exposure signals. The i.MX 95 ISP supports two HDR modes:

- Native HDR: The sensor outputs a single compressed stream and the stitching is done at the sensor side.
- 2-exposure Staggered HDR: The sensor outputs both the DCG and VS streams and the stitching is done at the ISP side.

The V4L2 routing mechanism defines how the data flows inside a complex camera device, especially when a device has multiple internal data paths or multiple streams. For the 2-exp Staggered HDR sensors, do the following:

- Implement the routing to control the long and short streams independently.
- Implement additional controls that configure the analog gain and exposure settings independently for dual streams.

```

- V4L2_CID_AGAIN_MULTI
- V4L2_CID_EXPOSURE_MULTI
- V4L2_CID_HDR_SENSOR_MODE

```

An example driver is in the `<linux_kernel>/drivers/media/i2c/ox05b1s/ox05b1s_mipi.c` file. This driver supports both the OS08A20 and OX05B1S image sensors.

A native HDR sensor operates differently by internally stitching multiple exposure frames within the sensor and outputting as a single compressed stream. Therefore, additional internal pad and routing ioctls are not required in the sensor driver. However, it remains necessary to implement control interfaces that allow independent configuration of analog and digital gain, as well as exposure settings, for each frame prior to the stitching process. Check the `<linux_kernel>/drivers/media/i2c/ox03c10.c` as an example.

5.2.3 Writing a DTSO file for a new sensor

See the `imx95-19x19-evk-os08a20.dts` file for a single-camera case and `imx95-19x19-evkox03c10.dts` for a multiple-camera case. The example device tree overlays are in the `<linux_kernel>/arch/arm64/boot/dts/freescale` folder.

Taking the OS08A20 as an example, here are some mandatory changes to create a DTSO for a new sensor:

- Create a sensor device node under the corresponding **I2C bus**, depending on the board design.
- Update the **I2C slave address** according to the sensor data sheet.
- Update the **pinctrl** setting for the reset and/or power-down pins, depending on the board design.
- Update the **data-lanes** property under the sensor and mipi-csi device node according to the sensor configuration.
- Keep the **orientation** and **rotation** properties, because the sensor driver must register the V4L2 controls associated with these properties.
- Update the endpoint name under the **sensor and mipi-csi device** node accordingly.
- Make sure that the **compatible** property matches the sensor driver.

```

&lpi2c3 {
    #address-cells = <1>;
    #size-cells = <0>;
    status = "okay";

    os08a20: os08a20_mipi@36 {
        compatible = "ovti,os08a20";
        reg = <0x36>;
        clocks = <&dummy>;
        reset-gpios = <&i2c3_gpio_expander_20 2 GPIO_ACTIVE_LOW>;
        avdd-supply = <&reg_dummy>;
        dovdd-supply = <&reg_dummy>;
        dvdd-supply = <&reg_dummy>;
        orientation = <2>;
        rotation = <0>;
        status = "okay";

        port {
            os08a20_mipi_0_ep: endpoint {
                remote-endpoint = <&mipi_csi0_ep>;
                data-lanes = <1 2 3 4>;
                clock-lanes = <0>;
            };
        };
    };
}

```

```
};  
};  
};  
  
&mipi_csi0 {  
    status = "okay";  
  
    ports {  
        #address-cells = <1>;  
        #size-cells = <0>;  
  
        port@0 {  
            reg = <0>;  
            mipi_csi0_ep: endpoint {  
                remote-endpoint = <&os08a20_mipi_0_ep>;  
                data-lanes = <1 2 3 4>;  
                clock-lanes = <0>;  
            };  
        };  
  
        port@1 {  
            reg = <1>;  
            mipi_csi0_out: endpoint {  
                remote-endpoint = <&formatter_0_in>;  
            };  
        };  
    };  
};  
};
```

5.2.4 Compiling and updating

1. Build the kernel:

```
$ git clone https://github.com/nxp-imx/linux-imx.git -b <release_tag>  
$ cd linux-imx  
$ source <toolchain_path>  
$ make mrproper  
$ make ARCH=arm64 imx_v8_defconfig O=./build_v8  
$ cd build_v8/  
$ make ARCH=arm64 -j16
```

2. Update the driver module and device tree files to the target board:

```
$ scp arch/arm64/boot/dts/freescale/<sensor.dtb> root@<target_IP>:/run/media/  
boot-mmcb1k1p1  
$ scp drivers/media/i2c/<sensor.ko> root@<target_IP>:/lib/modules/  
<kernel_version>/kernel/drivers/media/i2c
```

5.3 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 4. List of CameraHelper methods

Method	Description	Details
<code>sensorControlList</code>	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.	Optional
<code>setCameraMode</code>	Configure the camera helper for the current sensor mode of operation (camera sensor information)	Optional
<code>controlListSetAGC</code>	This function abstracts the AGC control for sensors with a proprietary programming model, converting the computed exposure and gain into specific sensor AGC controls.	Optional
<code>controlInfoMapGetExposureRange</code>	Retrieve the minimum, maximum, 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.	Optional
<code>controlInfoMapGetAnalogGainRange</code>	Retrieve the minimum, maximum, 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.	Optional
<code>controlListSetAWB</code>	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.	Optional
<code>parseEmbedded</code>	To extract metadata, parse the embedded data buffer.	Optional
<code>sensorControlsToMetadata</code>	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.	Optional
<code>exposure</code>	Convert from line length to line duration.	Optional
<code>exposureLines</code>	Convert from line duration to line length.	Optional

Table 4. List of CameraHelper methods...continued

Method	Description	Details
hblankToLineLength	Calculate the line duration based on the sensor H-Blank.	Optional
gainCode	Compute gain code from the analog gain absolute value.	Mandatory
gain	Compute the real gain from the V4L2 sub-device control gain code.	Mandatory

Table 5. 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::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`.

The CameraHelper subclass for the sensor Sony IMX519 is an example where only the mandatory methods need to be implemented (see `camera_helper_imx519.cpp`).

5.4 Libcamera enablement

5.4.1 Updating configuration files

This section describes how to update the configuration files.

5.4.1.1 Updating the ISP pipeline configuration file

The NEO ISP pipeline configuration file is a YAML file that defines the camera-specific properties and global pipeline settings. It is loaded at the pipeline initialization phase to configure the camera behavior and capabilities.

- It is located in the `<libcamera>/src/libcamera/pipeline/nxp/neo/data/config.yaml` file.
- The camera section defines camera-specific properties, which specifies some attributes for a given camera. The key properties are:
 - `model`: camera model name.
 - `format`: mapping of {`bpp`, `size`} to select specific sensor format.

- streams: stream values belong to [image1, edata]. The image1 stream is used for the ISP-based HDR merge and RGB-IR context switch. The edata means that there is a dedicated stream available for embedded data.

The configuration file is installed in the `/usr/share/libcamera/pipeline/nxp/neo` folder. An example configuration is provided below to demonstrate how the OS08A20 sensor can be set up to operate in the 1920×1080 RAW10 linear mode. Specify [image1] in the configuration file to enable the routing activation for 2-exp Staggered HDR sensors.

```
cameras:
- model: os08a20
  format: {bpp: 10, size: [1920, 1080]}
# Uncomment the line below to enable os08a20 HDR operation
#streams: [image1]
```

5.4.1.2 Updating the uGuzzi IPA configuration file

To use the uGuzzi IPA, define a new sensor model or entity entry in the `<neo-ipa-uguzzi>/data/config_ipa_uguzzi.yaml` configuration file.

The uGuzzi IPA configuration file is installed in the `/usr/share/libcamera/ipa/nxp/neo/uguzzi/config_ipa_uguzzi.yaml` file. An example configuration is provided below to demonstrate how to support both the Linear mode and the 2-exp Staggered HDR mode for OS08A20.

See the YAML file for more descriptions about the parameters.

```
- model: os08a20
  socket-port: 50010
  profiles:
  - resolution: [3840, 2160]
    bit-depth: 12
    mode: standard
    dtp-file: database_os08a20_SDR_3840-2160_12bpp.bin
    tuning-id: 2010
    tuning-mode: 0
  - resolution: [3840, 2160]
    bit-depth: 12
    mode: hdr
    dtp-file: database_os08a20_HDR_3840-2160_12bpp.bin
    tuning-id: 2010
    tuning-mode: 0
```

5.4.2 Compiling and updating

1. Build the Libcamera and update the package to the target board:

```
$ git clone https://github.com/nxp-imx/libcamera.git -b <release_tag>
$ cd libcamera
$ source <toolchain_path>
$ meson setup --buildtype=plain -Dprefix=/usr build -Dpipelines="nxp/neo,imx8-isi" -Dv4l2=enabled -Dcam=enabled -Dlc-compliance=disabled -Dtest=false -Ddocumentation=disabled -Dqcam=disabled
$ DESTDIR=./install ninja -C build install
$ scp -r ./build/install/usr/* root@<target_IP>:/usr
```

2. If using the uGuzzi IPA, build it and update the package to the target board:

```
$ git clone https://github.com/nxp-imx/neo-ipa-uguzzi.git -b <release_tag>
$ cd neo-ipa-uguzzi
$ source <toolchain_path>
$ meson setup --buildtype=plain -Dprefix=/usr/ build
$ DESTDIR=./install ninja -C build install
$ scp -r ./build/install/usr/* root@<target_IP>:/usr
```

Note: The toolchain used to compile should be compatible with the Libcamera and uGuzzi IPA version.

5.4.3 Calibration file generation

This section describes how to generate a calibration file.

5.4.3.1 Calibration file generation for NXP IPA

For the NXP IPA, the tuning data of each sensor are described in the YAML files located in the libcamera source tree, in the <libcamera>/src/ipa/nxp/neo/data/ folder.

To support a new camera sensor, create a dedicated calibration file there. The local `meson.build` file is updated so that this file is installed.

The YAML files are:

- Named as <sensor>.yaml.
- Installed in the /usr/share/libcamera/ipa/nxp/neo folder.

To bring up a new sensor quickly, copy the `uncalibrated.yaml` file and rename it to <sensor>.yaml with only the AWB/AGC enabled. All the other blocks will use the default settings.

As an example for the OS08A20 sensor, the associated YAML file is called `os08a20.yaml`. See it to configure other algorithms supported by the NXP IPA.

5.4.3.2 Calibration file generation for uGuzzi IPA

For the 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 in the <neo-ipa-uguzzi>/data/ folder.

Note: The MMS Tuning Tool is required to perform the calibration and tuning. To access the tools and related documents, contact a local Field Applications Engineer (FAE) or a sales representative.

To support a new camera sensor, create a dedicated calibration file there. The local `meson.build` file is updated so that this file is installed.

The DTP files are:

- Named as `database_<sensor_name>_<stream_mode>_<resolution>_<bitdepth>.bin`.
- Installed in the /usr/share/libcamera/ipa/nxp/neo/uguzzi folder.

As an example for the OS08A20 sensor, the associated DTP file is called `database_os08a20_SDR_3840-2160_12bpp.bin`. To support multiple operating modes for a single sensor, specify a distinct DTP file for each mode within the uGuzzi IPA configuration file. For details, see [Section 5.4.1.2](#).

5.5 Adding a new sensor in the uGuzzi IPA configuration file

The new sensor model or entity entry should be defined in the configuration file `config_ipa_uguzzi.yaml`. See [Section 6.1](#) 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 User's Guide* (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, bit depth, and stream mode 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 be 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 Starting the capturing and debugging tips

This section describes how to start the capturing and provides debugging tips.

7.1 Capturing commands

You can start capturing the pipeline after updating the software enablement on the target board, which includes:

- Linux Kernel image and device tree file
- Libcamera libraries, pipeline configuration file, and calibration file
- Additional uGuzzi IPA libraries, configuration file, and DTP file for using uGuzzi IPA

Libcamera comes with a test application `cam` that exercises the pipeline. It also provides the `libcamerasrc` plugin to bridge the Libcamera stack to the source of a GStreamer pipeline.

1. Make sure that the sensor is probed successfully before running the `cam` or `gstreamer` commands.
2. Specify the pipeline handler to use:

```
export LIBCAMERA_PIPELINES_MATCH_LIST='nxp/neo'
```

3. List the detected cameras:

```
cam -l
```

4. Capture 1000 decoded frames and display them over the HDMI output:

```
systemctl stop weston
cam \
  --camera 1 \
  --stream width=${W},height=${H},pixelformat=YUYV \
  --capture=1000 \
  --display=HDMI-A-1
```

5. Capture 20 RAW frames and save them to a file:

```
cam \  
  --camera 1 \  
  --stream width=${W},height=${H},pixelformat=${RAW} \  
  --capture=20 \  
  --file=frame-#.raw
```

6. Use the GStreamer pipeline for a preview:

```
gst-launch-1.0 \  
  libcamerasrc camera-name="${CAMERA0}" ! \  
  video/x-raw, format=YUY2 ! \  
  queue ! \  
  waylandsink
```

7.2 Debugging tips

This section provides debugging tips.

7.2.1 I2C communication issue

Implement a correct power-up sequence within the sensor driver to ensure a proper IC communication between the sensor and the SoC. If the sensor I2C device cannot be detected, do the following:

1. Check all supply voltages.
2. Check the sensor MCLK.
3. Check the reset and power sequence.

7.2.2 Capturing the hang issue

After starting the video capturing, the video pipeline may become stuck. To troubleshoot this issue, review and verify the following key items:

1. Check the i.MX95 Rx DPHY status register (PHY_RX).
2. Measure both single-ended and differential waveforms on the sensor clock lane and data lanes to verify compliance with the D-PHY specification.

7.2.3 Libcamera pipeline issue

If you encounter any errors reported by Libcamera, increase the Libcamera log level to obtain more detailed diagnostic information:

1. To enable all debug log (0 is for DEBUG, 4 is for FATAL)

```
export LIBCAMERA_LOG_LEVELS=*:0
```

2. To enable specific categories, use a comma-separated list of category:level pairs.

```
export LIBCAMERA_LOG_LEVELS=<category:level>[,<category,level>[,...]]
```

3. Instead of outputting the log to stderr, redirect it to a file by the following setting:

```
export LIBCAMERA_LOG_FILE=</path/to/logfile.txt>
```

8 i.MX 95 EVK reference camera modules

[Table 6](#) provides NXP reference RAW camera modules used to implement and validate the i.MX 95 applications processor Linux camera software stack.

Table 6. 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	-
Raspberry Pi Sony imx219	• 8 Megapixels • Adjustable focus • Lenses: 62.1° H FOV	-
Arducam Sony imx519	• 16 Megapixels • Adjustable focus • Lenses: 66° H FOV	-
Raspberry Pi Sony imx708	• 11.9 Megapixels • Adjustable focus, PDAF • Lenses: 66° H FOV HDR	-

9 Limitations

9.1 Camera software stack

[Table 7](#) lists the limitations associated with camera software stack.

Table 7. Camera software stack limitations

Items	Description
OX03C10 support	When using OX03C10 sensors, the four OX03C10 cameras must be connected to the deserializer.
HDR merges in the ISP	When enabling the second stream (short capture) of the OS08A20 sensor, the frame rate is divided by 2.
RGBIr context switch	RGBIr context switch is not functional from the OX05B1S sensor driver.
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.

9.2 uGuzzi IPA

[Table 8](#) lists the limitations associated with uGuzzi IPA.

Table 8. uGuzzi IPA limitations

Items	Description
uGuzzi IPA	Output buffers access is not supported for debug purpose with the Tuning tool. Access to the RAW bayer input0 and buffer1 are 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 <code>config_ipa_uguzzi.yaml</code> .
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.

10 Known issues

10.1 uGuzzi IPA

[Table 9](#) lists the known issues related to uGuzzi IPA.

Table 9. 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

11 Note About the Source Code in the Document

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

Copyright 2026 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.

12 Revision history

[Table 10](#) summarizes the revisions to this document.

Table 10. Revision history

Document ID	Release date	Description
UG10215 v.2.4	25 June 2026	Upgraded to the 6.18.20 kernel, removed the i.MX 95 15x15 EVK and i.MX 95 19x19 Verdin support, and added the i.MX 93W and i.MX 952 support.
UG10215 v.2.3	26 March 2026	Updated for the Linux BSP release LF6.18.2_1.0.0.
UG10215 v.2.2	12 December 2025	Updated for the Linux BSP release LF6.12.49_2.2.0.
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

1	Introduction	2	11	Note About the Source Code in the
2	Overview	2		Document
3	i.MX 95 applications processor camera		12	Revision history
	domain hardware architecture	2		Legal information
4	i.MX 95 applications processor camera			
	software architecture	3		
4.1	Libcamera architecture	4		
4.1.1	Libcamera overview	4		
4.1.2	Libcamera i.MX 95 support overview	4		
4.1.3	NEO-ISP image-processing algorithm	6		
4.1.4	Libcamera with uGuzzi IPA	6		
4.1.5	Libcamera with NXP IPA	7		
4.1.6	Embedded data support	8		
4.1.7	Operating libcamera for image capture and			
	video streaming	8		
5	Sensor porting guide	9		
5.1	i.MX 95 camera bandwidth calculation	9		
5.2	Linux kernel enablement	10		
5.2.1	Writing a V4L2 driver for the linear sensor	10		
5.2.2	Writing a V4L2 driver for the HDR sensor	10		
5.2.3	Writing a DTSO file for a new sensor	11		
5.2.4	Compiling and updating	12		
5.3	Implementing a libcamera CameraHelper			
	for a new sensor	12		
5.4	Libcamera enablement	14		
5.4.1	Updating configuration files	14		
5.4.1.1	Updating the ISP pipeline configuration file	14		
5.4.1.2	Updating the uGuzzi IPA configuration file	15		
5.4.2	Compiling and updating	15		
5.4.3	Calibration file generation	16		
5.4.3.1	Calibration file generation for NXP IPA	16		
5.4.3.2	Calibration file generation for uGuzzi IPA	16		
5.5	Adding a new sensor in the uGuzzi IPA			
	configuration file	16		
6	uGuzzi IPA usage details	17		
6.1	uGuzzi IPA configuration	17		
6.2	uGuzzi IPA isolation	17		
6.3	uGuzzi and tuning tool considerations	18		
6.4	uGuzzi IPA logs	18		
7	Starting the capturing and debugging			
	tips	18		
7.1	Capturing commands	18		
7.2	Debugging tips	19		
7.2.1	I2C communication issue	19		
7.2.2	Capturing the hang issue	19		
7.2.3	Libcamera pipeline issue	19		
8	i.MX 95 EVK reference camera modules	20		
9	Limitations	20		
9.1	Camera software stack	20		
9.2	uGuzzi IPA	21		
10	Known issues	21		
10.1	uGuzzi IPA	21		

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.