EIQMPPUG

eIQ Media Processing Pipeline User's Guide

Rev. 4 — 10 January 2024 User guide

Document information

Information	Content
Keywords	eIQ, Media, Media Processing, Processing Pipeline, Library
Abstract	This document describes the Media Processing Pipeline software library for MCUs. The library is used for constructing media-handling components graphs for Vision-specific applications.

1 MCU Media Processing Pipeline

This document describes the MCU Media Processing Pipeline API.

1.1 Features overview

The Media Processing Pipeline for MCUs is a software library for constructing graphs of media-handling components for Vision-specific applications.

This is a clean and simple API which makes it easy to build and prototype vision-based applications.

1.1.1 Concept

The concept behind the API is to create a Media Processing Pipeline (MPP) based on processing elements. The basic pipeline structure - the *mpp* in the API context - has a chain/queue structure which begins with a **source element**:

- Camera
- Static image

The pipeline continues with multiple **processing elements** having a single input and a single output:

- Image format conversion
- Labeled rectangle drawing
- Machine learning inference with three frameworks:
  - Tensorflow Lite Micro
  - GLOW
  - DeepViewRT

The pipeline can be closed by adding a **sink element**:

- Display panel
- Null sink

Also, multiple basic *mpps* can be **joined** into a new one to which further elements can be added. An *mpp* can also be **split** when the same media stream must follow different processing paths. With these join/split operations, more complex pipelines can be constructed.

Compatibility of elements and supplied parameters are checked at each step and only compatible elements can be added in an unequivocal way.

After the construction is complete, each *mpp* must be started for all hardware, and software required to run the pipeline to initialize. Pipeline processing begins as soon as the the last start call is flagged.

Each pipeline branch can be stopped individually. The process involves stopping the execution and the hardware peripherals of the branch. After being stopped, each branch can be started again. To stop the whole pipeline, you must stop each of its branches separately.

At runtime, the application receives events from the pipeline processing and may use these events to update the elements parameters. For example, in object detection when the label of a bounding box must be updated whenever a new object is detected.

Summarizing, the application controls:

- Creation of the pipeline
- Instantiation of processing elements
- Connection of elements to each other
- Reception of callbacks based on specific events

- Updating specific elements (not all elements can be updated)
- Stopping the pipeline (includes shut down of the hardware peripherals)

Application does not control:

- Memory management
- Data structures management

The order in which an element is added to the pipeline defines its position within this pipeline, and therefore the order is important.

1.2 Example and references

See the examples/reference documentation for practical examples using the MPP API.

2 Deployment

The eIQ Media Processing Pipeline is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK.

The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com.

To include eIQ Media Processing Pipeline into the MCUXpresso SDK package, select both “eIQ” and “FreeRTOS” in the software component selector on the SDK Builder page.

For details, see, Figure 1.

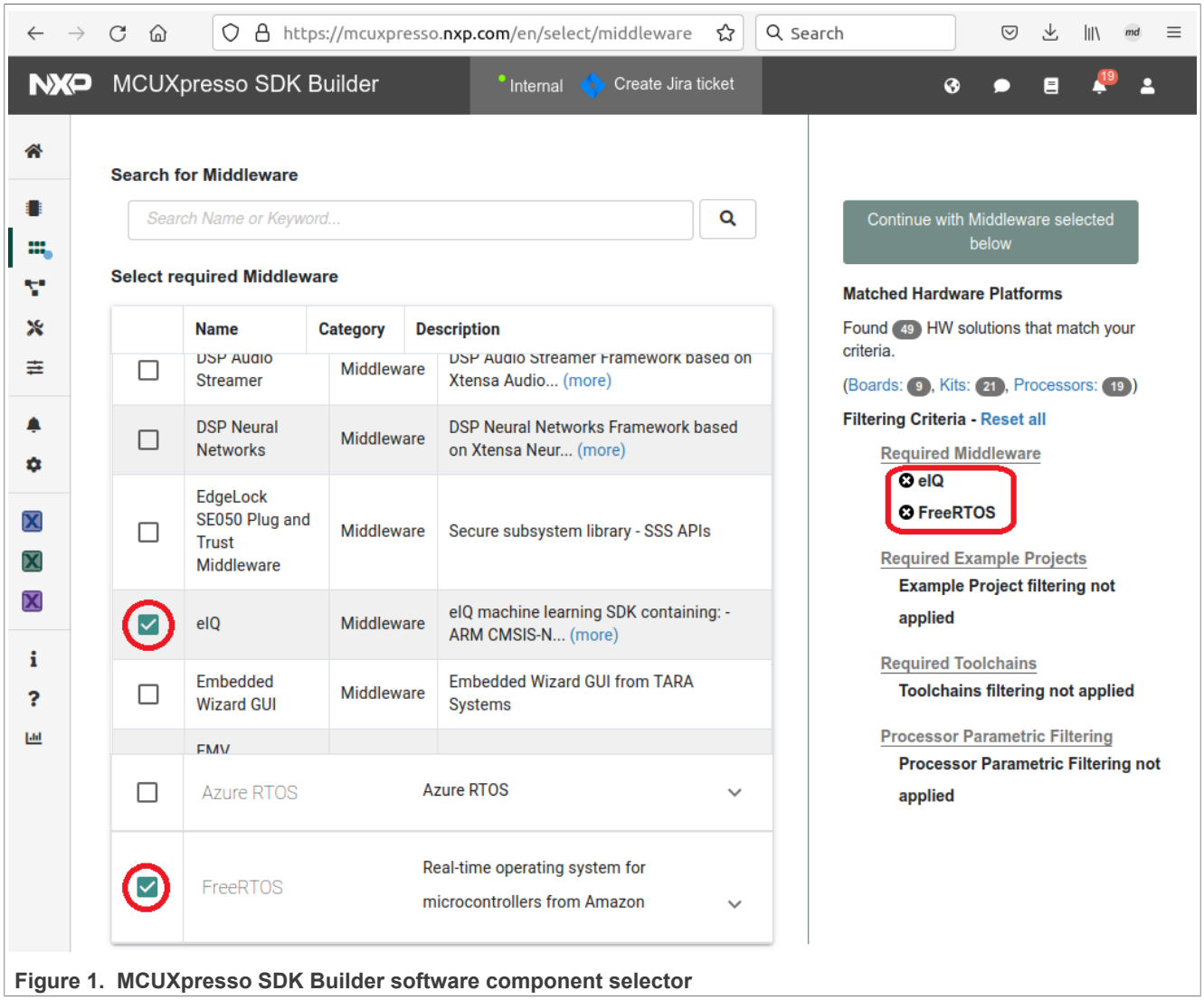

Figure 1. MCUXpresso SDK Builder software component selector

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to Figure 2 and Figure 3. The eIQ Media Processing Pipeline directories are highlighted in red.

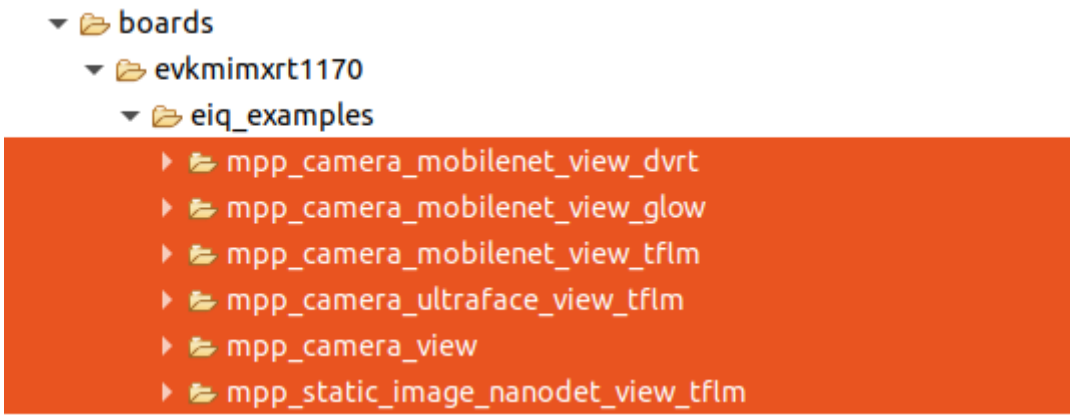

Figure 2. MCUXpresso SDK directory structure for examples

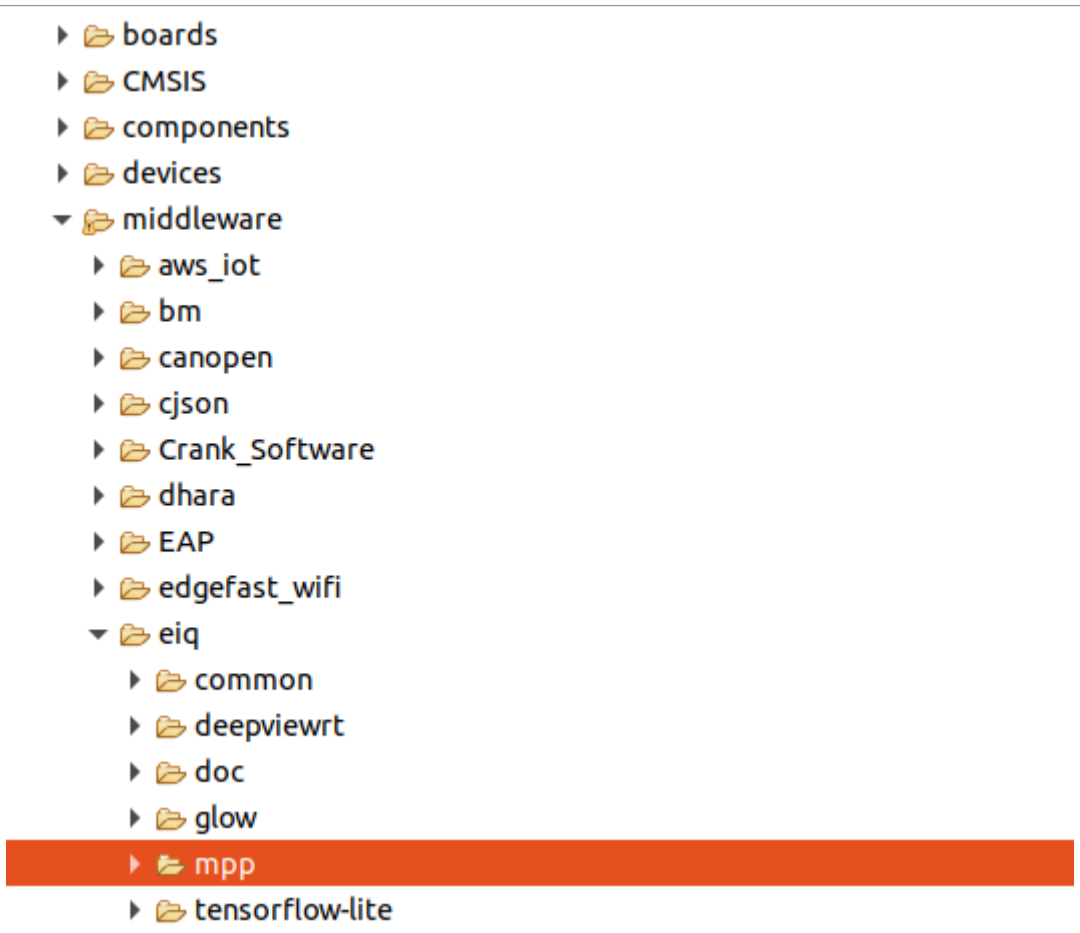

Figure 3. MCUXpresso SDK directory structure for mpp

The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

3 Example applications

3.1 How to get examples

The eIQ Media Processing Pipeline is provided with a set of example applications. For details, see Table 1. The applications demonstrate the usage of the API in several use cases.

Table 1. Example applications

Name	Description	Availability
`mpp_camera_view`	This basic example shows how to use the library to create a simple camera preview pipeline.	EVK-MIMXRT1170 EVKB-MIMXRT1170 EVKB-IMXRT1050
`mpp_camera_mobilenet_view_tflm`	This example shows how to use the library to create an image classification use case using camera as a source. The machine learning framework used is TensorFlow Lite Micro. The image classification model used is quantized Mobilenet convolution neural network model that classifies the input image into one of 1000 output classes.	EVK-MIMXRT1170 EVKB-MIMXRT1170 EVKB-IMXRT1050
`mpp_camera_mobilenet_view_glow`	This example shows how to use the library to create an image classification use case using camera as a source. The machine learning framework used is GLOW. The image classification model used is quantized Mobilenet convolution neural network model that classifies the input image into one of 1000 output classes.	EVK-MIMXRT1170 EVKB-MIMXRT1170 EVKB-IMXRT1050
`mpp_camera_mobilenet_view_dvrt`	This example shows how to use the library to create an image classification use case using camera as a source. The machine learning framework used is DeepViewRT. The image classification model used is quantized Mobilenet convolutional neural network model that classifies the input image into one of 1000 output classes.	EVK-MIMXRT1170 EVKB-MIMXRT1170 EVKB-IMXRT1050
`mpp_camera_ultraface_view_tflm`	This example shows how to use the library to create a use case for face detection using camera as a source. To generate a new static image for this example, rsee the documentation at: `eiq/mpp/tools/image_conversion.readme`. The machine learning framework used is TensorFlow Lite Micro. The face detection model used is quantized Ultraface slim model that detects multiple faces in an input image.	EVK-MIMXRT1170 EVKB-MIMXRT1170 EVKB-IMXRT1050
`mpp_static_image_nanodet_m_view_tflm`	This example shows how to use the library to create an object detection use case using a static image as a source. The machine learning framework is TensorFlow Lite Micro.	EVK-MIMXRT1170 EVKB-MIMXRT1170 EVKB-IMXRT1050

Table 1. Example applications...*continued*

Name	Description	Availability
	The object detection model used is quantized Nanodet m with two output tensors. The model performs multiple objects detection among 80 classes. The application also performs Intersection Over Union (IOU) and Non-Maximum Suppression (NMS) to pick the best box for each detected object.	

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG).

When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in Figure 4.

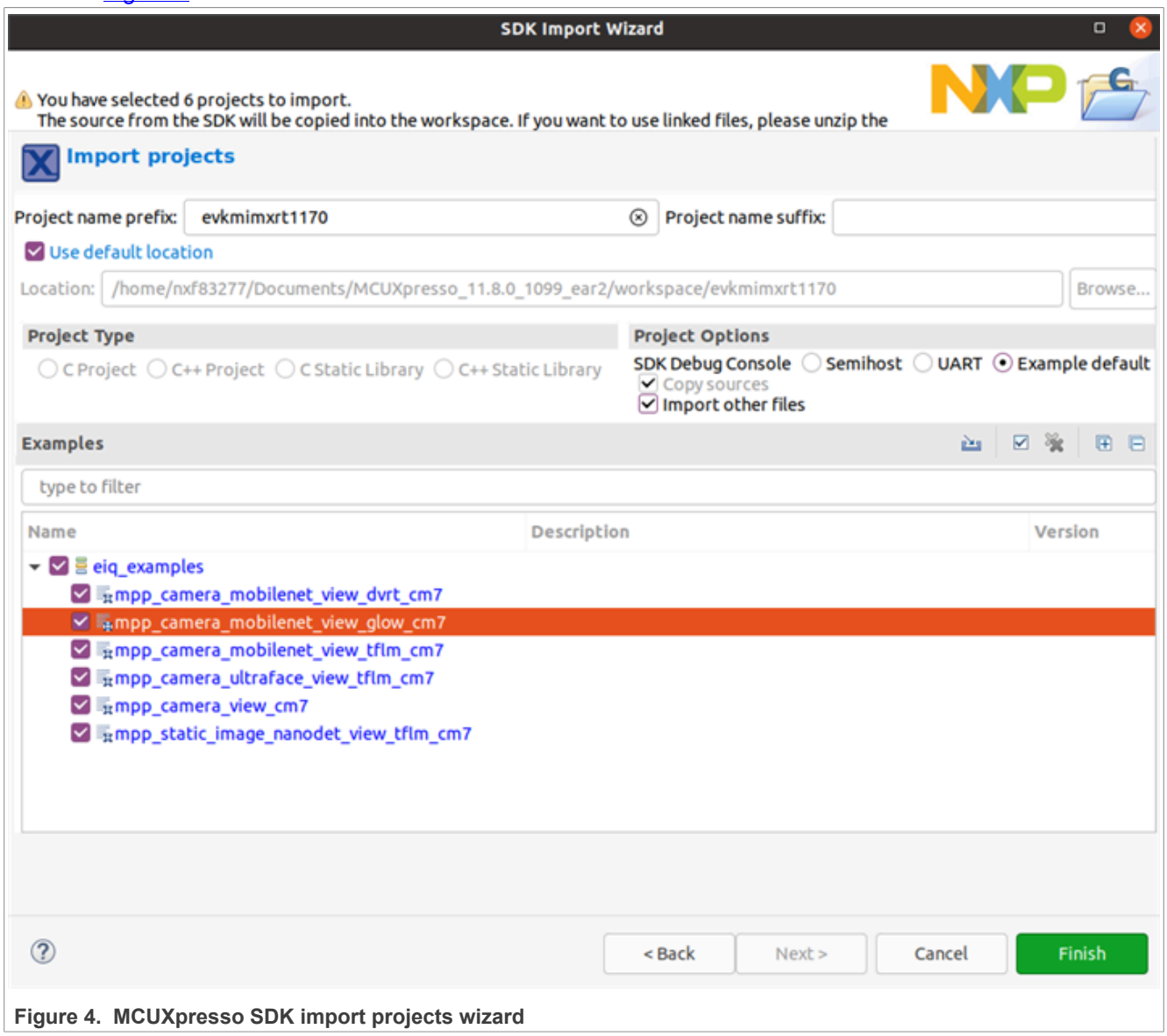

Figure 4. MCUXpresso SDK import projects wizard

The build scripts for armgcc are available under the directory as shown in Figure 5.

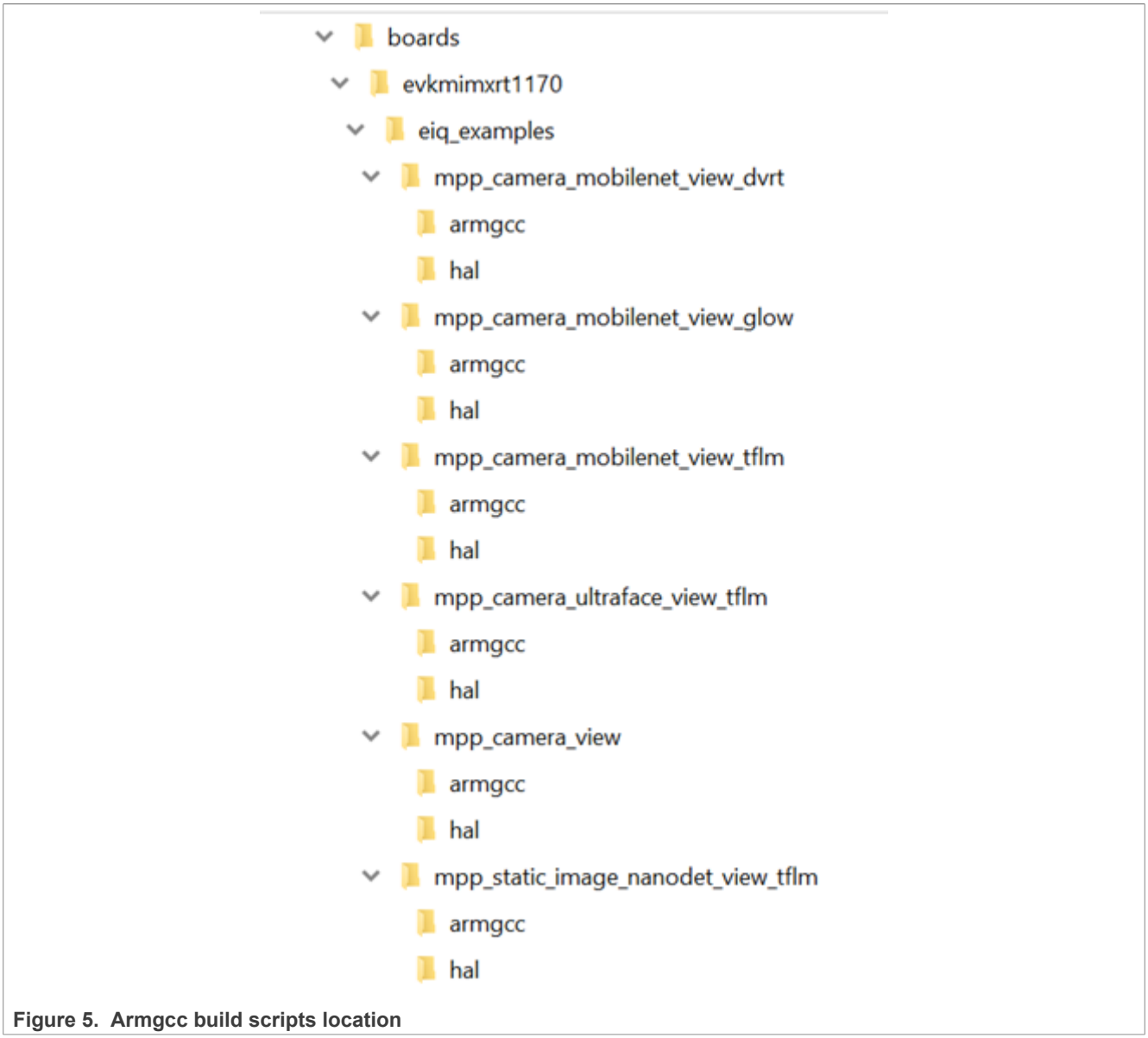

Figure 5. Armgcc build scripts location

3.2 Description of the mpp_camera_mobilenet_view example

This section provides a short description of the `mpp_camera_mobilenet_view` application.

This example shows how to use the library to create a use case for image classification using camera as source.

The machine learning frameworks used are TensorFlow Lite Micro, GLOW, or DeepViewRT.

The image classification model used is quantized Mobilenet convolutional neural network model [1] that classifies the input image into one of 1000 output classes.

- **High-level description**

1 https://www.tensorflow.org/lite/models

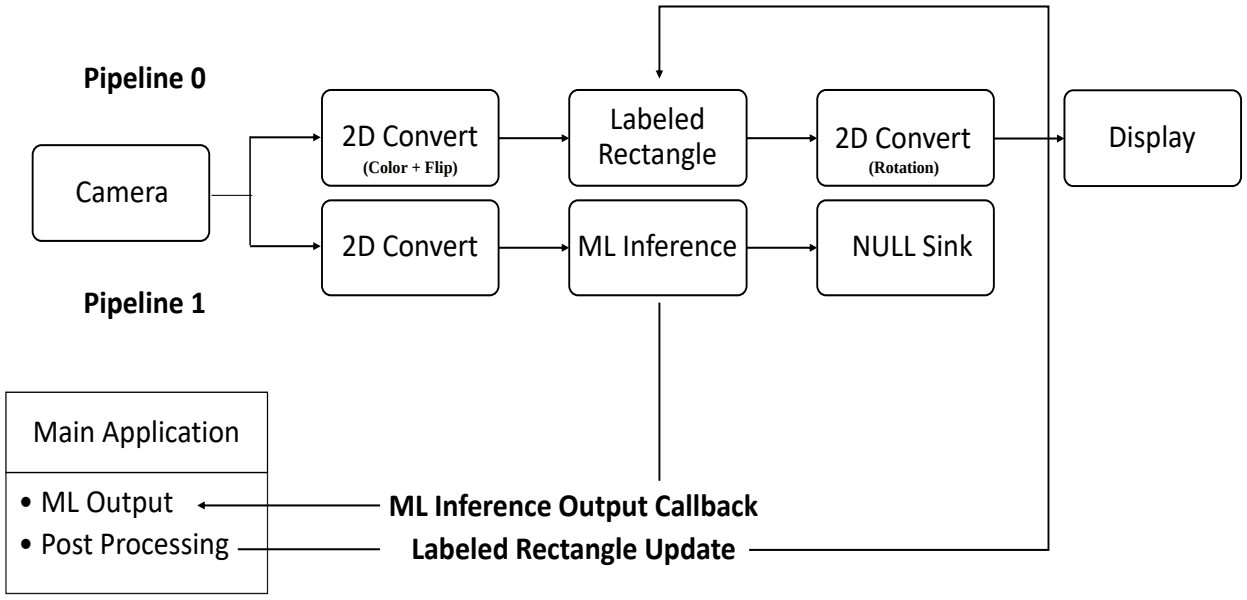

Figure 6. Application overview

- **Detailed description**
  The application creates two pipelines:
  - One pipeline that runs the camera preview.
  - Another pipeline that runs the ML inference on the image coming from the camera.
  - Pipeline 1 is split from Pipeline 0.
  - Pipeline 0 executes the processing of each element sequentially and cannot be preempted by another pipeline.
  - Pipeline 1 executes the processing of each element sequentially but can be preempted.
- **Pipelines elements description**
  - Camera element is configured for a specific pixel format and resolution (board dependent).
  - Display element is configured for a specific pixel format and resolution (board dependent).
  - 2D converts element on pipeline 0 is configured to perform:
    - color space conversion from the camera pixel format to the display pixel format.
    - rotation depending on the display orientation compared to the landscape mode.
      Note: *To get labels in the right orientation, the rotation is performed after the labeled-rectangle.*
  - 2D converts element on pipeline 1 is configured to perform:
    - color space conversion from the camera pixel format to RGB888.
    - cropping to maintain image aspect ratio.
    - scaling to 128 * 128 as mandated by the image classification model.
- The labeled rectangle element draws a crop window from which the camera image is sent to the ML inference element. The labeled rectangle element also displays the label of the object detected.
- The ML inference element runs an inference on the image pre-processed by the 2D convert element.
- The NULL sink element closes pipeline 1 (in MPP concept, only sink elements can close a pipeline).
- At every inference, the ML inference element invokes a callback containing the inference outputs. These outputs are post-processed by the callback client component. In this case, it is the main task of the application.

3.3 Output example

After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, Figure 7 shows the output of the `mpp_camera_mobilenet_view_tflm` example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.

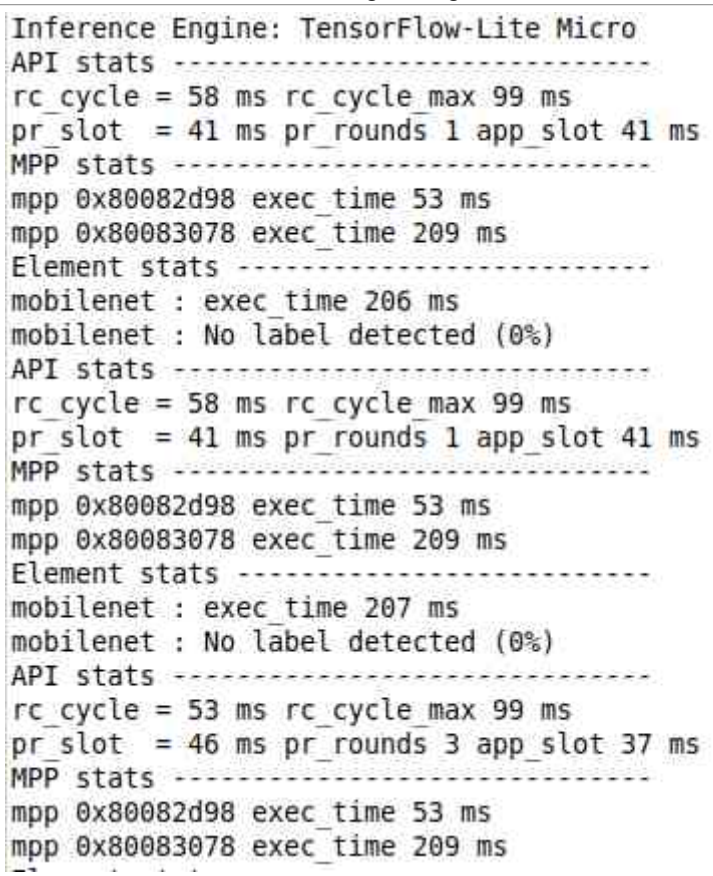

Figure 7. PuTTY console window

4 API references

4.1 Module documentation

This section provides information on:

- MPP API
- MPP types
- Return codes

4.1.1 MPP API

4.1.1.1 Functions

- int mpp_api_init (mpp_api_params_t *params)
- mpp_t mpp_create (mpp_params_t *params, int *ret)
- int mpp_camera_add (mpp_t mpp, const char name, mpp_camera_params_t *params,)

- int mpp_static_img_add (mpp_t mpp, mpp_img_params_t *params, void *addr)
- int mpp_display_add (mpp_t mpp, const char * name, mpp_display_params_t *params)
- int mpp_nullsink_add (mpp_t mpp)
- int mpp_element_add (mpp_t mpp, mpp_element_id_t id, mpp_element_params_t *params, mpp_elem_handle_t *elem_h)
- int mpp_split (mpp_t mpp, unsigned int num, mpp_params_t *params, mpp_t *out_list)
- int mpp_element_join (mpp_t *in_list, unsigned int num, mpp_element_id_t id, mpp_element_params_t *params, mpp_t out)
- int mpp_element_update (mpp_t mpp, mpp_elem_handle_t elem_h, mpp_element_params_t *params)
- int mpp_start (mpp_t mpp, int last)
- int mpp_stop (mpp_t mpp)
- void mpp_stats_enable (mpp_stats_grp_t grp)
- void mpp_stats_disable (mpp_stats_grp_t grp)
- char *mpp_get_version (void)

4.1.1.2 Detailed Description

This section provides the detailed documentation for the MCU Media Processing Pipeline API.

4.1.1.3 Function Documentation

4.1.1.3.1 mpp_api_init()

```
int mpp_api_init (mpp_api_params_t  *params)
```

Pipeline initialization.

This function initializes the library and its data structures.

It must be called before any other function of the API is called.

Parameters

in	*params*	API global parameters

Returns

Return_codes

4.1.1.3.2 mpp_create()

```
mpp_t mpp_create (mpp_params_t * params, int * ret)
```

Basic pipeline creation.

This function returns a handle to the pipeline.

Parameters

in	*params*	pipeline parameters
out	*ret*	return code (0 - success, non-zero - error)

Returns

A handle to the pipeline if success. NULL, if there is an error.

4.1.1.3.3 mpp_camera_add()

```
int mpp_camera_add (mpp_t mpp, const char name, mpp_camera_params_t * params)
```

Camera addition.

This function adds a camera to the pipeline.

in	*mpp*	input pipeline
in	*name*	camera driver name
in	*params*	parameters to be configured on the camera

Returns

Return_codes

4.1.1.3.4 mpp_static_img_add()

```
int mpp_static_img_add (mpp_t mpp, mpp_img_params_t params, void addr)
```

Static image addition.

Parameters

in	*mpp*	input pipeline
in	*params*	static image parameters
in	*addr*	image buffer

Returns

Return_codes

Preconditions

Image buffer allocation/free is the responsibility of the user.

4.1.1.3.5 mpp_display_add()

```
int mpp_display_add (mpp_t mpp, const char name, mpp_display_params_t params)
```

Display addition.

This function adds a display to the pipeline.

in	*mpp*	input pipeline
in	*name*	display driver name
in	*params*	parameters that are configured on the display

Returns

Return_codes

4.1.1.3.6 mpp_nullsink_add()

```
int mpp_nullsink_add (mpp_t mpp)
```

Null sink addition.

This function adds a null-type sink to the pipeline.

After this call pipeline is closed and no further elements can be added. Input frames are discarded.

in	*mpp*	input pipeline

Returns

Return_codes

4.1.1.3.7 mpp_element_add()

```
int mpp_element_add (mpp_t mpp, mpp_element_id_t id, mpp_element_params_t params,
 mpp_elem_handle_t elem_h)
```

Add processing element (single input, single output). This function adds an element to the pipeline.

Available elements are:

- 2D image processing
- ML inference engine
- Labeled rectangle
- Compositor

in	*mpp*	input pipeline
in	*id*	element id
in	*params*	element parameters
out	elem_h	element handle in pipeline

Returns

Return_codes

4.1.1.3.8 mpp_split()

```
int mpp_split (
  mpp_t mpp,
  unsigned int num,
  mpp_params_t * para,
  mpp_t out_list)
```

Pipeline multiplication.

Parameters

in	*mpp*	input pipeline
in	*num*	number of output pipeline
in	*params*	split mmp parameters

out	*out_list*	list of output pipelines

Returns

Return_codes

Preconditions

out_list array must contain at least *num* elements.

4.1.1.3.9 mpp_element_join()

```
int mpp_element_join (mpp_t in_list, unsigned
 int num, mpp_element_id_t id, mpp_element_params_t params, mpp_t out)
```

Join multiple pipelines through an element.

The element becomes a source for output pipeline.

Warning

NOT TESTED

Parameters

in	*in_list*	list of joined pipelines
in	*num*	number of pipelines in the list
in	*id*	element id
in	*params*	element params
out	*out*	output pipeline

Returns

Return_codes

4.1.1.3.10 mpp_element_update()

```
int mpp_element_update (mpp_t mpp, mpp_elem_handle_t elem_h, mpp_element_params_t params)
```

Update element parameters.

Parameters

in	*mpp*	input pipeline
in	*elem_h*	element handle in the pipeline.
in	*params*	new element parameters

Returns

Return_codes

4.1.1.3.11 mpp_start()

```
int mpp_start (mpp_t mpp, int last)
```

Start pipeline.

When called with last=0, this function prepares the branch of the pipeline specified with mpp. When called with last!=0, this function starts the data flow of the pipeline.

Data flow should start after all the branches of the pipeline have been prepared.

Parameters

in	*mpp*	pipeline branch handle to start/prepare
in	*last*	if non-zero start pipeline processing. No further start call is possible thereafter.

Returns

Return_codes

4.1.1.3.12 mpp_stop()

```
int mpp_stop (mpp_t mpp)
```

Stop a branch of the pipeline.

This function stops the data processing and peripherals of a pipeline branch.

Parameters

in	*mpp*	pipeline branch to stop

Returns

Return_codes

4.1.1.3.13 mpp_stats_enable ()

```
void mpp_stats_enable (mpp_stats_grp_t) grp
```

Enable statistics collection.

This function enables statistics collection for a given group Statistics collection is disabled by default after API initialization. Calling this function when stats are enabled has no effect.

Parameters

in	*grp*	Statistics group

4.1.1.3.14 mpp_stats_disable()

```
void mpp_stats_disable (mpp_stats_grp_t grp)
```

Disable statistics collection.

This function disables statistics collection for a given group Calling this function when stats are disabled has no effect. This function is used to ensure stats are not updated while application tasks use the stats structures.

Parameters

[in}	grp statistics group

4.1.1.3.15 mpp_get_version()

```
char mpp_get_version (void)
```

Get MPP version.

Returns

Pointer to the MPP version string.

4.1.2 MPP types

4.1.2.1 Data Structures

- struct mpp_params_t
- struct mpp_camera_params_t
- struct mpp_img_params_t
- struct mpp_display_params_t
- struct mpp_tensor_dims_t
- struct mpp_inference_out_tensor_param_t
- struct mpp_inference_cb_param_t
- union mpp_color_t
- struct mpp_color_t.rgb
- struct mpp_labeled_rect_t
- struct mpp_area_t
- struct mpp_dims_t
- struct mpp_position_t
- struct mpp_inference_params_t
- union mpp_element_params_t
- struct mpp_element_params_t.compose
- struct mpp_element_params_t.labels
- struct mpp_element_params_t.convert
- struct mpp_element_params_t.resize
- struct mpp_element_params_t.color_conv
- struct mpp_element_params_t.rotate
- struct mpp_element_params_t.test
- struct mpp_element_params_t.ml_inference

4.1.2.2 Macros

- #define MPP_INFERENCE_MAX_OUTPUTS
- #define MPP_INFERENCE_MAX_INPUTS
- #define MPP_APP_MAX_PRIO
- #define MPP_INVALID
- #define MPP_EVENT_ALL
- #define MAX_TENSOR_DIMS

4.1.2.3 Typedefs

- typedef void * mpp_t

- typedef uintptr_t mpp_elem_handle_t
- typedef unsigned int mpp_evt_mask_t
- typedef typedef int(* inference_entry_point_t) (uint8_t *, uint8_t *, uint8_t *)

4.1.2.4 Enumerations

- enum mpp_evt_t { MPP_EVENT_INVALID, MPP_EVENT_INFERENCE_OUTPUT_READY, MPP_EVENT_INTERNAL_TEST_RESERVED, MPP_EVENT_NUM }
- enum mpp_exec_flag_t { MPP_EXEC_INHERIT, MPP_EXEC_RC, MPP_EXEC_PREEMPT }
- enum mpp_rotate_degree_t { ROTATE_0, ROTATE_90, ROTATE_180, ROTATE_270 }
- enum mpp_flip_mode_t { FLIP_NONE, FLIP_HORIZONTAL, FLIP_VERTICAL, FLIP_BOTH }
- enum mpp_convert_ops_t { MPP_CONVERT_NONE, MPP_CONVERT_ROTATE, MPP_CONVERT_SCALE, MPP_CONVERT_COLOR, MPP_CONVERT_CROP, MPP_CONVERT_OUT_WINDOW}
- enum mpp_pixel_format_t { MPP_PIXEL_ARGB, MPP_PIXEL_RGB, MPP_PIXEL_RGB565, MPP_PIXEL_BGR, MPP_PIXEL_GRAY888, MPP_PIXEL_GRAY888X, MPP_PIXEL_GRAY, MPP_PIXEL_GRAY16, MPP_PIXEL_YUV1P444, MPP_PIXEL_VYUY1P422, MPP_PIXEL_UYVY1P422, MPP_PIXEL_YUYV, MPP_PIXEL_DEPTH16, MPP_PIXEL_DEPTH8, MPP_PIXEL_YUV420P, MPP_PIXEL_INVALID }
- enum mpp_element_id_t { MPP_ELEMENT_INVALID, MPP_ELEMENT_COMPOSE, MPP_ELEMENT_LABELED_RECTANGLE, MPP_ELEMENT_TEST, MPP_ELEMENT_INFERENCE, MPP_ELEMENT_CONVERT, MPP_ELEMENT_NUM }
- enum mpp_tensor_type_t { MPP_TENSOR_TYPE_FLOAT32, MPP_TENSOR_TYPE_UINT8, MPP_TENSOR_TYPE_INT8 }
- enum mpp_tensor_order_t { MPP_TENSOR_ORDER_UNKNOWN, MPP_TENSOR_ORDER_NHWC, MPP_TENSOR_ORDER_NCHW }
- enum mpp_inference_type_t { MPP_INFERENCE_TYPE_TFLITE, MPP_INFERENCE_TYPE_DEEPVIEWRT, MPP_INFERENCE_TYPE_GLOW}

4.1.2.5 Detailed Description

This section provides the detailed documentation for the MCU Media Processing Pipeline types.

4.1.2.6 Data Structure Documentation

4.1.2.6.1 union mpp_stats_t

Data Fields:

struct mpp_stats_t	api	Global execution performance counters.
struct mpp_stats_t	mpp	Pipeline execution performance counters.
struct mpp_stats_t	elem	Element execution performance counters.

4.1.2.6.2 struct mpp_stats_t.api

Data Fields:

unsigned int	rc_cycle	run-to-completion (RC) cycle duration (ms)

unsigned int	rc_cycle_max	run-to-completion work deadline (ms)
unsigned int	pr_slot	available slot for preemptable (PR) work (ms)
unsigned int	pr_rounds	number of RC cycles required to complete one PR cycle (ms)
unsigned int	app_slot	remaining time for application (ms)

4.1.2.6.3 struct mpp_stats_t.mpp

Data Fields:

mpp_t	mpp	
unsigned int	mpp_exec_time	pipeline execution time (ms)

4.1.2.6.4 struct mpp_stats_t.elem

Data Fields:

mpp_elem_handle_t	hnd	
unsigned int	elem_exec_time	element execution time (ms)

4.1.2.6.5 struct mpp_api_params_t

Data Fields:

mpp_stats_t *	stats	API stats
unsigned int	rc_cycle_min	minimum cycle duration for RC tasks (ms), 0: sets default value
unsigned int	rc_cycle_inc	time increment for RC tasks (ms), 0: sets default value

4.1.2.6.6 struct mpp_params_t

Pipeline creation parameters.

Data Fields

- int(∗ **evt_callback_f**)(mpp_t mpp, mpp_evt_t evt, void evt_data, void ∗user_data)
- mpp_evt_mask_t**mask**
- mpp_exec_flag_t**exec_flag**
- void * **cb_userdata**
- mpp_stats_t * stats

4.1.2.6.7 struct mpp_camera_params_t

Camera parameters.

Data Fields

int	height	buffer height

int	width	buffer width
mpp_pixel_format_t	format	pixel format
int	fps	frames per second

4.1.2.6.8 struct mpp_img_params_t

Static image parameters.

Data Fields

int	height	
int	width	
mpp_pixel_format_t	format	

4.1.2.6.9 struct mpp_display_params_t

Display parameters.

Data Fields

int	height	buffer resolution: setting to 0 will default to panel physical resolution
int	width	buffer resolution: setting to 0 will default to panel physical resolution
int	pitch	buffer resolution: setting to 0 will default to panel physical resolution
int	left	active rect: setting to 0 will default to fullscreen
int	top	active rect: setting to 0 will default to fullscreen
int	right	active rect: setting to 0 will default to fullscreen
int	bottom	active rect: setting to 0 will default to fullscreen
mpp_rotate_degree_t	rotate	rotate degree
mpp_pixel_format_t	format	pixel format

4.1.2.6.10 struct mpp_tensor_dims_t

Inference tensor dimensions.

Data Fields

uint32_t	size
uint32_t	data[MAX_TENSOR_DIMS]

4.1.2.6.11 struct mpp_inference_out_tensor_param_t

Tensor parameters.

Data Fields

const uint8_t *	data	output data
mpp_tensor_dims_t	dims	tensor data dimensions

mpp_tensor_type_t	type	tensor data type

4.1.2.6.12 struct mpp_inference_cb_param_t

Inference callback parameters.

Data Fields

void *	user_data	callback will pass this pointer
mpp_inference_out_tensor_params_t *	out_tensors [MPP_INFERENCE_MAX_OUTPUTS]	output tensors parameters
int	inference_time_ms	inference run time measurement - output to user
mpp_inference_type_t	inference_type	type of the inference

4.1.2.6.13 union mpp_color_t

MPP color encoding.

Data Fields

uint32_t	raw	Raw color.
struct mpp_color_t	rgb	rgb color values RGB color

4.1.2.6.14 struct mpp_color_t.rgb

RGB color values.

Data Fields

uint8_t	R	Red byte.
uint8_t	G	Green byte.
uint8_t	B	Blue byte.
uint8_t	pad	padding byte

4.1.2.6.15 struct mpp_labeled_rect_t

MPP labeled rectangle element structure.

Data Fields

uint8_t	label[64]	label to print
uint16_t	clear	clear rectangle
uint16_t	line_width	rectangle line thickness
mpp_color_t	line_color	rectangle line color
uint16_t	top	rectangle top position
uint16_t	left	rectangle left position
uint16_t	bottom	rectangle bottom position

uint16_t	right	rectangle right position
uint16_t	tag	labeled rectangle tag
uint16_t	reserved	pad for 32 bits alignment

4.1.2.6.16 struct mpp_area_t

Image area coordinates.

Data Fields

int	top	
int	left	
int	bottom	
int	right	

4.1.2.6.17 struct mpp_dims_t

Image dimensions.

Data Fields

unsigned int	width	
unsigned int	height	

4.1.2.6.18 struct mpp_position_t

Image dimensions.

Data Fields

unsigned int	top	
unsigned int	left	

4.1.2.6.19 struct mpp_inference_param_t

Processing element parameters.

Data Fields

uint64_t	constant_weight_MemSize	model constant weights memory size
uint64_t	mutable_weight_MemSize	Defines the amount of memory required both input & output data buffers.
uint64_t	activations_MemSize	Size of scratch memory used for intermediate computations needed by the model.
int	num_inputs	model's number of inputs
int	num_outputs	model's number of outputs
uint64_t	inputs_offsets[MPP_INFERENCE_MAX_INTPUTS]	offset of each input

uint64_t	outputs_offsets[MPP_INFERENCE_MAX_OUTPUTS]	offset of each output
inference_entry_point_t	model_entry_point	function called to perform the inference
mpp_tensor_type_t	model_input_tensors_type	type of input buffer

4.1.2.6.20 struct mpp_element_params_t

Data Fields:

union mpp_element_params_t		
mpp_stats_t *	stats	

4.1.2.6.21 union mpp_element_params_t

Processing element parameters.

Data Fields

struct mpp_element_params_t	compose	Compose element's parameters - NOT IMPLEMENTED YET.
struct mpp_element_params_t	labels	Labeled Rectangle element's parameters.
struct mpp_element_params_t	convert	Convert element's parameters.
struct mpp_element_params_t	resize	Resize element's parameters.
struct mpp_element_params_t	color_conv	Color convert element's parameters.
struct mpp_element_params_t	rotate	Rotate element's parameters.
struct mpp_element_params_t	test	Test element's parameters.
struct mpp_element_params_t	ml_inference	ML inference element's parameters.

4.1.2.6.22 struct mpp_element_params_t.compose

Compose element's parameters. NOT IMPLEMENTED YET.

Data Fields

float	a	
float	b	

4.1.2.6.23 struct mpp_element_params_t.labels

Labeled rectangle element's parameters.

Data Fields

uint32_t	max_count	maximum number of rectangles
uint32_t	detected_count	detected rectangles
mpp_labeled_rect_t *	rectangles	array of rectangle data

4.1.2.6.24 struct mpp_element_params_t.convert

Convert element's parameters.

Data Fields

mpp_dims_t	out_buf	buffer dimensions
mpp_pixel_format_t	pixel_format	new pixel format
mpp_rotate_degree_t	angle	rotation angle
mpp_area_t	crop	input crop area
mpp_dims_t	scale	scaling dimensions
mpp_position_t	out_window	output window position
mpp_convert_ops_t	ops	operation selector mask
const char*	dev_name	device name used for graphics

4.1.2.6.25 struct mpp_element_params_t.resize

Resize element's parameters.

Data Fields

unsigned int	width	
unsigned int	height	

4.1.2.6.26 struct mpp_element_params_t.color_conv

Color convert element's parameters.

Data Fields

mpp_pixel_format_t	pixel_format	

4.1.2.6.27 struct mpp_element_params_t.rotate

Rotate element's parameters.

Data Fields

mpp_rotate_degree_t	angle	

4.1.2.6.28 struct mpp_element_params_t.test

Test element's parameters.

Data Fields

_Bool	inp	
unsigned int	width	
unsigned int	height	
mpp_pixel_format_t	format	

4.1.2.6.29 struct mpp_element_params_t.ml_inference

ML inference element's parameters.

Data Fields

const void *	model_data	pointer to model binary
mpp_inference_type_t	type	inference type
int	model_size	model binary size (unused by GLOW)
float	model_input_mean	model 'mean' of input values, used for normalization
float	model_input_std	model 'standard deviation' of input values, used for normalization
mpp_tensor_order_t	tensor_order	model input tensor component order
mpp_int_params_t	inference_params	model specific parameters used by the inference

4.1.2.7 Macro Definition Documentation

4.1.2.7.1 MPP_INFERENCE_MAX_OUTPUTS

```
#define MPP_INFERENCE_MAX_OUTPUTS
```

Maximum number of outputs supported by the pipeline.

4.1.2.7.2 MPP_INFERENCE_MAX_INPUTS

```
#define MPP_INFERENCE_MAX_INPUTS
```

Maximum number of inputs supported by the pipeline.

4.1.2.7.3 MPP_INVALID

```
#define MPP_INVALID
```

Invalid pipeline handle.

4.1.2.7.4 MPP_APP_MAX_PRO

```
#define MPP+APP_MAX_PRIO
```

Maximum priority for application tasks Tasks created by the application should have a maximum priority otherwise scheduling of pipeline processing tasks may be impacted.

4.1.2.7.5 MPP_EVENT_ALL

```
#define MPP_EVENT_ALL
```

Bit mask to receive all events.

4.1.2.7.6 MAX_TENSOR_DIMS

```
#define TENSOR_DIMS
```

Maximum number of dimensions for tensors.

4.1.2.8 Typedef Documentation

4.1.2.8.1 mpp_t

```
typedef void mpp_t
```

Pipeline handle type.

4.1.2.8.2 mpp_elem_handle_t

```
typedef uintptr_t mpp_elem_handle_t
```

Element handle type.

4.1.2.8.3 mpp_evt_mask_t

```
typedef unsigned int mpp_evt_mask_t
```

Event mask for pipeline creation.

4.1.2.8.4 inference_entry_point_t

```
typedef int(* inference_entry_point_t) (uint8_t *, uint8_t *, uint8_t *)
```

Bundle inference function type.

4.1.2.9 Enumeration Type Documentation

4.1.2.9.1 mpp_evt_t

```
enum mpp_evt_t
```

Pipeline generated events.

Enumerator

MPP_EVENT_INVALID	invalid event
MPP_EVENT_INFERENCE_OUTPUT_READY	inference out is ready
MPP_EVENT_INTERNAL_TEST_RESERVED	INTERNAL.DO NOT USE.
MPP_EVENT_NUM	DO NOT USE.

4.1.2.9.2 mpp_exec_flag_t

```
enum mpp_exec_flag_t
```

Execution parameters.

These parameters control the execution of the elements of an mpp.

The "mpps" created using the flag MPP_EXEC_RC are guaranteed to run up to the completion of all processing elements, while not being preempted by other "mpps".

The "mpps" created using the flag MPP_EXEC_PREEMPT are preempted after a given time interval by "mpps" that will run-to-completion again.

The "mpps" created with the MPP_EXEC_INHERIT flag inherit the same execution flag as the parent(s) in case of split/join operation.

Note: *It is not possible to request run-to-completion execution when spliting/joining preemptable-execution "mpps".*

Enumerator

MPP_EXEC_INHERIT	inherit from parent(s)
MPP_EXEC_RC	run-to-completion
MPP_EXEC_PREEMPT	preemptable

4.1.2.9.3 mpp_stats_grp_t

Enumerator:

MPP_STATS_GRP_API	API (global) stats
MPP_STATS_GRP_MPP	mpp_t stats
MPP_STATS_GRP_ELEMENT	element stats
MPP_STATS_GRP_NUM	number of groups

4.1.2.9.4 mpp_rotate_degree_t

`enum mpp_rotate_degree_t`

Rotation value.

Enumerator

ROTATE_0	0 degree
ROTATE_90	90 degrees
ROTATE_180	180 degrees
ROTATE_270	270 degrees

4.1.2.9.5 mpp_flip_mode_t

`enum mpp_flip_mode_t`

Flip type.

Enumerator

FLIP_NONE	no flip
FLIP_HORIZONTAL	horizontal flip
FLIP_VERTICAL	vertical flip
FLIP_BOTH	vertical and horizontal flip

4.1.2.9.6 mpp_convert_ops_t

enum mpp_convert_ops_t

The convert operations selector flags.

Enumerator

MPP_CONVERT_NONE	no frame conversion
MPP_CONVERT_ROTATE	frame rotation
MPP_CONVERT_SCALE	scaling from input frame toward output window
MPP_CONVERT_COLOR	frame color conversion
MPP_CONVERT_CROP	input frame crop
MPP_CONVERT_OUT_WINDOW	output window

4.1.2.9.7 mpp_pixel_format_t

enum mpp_pixel_format_t

Pixel format.

Enumerator

MPP_PIXEL_ARGB	ARGB 32 bits.
MPP_PIXEL_RGB	RGB 24 bits.
MPP_PIXEL_RGB565	RGB 16 bits.
MPP_PIXEL_BGR	BGR 24 bits.
MPP_PIXEL_GRAY888	gray 3x8 bits
MPP_PIXEL_GRAY888X	gray 3x8 bits +8 unused bits
MPP_PIXEL_GRAY	gray 8 bits
MPP_PIXEL_GRAY16	gray 16 bits
MPP_PIXEL_YUV1P444	YUVX interleaved 4:4:4.
MPP_PIXEL_VYUY1P422	VYUY interleaved 4:2:2.
MPP_PIXEL_UYVY1P422	UYVY interleaved 4:2:2.
MPP_PIXEL_YUYV	YUYV interleaved 4:2:2.
MPP_PIXEL_DEPTH16	depth 16 bits
MPP_PIXEL_DEPTH8	depth 8 bits
MPP_PIXEL_YUV420P	YUV planar 4:2:0.
MPP_PIXEL_INVALID	invalid pixel format

4.1.2.9.8 mpp_element_id_t

enum mpp_element_id_t

Processing element ids.

Enumerator

MPP_ELEMENT_INVALID	Invalid element.
MPP_ELEMENT_COMPOSE	Image composition - NOT IMPLEMENTED YET.
MPP_ELEMENT_LABELED_RECTANGLE	Labeled rectangle - bounding box.
MPP_ELEMENT_TEST	Test inplace element - NOT FOR USE.
MPP_ELEMENT_INFERENCE	Inference engine.
MPP_ELEMENT_CONVERT	Image conversion: resolution, orientation, color format.
MPP_ELEMENT_NUM	DO NOT USE.

4.1.2.9.9 mpp_tensor_type_t

```
enum mpp_tensor_type_t
```

Inference tensor type.

Enumerator

MPP_TENSOR_TYPE_FLOAT32	floating point 32 bits
MPP_TENSOR_TYPE_UINT8	unsigned integer 8 bits
MPP_TENSOR_TYPE_INT8	signed integer 8 bits

4.1.2.9.10 mpp_tensor_order_t

```
enum mpp_tensor_order_t
```

Inference input tensor order.

Enumerator

MPP_TENSOR_ORDER_UNKNOWN	Order not set
MPP_TENSOR_ORDER_NHWC	Order: Batch, Height, Width, Channels
MPP_TENSOR_ORDER_NCHW	Order: Batch, Channels, Height, Width

4.1.2.9.11 mpp_inference_type_t

```
enum mpp_inference_type_t
```

Inference type.

Enumerator

MPP_INFERENCE_TYPE_TFLITE	TensorFlow-Lite
MPP_INFERENCE_TYPE_DEEPVIEWRT	DeepView RT
MPP_INFERENCE_TYPE_GLOW	GLOW

4.1.3 Return codes

4.1.3.1 Macros

- #define MPP_SUCCESS
- #define MPP_ERROR
- #define MPP_INVALID_ELEM
- #define MPP_INVALID_PARAM
- #define MPP_ERR_ALLOC_MUTEX
- #define MPP_INVALID_MUTEX
- #define MPP_MUTEX_TIMEOUT
- #define MPP_MUTEX_ERROR
- #define MPP_MALLOC_ERROR

4.1.3.2 Detailed Description

MPP APIs return status definitions.

4.1.3.3 Macro Definition Documentation

4.1.3.3.1 MPP_SUCCESS

```
#define MPP_SUCCESS
```

Success return code.

4.1.3.3.2 MPP_ERROR

```
#define MPP_ERROR
```

A generic error occured.

4.1.3.3.3 MPP_INVALID_ELEM

```
#define MPP_INVALID_ELEM
```

Invalid element provided.

4.1.3.3.4 MPP_INVALID_PARAM

```
#define MPP_INVALID_PARAM
```

Invalid parameter provided.

4.1.3.3.5 MPP_ERR_ALLOC_MUTEX

```
#define MPP_ERR_ALLOC_MUTEX
```

Error occured while allocating mutex.

4.1.3.3.6 MPP_INVALID_MUTEX

```
#define MPP_INVALID_MUTEX
```

Invalid mutex provided.

4.1.3.3.7 MPP_MUTEX_TIMEOUT

```
#define MPP_MUTEX_TIMEOUT
```

Mutex timeout occured.

4.1.3.3.8 MPP_MUTEX_ERROR

```
#define MPP_MUTEX_ERROR
```

Mutex error occured.

4.1.3.3.9 MPP_MALLOC_ERROR

```
#define MPP_MALLOC_ERROR
```

Memory allocation error occured.

5 Hardware Abstraction Layer

This is the documentation for the Hardware Abstraction Layer (HAL) API.

5.1 HAL overview

The hardware abstraction layer is used to abstract hardware and software components. With the usage of an HAL abstraction, the vision pipeline leverages hardware accelerated components, whenever possible.

5.1.1 MPP HAL description

The MPP HAL is presented regarding the following points:

- A common header file "hal.h" includes all hardware top-level functions.
- All hardware top-level functions are using the prefix: "hal_ "
- For each platform, all hal_ functions defined in hal.h should be implemented at least with an empty function.

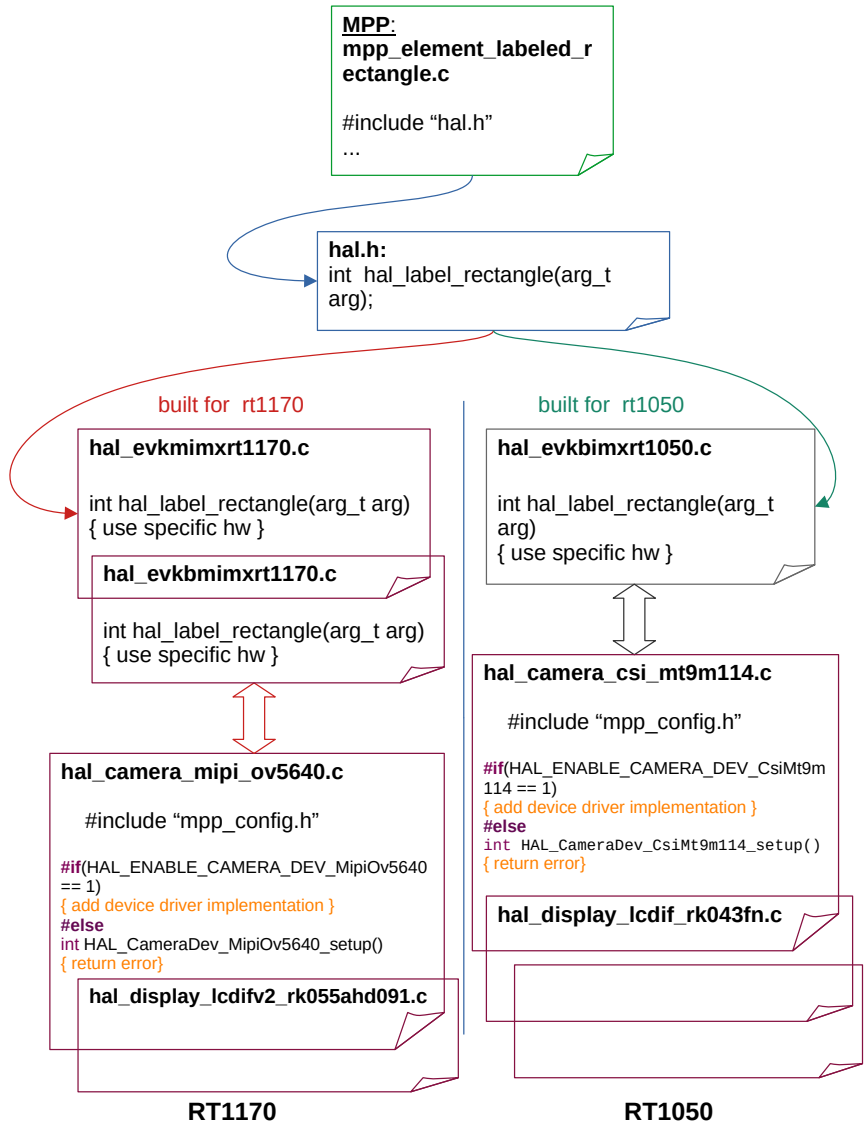

Figure 8. HAL overview

5.1.2 MPP HAL components

This section lists the source, processing, and sink elements.

5.1.2.1 Source elements

- Camera
- Static image

5.1.2.2 Processing elements

- Graphics driver
- Vision algorithms
- Labeled rectangle

5.1.2.3 Sink elements

- Display

5.1.3 Supported devices

At present, the MPP HAL supports the following devices:

- Cameras:
  - OV5640
  - MT9M114
  - OV7670
- Displays:
  - RK055AHD091
  - RK055MHD091
  - RK043FN02H-CT
  - Mikroe TFT Proto 5(SSD1963 controller)
- Graphics:
  - PXP
  - CPU

5.1.4 Supported boards

Currently, the MPP HAL supports the following boards:

- evkmimxrt1170: The evkmimxrt1170 is supported with the following devices:
  - Cameras: OV5640
  - Displays: RK055AHD091 and RK055MHD091
- evkbimxrt1050: The evkbimxrt1050 is supported with the following devices:
  - Cameras: MT9M114
  - Displays: RK043FN02H-CT
- evkbmimxrt1170: The evkbmimxrt1170 is supported by porting the following devices:
  - Cameras: OV5640
  - Displays: RK055AHD091 and RK055MHD091
- frdmmcxn947: The frdmmcxn947 is supported by porting the following devices:
  - Cameras: OV7670
  - Displays: Mikroe TFT Proto 5"

5.2 How to port new boards/devices

The MPP HAL provides the flexibility to the user to port new boards and devices. For example, cameras and displays.

5.2.1 Supporting new boards

To support a new board a new file `hal_{board_name}` should be added under the HAL directory.

5.2.2 Supporting new devices

The HAL components that can support new devices are:

- Cameras
- Display
- Graphics processing

To support a new device:

- Provide the appropriate hal_{device_module} implementation.
- Addthe name and setup entry point to the appropriate device list in the associated board hal_{board_name} file.

5.2.3 Enabling or disabling HAL components and devices

- The HAL components can be enabled/disabled from "mpp_config.h" using the compilation flags(HAL_ENABLE_{component_name}).
- The HAL devices can also be enabled/disabled from "mpp_config.h" using the compilation flags(HAL_ENABLE_{device_name}).

5.3 Module documentation

5.3.1 HAL types

5.3.1.1 Data Structures

- struct camera_dev_static_config_t
- struct camera_dev_private_capability_t
- struct camera_dev_t
- struct static_image_static_config_t
- struct static_image_t
- struct _gfx_surface
- struct _gfx_rotate_config
- struct gfx_dev_t
- struct _model_param_t
- struct _valgo_dev_private_capability
- struct _vision_frame
- struct vision_algo_dev_t
- struct display_dev_private_capability_t
- struct display_dev_t
- struct hw_buf_desc_t
- struct hal_graphics_setup_t
- struct hal_display_setup_t
- struct hal_camera_setup_t
- struct checksum_data_t

5.3.1.2 Macros

- #define HAL_GFX_DEV_CPU_NAME
- #define GUI_PRINTF_BUF_SIZE
- #define GUI_PRINTF_BUF_SIZE
- #define MAX_INPUT_PORTS
- #define **MAX_OUTPUT_PORTS**
- #define HAL_DEVICE_NAME_MAX_LENGTH

5.3.1.3 Typedefs

- typedef int(* camera_dev_callback_t) (const camera_dev_t *dev, camera_event_t event, void *param, uint8_t fromISR)
- typedef void * **vision_algo_private_data_t**
- typedef int(* mpp_callback_t) (mpp_t mpp, mpp_evt_t evt, void *evt_data, void *user_data)
- typedef int(* graphic_setup_func_t) (gfx_dev_t *)
- typedef int(* display_setup_func_t) (display_dev_t *)
- typedef int(* camera_setup_func_t) (const char *, camera_dev_t *)

5.3.1.4 Enumerations

- enum _hal_camera_status { kStatus_HAL_CameraSuccess, kStatus_HAL_CameraBusy, kStatus_HAL_CameraNonBlocking, kStatus_HAL_CameraError }
- enum _camera_event { kCameraEvent_SendFrame, kCameraEvent_CameraDeviceInit }
- enum _hal_image_status { MPP_kStatus_HAL_ImageSuccess, MPP_kStatus_HAL_ImageError }
- enum _gfx_rotate_target { **kGFXRotateTarget_None**, **kGFXRotate_SRCSurface**, **kGFXRotate_DSTSurface** }
- enum _hal_valgo_status { kStatus_HAL_ValgoSuccess, kStatus_HAL_ValgoMallocError, kStatus_HAL_ValgoInitError, kStatus_HAL_ValgoError, kStatus_HAL_ValgoStop }
- enum _display_event { kDisplayEvent_RequestFrame }
- enum _hal_display_status { kStatus_HAL_DisplaySuccess, kStatus_HAL_DisplayTxBusy, kStatus_HAL_DisplayNonBlocking, kStatus_HAL_DisplayError }
- enum mpp_memory_policy_e { HAL_MEM_ALLOC_NONE, HAL_MEM_ALLOC_INPUT, HAL_MEM_ALLOC_OUTPUT, HAL_MEM_ALLOC_BOTH }
- enum checksum_type_t { CHECKSUM_TYPE_PISANO, CHECKSUM_TYPE_CRC_ELCDIF }

5.3.1.5 Functions

- int HAL_GfxDev_CPU_Register (gfx_dev_t *dev)
- int **setup_static_image_elt** (static_image_t *elt)
- uint32_t **calc_checksum** (int size_b, void *pbuf)

5.3.1.6 Detailed Description

This section provides the detailed documentation for the MPP HAL types.

5.3.1.7 Data Structure Documentation

5.3.1.7.1 struct camera_dev_static_config_t

Structure that characterizes the camera device.

5.3.1.7.1.1 Data Fields

int	height	buffer height
int	width	buffer width
int	pitch	buffer pitch
int	left	left position
int	top	top position
int	right	right position
int	bottom	bottom position
mpp_rotate_degree_t	rotate	rotate degree
mpp_flip_mode_t	flip	flip
int	swapByte	swap byte per two bytes
mpp_pixel_format_t	format	pixel format
int	framerate	frame rate

5.3.1.7.2 struct camera_dev_private_capability_t

camera device private capability.

5.3.1.7.2.1 Data Fields

camera_dev_callback_t	callback	callback
void *	param	parameter for the callback

5.3.1.7.3 struct _camera_dev

Attributes of a camera device.

hal camera device declaration.

Camera devices can enqueue and dequeue frames as well as react to events from input devices via the "inputNotify" function. Camera devices can use any number of interfaces, including MIPI and CSI as long as the HAL driver implements the necessary functions found in camera_dev_operator_t. Examples of camera devices include the Orbbec U1S 3D SLM camera module, and the OnSemi MT9M114 camera module.

5.3.1.7.3.1 Data Fields

int	id	unique id which is assigned by camera manager during registration
char	name[HAL_DEVICE_NAME_MAX_LENGTH]	name of the device
const camera_dev_operator_t *	ops	operations
camera_dev_static_config_t	config	static configurations
camera_dev_private_capability_t	cap	private capability

5.3.1.7.4 struct static_image_static_config_t

Structure that characterize the image element.

5.3.1.7.4.1 Data Fields

int	height	buffer height
int	width	buffer width
int	left	left position
int	top	top position
int	right	right position
int	bottom	bottom position
mpp_pixel_format_t	format	pixel format

5.3.1.7.5 struct _static_image

Attributes of a an image element.

5.3.1.7.5.1 Data Fields

int	id	unique id which is assigned by image manager
const static_image_operator_t *	ops	operations
static_image_static_config_t	config	static configs
uint8_t *	buffer	static image buffer

5.3.1.7.6 struct _gfx_surface

Gfx surface parameters.

5.3.1.7.6.1 Data Fields

int	height	buffer height
int	width	buffer width
int	pitch	buffer pitch
int	left	left position
int	top	top position
int	right	right position
int	bottom	bottom position
int	swapByte	swap byte per two bytes
mpp_pixel_format_t	format	pixel format
void *	buf	buffer
void *	lock	the structure is determined by hal and set to null if not use in hal

5.3.1.7.7 struct _gfx_rotate_config

gfx rotate configuration

5.3.1.7.7.1 Data Fields

gfx_rotate_target_t	target	
mpp_rotate_degree_t	degree	

5.3.1.7.8 struct _gfx_dev

5.3.1.7.8.1 Data Fields

int	id	
const gfx_dev_operator_t *	ops	
gfx_surface_t	src	
gfx_surface_t	dst	
mpp_callback_t	callback	
void *	user_data	

5.3.1.7.9 struct _model_param_t

Structure passed to HAL as description of the binary model provided by user.

5.3.1.7.9.1 Data Fields

- const void * model_data
- int model_size
- float model_input_mean
- float model_input_std
- mpp_inference_params_t inference_params
- int height
- int width
- mpp_pixel_format_t format
- mpp_tensor_type_t inputType
- mpp_tensor_order_t tensor_order
- int(* evt_callback_f)(mpp_t mpp, mpp_evt_t evt, void *evt_data, void *user_data)
- void * cb_userdata

5.3.1.7.9.2 Field Documentation

const void* _model_param_t::model_data

pointer to model binary

int _model_param_t::model_size

model binary size

float _model_param_t::model_input_mean

model 'mean' of input values, used for normalization

float _model_param_t::model_input_std

model 'standard deviation' of input values, used for normalization

mpp_inference_params_t _model_param_t::inference_params

inference parameters

int _model_param_t::height

frame height

int _model_param_t::width

frame width

mpp_pixel_format_t _model_param_t::format

pixel format

mpp_tensor_type_t _model_param_t::inputType

input type

mpp_tensor_order_t _model_param_t::tensor_order

tensor order

int(* _model_param_t::evt_callback_f) (mpp_t mpp, mpp_evt_t evt, void *evt_data, void *user_data)

the callback to be called when model output is ready

void* _model_param_t::cb_userdata

pointer to user data, should be passed by callback

5.3.1.7.10 struct _valgo_dev_private_capability

Valgo devices private capability.

5.3.1.7.10.1 Data Fields

void *	param	param for the callback

5.3.1.7.11 struct _vision_frame

Characteristics that need to be defined by a vision algo.

5.3.1.7.11.1 Data Fields

int	height	frame height
int	width	frame width
int	pitch	frame pitch
mpp_pixel_format_t	format	pixel format
void *	input_buf	pixel input buffer

5.3.1.7.12 struct _vision_algo_dev

Attributes of a vision algo device.

5.3.1.7.12.1 Data Fields

int	id	unique id which is assigned by algorithm manager during the registration
char	name[HAL_DEVICE_NAME_MAX_LENGTH]	name to identify
valgo_dev_private_capability_t	cap	private capability
const vision_algo_dev_operator_t *	ops	operations
vision_algo_private_data_t	priv_data	private data

5.3.1.7.13 struct _display_dev_private_capability

Structure that characterizes the display device.

5.3.1.7.13.1 Data Fields

int	height	buffer height
int	width	buffer width
int	pitch	buffer pitch
int	left	left position
int	top	top position
int	right	right position
int	bottom	bottom position

mpp_rotate_degree_t	rotate	rotate degree
mpp_pixel_format_t	format	pixel format
int	nbFrameBuffer	number of input buffers
void **	frameBuffers	array of pointers to frame buffer
mpp_callback_t	callback	callback
void *	user_data	parameter for the callback

5.3.1.7.14 struct _display_dev

Attributes of a display device.

hal display device declaration.

Display devices can be used to display images, GUI overlays, etc. Examples of display devices include display panels like the RK024hh298 display, and external displays like UVC (video over USB).

5.3.1.7.14.1 Data Fields

int	id	unique id which is assigned by display manager during the registration
char	name[HAL_DEVICE_NAME_MAX_LENGTH]	name of the device
const display_dev_operator_t *	ops	operations
display_dev_private_capability_t	cap	private capability

5.3.1.7.15 struct hw_buf_desc_t

the hardware specific buffer requirements

5.3.1.7.15.1 Data Fields

int	stride	the number of bytes between 2 lines of image
int	nb	the number of lines required (set to 0 if the element does not require a specific number of lines)
int	alignment	alignment requirement in bytes
bool	cacheable	if true, HW will require cache maintenance
unsigned char *	addr	the buffer address

5.3.1.7.16 struct hal_graphics_setup_t

5.3.1.7.16.1 Data Fields

const char *	gfx_dev_name	
graphic_setup_func_t	gfx_setup_func	

5.3.1.7.17 struct hal_display_setup_t

5.3.1.7.17.1 Data Fields

const char *	display_name	
display_setup_func_t	display_setup_func	

5.3.1.7.18 struct hal_camera_setup_t

5.3.1.7.18.1 Data Fields

const char *	camera_name	
camera_setup_func_t	camera_setup_func	

5.3.1.7.19 struct checksum_data_t

computed checksum

5.3.1.7.19.1 Data Fields

checksum_type_t	type	checksum calculation method
uint32_t	value	checksum value

5.3.1.8 Macro Definition Documentation

5.3.1.8.1 #define HAL_GFX_DEV_CPU_NAME

hal graphics (gfx) device declaration.

Graphics processing devices can be used to perform conversion from one image format to another, resize images, and compose images on top of one another. Examples of graphics devices include the PXP (pixel pipeline) found on many i.MXRT series MCUs. Name of the graphic device using CPU operations

5.3.1.8.2 #define GUI_PRINTF_BUF_SIZE

Local text buffer size.

5.3.1.8.3 #define GUI_PRINTF_BUF_SIZE

Local text buffer size.

5.3.1.8.4 #define MAX_INPUT_PORTS

HAL public types header.

maximum number of element inputs/outputs

5.3.1.8.5 #define HAL_DEVICE_NAME_MAX_LENGTH

maximum length of device name

5.3.1.9 Typedef Documentation

5.3.1.9.1 typedef int(* camera_dev_callback_t) (const camera_dev_t *dev, camera_event_t event, void *param, uint8_t fromISR)

Callback function to notify camera manager that one frame is dequeued.

5.3.1.9.1.1 Parameters

dev	Device structure of the camera device calling this function
event	id of the event that took place
param	Parameters
fromISR	True if this operation takes place in an irq, 0 otherwise

5.3.1.9.1.2 Returns

0 if the operation was successfully

5.3.1.9.2 typedef int(* mpp_callback_t) (mpp_t mpp, mpp_evt_t evt, void *evt_data, void *user_data)

The mpp callback function prototype.

5.3.1.9.3 typedef int(* graphic_setup_func_t) (gfx_dev_t *)

graphics setup

5.3.1.9.4 typedef int(* display_setup_func_t) (display_dev_t *)

display setup

5.3.1.9.5 typedef int(* camera_setup_func_t) (const char *, camera_dev_t *)

camera setup

5.3.1.10 Enumeration Type Documentation

5.3.1.10.1 enum _hal_camera_status

Camera return status.

5.3.1.10.1.1 Enumerator

kStatus_HAL_Camera Success	HAL camera successful.

kStatus_HAL_Camera Busy	Camera is busy.
kStatus_HAL_Camera NonBlocking	Camera will return immediately.
kStatus_HAL_Camera Error	Error occurs on HAL Camera.

5.3.1.10.2 enum _camera_event

Type of events that are supported by calling the callback function.

5.3.1.10.2.1 Enumerator

kCameraEvent_Send Frame	Camera new frame is available.
kCameraEvent_Camera DeviceInit	Camera device finished the initialization process.

5.3.1.10.3 enum _hal_image_status

static image return status

5.3.1.10.3.1 Enumerator

MPP_kStatus_HAL_ ImageSuccess	Successfully.
MPP_kStatus_HAL_ ImageError	Error occurs on HAL Image.

5.3.1.10.4 enum _gfx_rotate_target

gfx rotate target

5.3.1.10.5 enum _hal_valgo_status

Valgo Error codes for hal operations.

5.3.1.10.5.1 Enumerator

kStatus_HAL_Valgo Success	Successfully.
kStatus_HAL_Valgo MallocError	memory allocation failed for HAL algorithm
kStatus_HAL_ValgoInit Error	algorithm initialization error
kStatus_HAL_Valgo Error	Error occurs in HAL algorithm.

kStatus_HAL_ValgoStop	HAL algorithm stop.

5.3.1.10.6 enum _display_event

Type of events that are supported by calling the callback function.

5.3.1.10.6.1 Enumerator

kDisplayEvent_RequestFrame	Display finished sending the frame asynchronously, provide another frame.

5.3.1.10.7 enum _hal_display_status

Error codes for display hal devices.

5.3.1.10.7.1 Enumerator

kStatus_HAL_DisplaySuccess	HAL display successful.
kStatus_HAL_DisplayTxBusy	Display tx is busy.
kStatus_HAL_DisplayNonBlocking	Display will return immediately.
kStatus_HAL_DisplayError	Error occurs on HAL Display.

5.3.1.10.8 enum mpp_memory_policy_e

The memory allocation policy of an element's hal.

During the pipeline construction, the HAL uses this enum to tell the pipeline if it already owns input/ouput buffers. Before the pipeline starts, the memory manager will map the existing buffers to elements and allocate missing buffers from the heap.

5.3.1.10.8.1 Enumerator

HAL_MEM_ALLOC_NONE	element requires buffers to be provided by other elements, or by the pipeline
HAL_MEM_ALLOC_INPUT	element allocates its input buffer, it may require output buffers to be provided by other elements, or by the pipeline
HAL_MEM_ALLOC_OUTPUT	element allocates its output buffer, it may require input buffers to be provided by other elements, or by the pipeline
HAL_MEM_ALLOC_BOTH	element allocates both its input and output buffers

5.3.1.10.9 enum checksum_type_t

checksum calculation method

5.3.1.10.9.1 Enumerator

CHECKSUM_TYPE_PISANO	checksum computed using Pisano
CHECKSUM_TYPE_CRC_ELCDIF	checksum computed CRC from ELCDIF

5.3.1.11 Function Documentation

5.3.1.11.1 int HAL_GfxDev_CPU_Register (gfx_dev_t * dev)

Register the graphic device with the CPU operations.

5.3.1.11.1.1 Parameters

in	*dev*	graphic device to register

5.3.1.11.1.2 Returns

error code (0: success, otherwise: failure)

5.3.2 HAL Operations

5.3.2.1 Data Structures

- struct _camera_dev_operator
- struct _static_image_operator
- struct gfx_dev_operator_t
- struct vision_algo_dev_operator_t
- struct _display_dev_operator

5.3.2.2 Typedefs

- typedef int(* mpp_callback_t) (mpp_t mpp, mpp_evt_t evt, void *evt_data, void *user_data)

5.3.2.3 Functions

- void GUI_DrawText (uint16_t *lcd_buf, uint16_t fcolor, uint16_t bcolor, uint32_t width, int x, int y, const char *label)
- void hal_draw_pixel (uint16_t *pDst, uint32_t x, uint32_t y, uint16_t color, uint32_t lcd_w)
- void hal_draw_text (uint16_t *lcd_buf, uint16_t fcolor, uint16_t bcolor, uint32_t width, int x, int y, const char *label)
- void hal_draw_rect (uint16_t *lcd_buf, uint32_t x, uint32_t y, uint32_t xsize, uint32_t ysize, uint32_t r, uint32_t g, uint32_t b, uint32_t width)
- static int get_bitpp (mpp_pixel_format_t type)
- Section 5.3.2.7.6 (uint8_t *data, int size)

5.3.2.4 Detailed Description

This section provides the detailed documentation for the MPP HAL operations that needs to be implemented for each component.

5.3.2.5 Data Structure Documentation

5.3.2.5.1 struct _camera_dev_operator

Operation that needs to be implemented by a camera device.

5.3.2.5.1.1 Data Fields

- hal_camera_status_t(* init)(camera_dev_t *dev, mpp_camera_params_t *config, camera_dev_callback_t callback, void *param)
- hal_camera_status_t(* deinit)(camera_dev_t *dev)
- hal_camera_status_t(* start)(const camera_dev_t *dev)
- hal_camera_status_t(* stop)(const camera_dev_t *dev)
- hal_camera_status_t(* enqueue)(const camera_dev_t *dev, void *data)
- hal_camera_status_t(* dequeue)(const camera_dev_t *dev, void **data, mpp_pixel_format_t *format)
- hal_camera_status_t(* get_buf_desc)(const camera_dev_t *dev, hw_buf_desc_t *out_buf, mpp_memory_policy_t *policy)

5.3.2.5.1.2 Field Documentation

5.3.2.5.1.3 hal_camera_status_t(* _camera_dev_operator::init) (camera_dev_t *dev, mpp_camera_params_t *config, camera_dev_callback_t callback, void *param)

initialize the dev

5.3.2.5.1.4 hal_camera_status_t(* _camera_dev_operator::deinit) (camera_dev_t *dev)

deinitialize the dev

5.3.2.5.1.5 hal_camera_status_t(* _camera_dev_operator::start) (const camera_dev_t *dev)

start the dev

5.3.2.5.1.6 hal_camera_status_t(* _camera_dev_operator::stop) (const camera_dev_t *dev)

stop the dev

5.3.2.5.1.7 hal_camera_status_t(* _camera_dev_operator::enqueue) (const camera_dev_t *dev, void *data)

enqueue a buffer to the dev

5.3.2.5.1.8 hal_camera_status_t(* _camera_dev_operator::dequeue) (const camera_dev_t *dev, void **data, mpp_pixel_format_t *format)

dequeue a buffer from the dev (blocking)

5.3.2.5.1.9 hal_camera_status_t(* _camera_dev_operator::get_buf_desc) (const camera_dev_t *dev, hw_buf_desc_t *out_buf, mpp_memory_policy_t *policy)

get buffer descriptors and policy

5.3.2.5.2 struct _static_image_operator

Operation that needs to be implemented by an image element.

5.3.2.5.2.1 Data Fields

- hal_image_status_t(* init)(static_image_t *elt, mpp_img_params_t *config, void *param)
- hal_image_status_t(* dequeue)(const static_image_t *elt, hw_buf_desc_t *out_buf, mpp_pixel_format_t *format)

5.3.2.5.2.2 Field Documentation

5.3.2.5.2.3 hal_image_status_t(* _static_image_operator::init) (static_image_t *elt, mpp_img_params_t *config, void *param)

initialize the elt

5.3.2.5.2.4 hal_image_status_t(* _static_image_operator::dequeue) (const static_image_t *elt, hw_buf_desc_t *out_buf, mpp_pixel_format_t *format)

dequeue a buffer from the elt

5.3.2.5.3 struct gfx_dev_operator_t

Operation that needs to be implemented by gfx device.

5.3.2.5.3.1 Data Fields

- int(* **init**)(const gfx_dev_t *dev, void *param)
- int(* **deinit**)(const gfx_dev_t *dev)
- int(* **get_buf_desc**)(const gfx_dev_t *dev, hw_buf_desc_t *in_buf, hw_buf_desc_t *out_buf, mpp_memory_policy_t *policy)
- int(* **blit**)(const gfx_dev_t *dev, const gfx_surface_t *pSrc, const gfx_surface_t *pDst, const gfx_rotate_config_t *pRotate, mpp_flip_mode_t flip)
- int(* **drawRect**)(const gfx_dev_t *dev, gfx_surface_t *pOverlay, int x, int y, int w, int h, int color)
- int(* **drawPicture**)(const gfx_dev_t *dev, gfx_surface_t *pOverlay, int x, int y, int w, int h, int alpha, const char *pIcon)
- int(* **drawText**)(const gfx_dev_t *dev, gfx_surface_t *pOverlay, int x, int y, int textColor, int bgColor, int type, const char *pText)

- int(* **compose**)(const gfx_dev_t *dev, gfx_surface_t *pSrc, gfx_surface_t *pOverlay, gfx_surface_t *pDst, gfx_rotate_config_t *pRotate, mpp_flip_mode_t flip)

5.3.2.5.4 struct vision_algo_dev_operator_t

Operation that needs to be implemented by a vision algorithm device.

5.3.2.5.4.1 Data Fields

- hal_valgo_status_t(* init)(vision_algo_dev_t *dev, model_param_t *param)
- hal_valgo_status_t(* deinit)(vision_algo_dev_t *dev)
- hal_valgo_status_t(* run)(const vision_algo_dev_t *dev, void *data)
- hal_valgo_status_t(* get_buf_desc)(const vision_algo_dev_t *dev, hw_buf_desc_t *in_buf, mpp_memory_policy_t *policy)

5.3.2.5.4.2 Field Documentation

5.3.2.5.4.3 hal_valgo_status_t(* vision_algo_dev_operator_t::init) (vision_algo_dev_t *dev, model_param_t *param)

initialize the dev

5.3.2.5.4.4 hal_valgo_status_t(* vision_algo_dev_operator_t::deinit) (vision_algo_dev_t *dev)

deinitialize the dev

5.3.2.5.4.5 hal_valgo_status_t(* vision_algo_dev_operator_t::run) (const vision_algo_dev_t *dev, void *data)

start the dev

5.3.2.5.4.6 hal_valgo_status_t(* vision_algo_dev_operator_t::get_buf_desc) (const vision_algo_dev_t *dev, hw_buf_desc_t *in_buf, mpp_memory_policy_t *policy)

read input parameters

5.3.2.5.5 struct _display_dev_operator

Operation that needs to be implemented by a display device.

5.3.2.5.5.1 Data Fields

- hal_display_status_t(* init)(display_dev_t *dev, mpp_display_params_t *config, mpp_callback_t callback, void *user_data)
- hal_display_status_t(* deinit)(const display_dev_t *dev)
- hal_display_status_t(* start)(display_dev_t *dev)
- hal_display_status_t(* stop)(display_dev_t *dev)
- hal_display_status_t(* blit)(const display_dev_t *dev, void *frame, int width, int height)
- hal_display_status_t(* get_buf_desc)(const display_dev_t *dev, hw_buf_desc_t *in_buf, mpp_memory_policy_t *policy)

5.3.2.5.5.2 Field Documentation

5.3.2.5.5.3 hal_display_status_t(* _display_dev_operator::init) (display_dev_t *dev, mpp_display_params_t *config, mpp_callback_t callback, void *user_data)

initialize the dev

5.3.2.5.5.4 hal_display_status_t(* _display_dev_operator::deinit) (const display_dev_t *dev)

deinitialize the dev

5.3.2.5.5.5 hal_display_status_t(* _display_dev_operator::start) (display_dev_t *dev)

start the dev

5.3.2.5.5.6 hal_display_status_t(* _display_dev_operator::stop) (display_dev_t *dev)

stop the dev

5.3.2.5.5.7 hal_display_status_t(* _display_dev_operator::blit) (const display_dev_t *dev, void *frame, int width, int height)

blit a buffer to the dev

5.3.2.5.5.8 hal_display_status_t(* _display_dev_operator::get_buf_desc) (const display_dev_t *dev, hw_buf_desc_t *in_buf, mpp_memory_policy_t *policy)

get buffer descriptors and policy

5.3.2.6 Typedef Documentation

5.3.2.6.1 typedef int(* mpp_callback_t) (mpp_t mpp, mpp_evt_t evt, void *evt_data, void *user_data)

The mpp callback function prototype.

5.3.2.7 Function Documentation

5.3.2.7.1 void GUI_DrawText (uint16_t * lcd_buf, uint16_t fcolor, uint16_t bcolor, uint32_t width, int x, int y, const char * label)

Draws text stored in label pointer to LCD buffer.

This function copy content of data from label text buffer to the LCD.

5.3.2.7.1.1 Parameters

lcd_buf	LCD buffer address destination for drawing text

fcolor	foreground color in rgb565 format
bcolor	background color in rgb565 format
width	LCD width
x	drawing position on X axe
y	drawing position on Y axe
format	C string pointed by format

5.3.2.7.1.2 Returns

The return number of written chars to the buffer

5.3.2.7.2 void hal_draw_pixel (uint16_t * pDst, uint32_t x, uint32_t y, uint16_t color, uint32_t lcd_w)

Draws pixel with RGB565 color to defined point.

5.3.2.7.2.1 Parameters

pDst	image data address of destination buffer
x	drawing position on X axe
y	drawing position on Y axe
color	RGB565 encoded value
lcd_w	lcd width

5.3.2.7.3 void hal_draw_text (uint16_t * lcd_buf, uint16_t fcolor, uint16_t bcolor, uint32_t width, int x, int y, const char * label)

Draws text stored in label pointer to LCD buffer.

This function copy content of data from label text buffer to the LCD.

5.3.2.7.3.1 Parameters

lcd_buf	LCD buffer address destination for drawing text
fcolor	foreground color in rgb565 format
bcolor	background color in rgb565 format
width	LCD width
x	drawing position on X axe
y	drawing position on Y axe
format	C string pointed by format

5.3.2.7.3.2 Returns

The return number of written chars to the buffer

5.3.2.7.4 void hal_draw_rect (uint16_t * lcd_buf, uint32_t x, uint32_t y, uint32_t xsize, uint32_t ysize, uint32_t r, uint32_t g, uint32_t b, uint32_t width)

Draws rectangle.

5.3.2.7.4.1 Parameters

in	*lcd_buf*	LCD buffer address destination for drawing rectangle
in	*color*	background color in rgb565 format
in	*x*	drawing position on X axe
in	*y*	drawing position on Y axe
in	*xsize*	rectangle width
in	*ysize*	rectangle height
in	*r*	0-255 red color value
in	*g*	0-255 green color value
in	*b*	0-255 blue color value
in	*width*	LCD width

5.3.2.7.4.2 Returns

N/A

5.3.2.7.5 static int get_bitpp (mpp_pixel_format_t type)[static]

returns the number of bits per pixel per format, unknown format return 0

5.3.2.7.6 void swap_2_bytes (uint8_t * *data*, int *size*)

Swaps a buffer's MSB and LSB bytes.

5.3.2.7.6.1 Parameters

data	pointer to the buffer to be converted(from little endian to big endian and vice-versa).
size	buffer size.

5.3.3 HAL setup functions

5.3.3.1 Functions

- int hal_label_rectangle (uint8_t *frame, int width, int height, mpp_pixel_format_t format, mpp_labeled_rect_t *lr)
- int hal_inference_tflite_setup (vision_algo_dev_t *dev)
- int hal_inference_glow_setup (vision_algo_dev_t *dev)
- int hal_inference_dvrt_setup (vision_algo_dev_t *dev)
- int hal_display_setup (const char *name, display_dev_t *dev)

- int hal_camera_setup (const char *name, camera_dev_t *dev)
- int hal_gfx_setup (const char *name, gfx_dev_t *dev)

5.3.3.2 Detailed Description

This section provides the detailed documentation for the HAL setup functions that should be defined by each device.

5.3.3.3 Function Documentation

5.3.3.3.1 int hal_label_rectangle (uint8_t * frame, int width, int height, mpp_pixel_format_t format, mpp_labeled_rect_t * lr)

Implementation of hal labeled rectangle component that draws a rectangle and a text on an input image.

5.3.3.3.1.1 Parameters

in	*frame*	The buffer address
in	*width*	Image width
in	*height*	Image height
in	*format*	Image format
in	*lr*	Labeled rectangle parameters

5.3.3.3.1.2 Returns

0

5.3.3.3.2 int hal_inference_tflite_setup (vision_algo_dev_t * dev)

Hal setup function for inference engine Tensorflow-Lite Micro.

5.3.3.3.2.1 Parameters

in	*dev*	vision algo device to register

5.3.3.3.2.2 Returns

error code (0: success, otherwise: failure)

5.3.3.3.3 int hal_inference_glow_setup (vision_algo_dev_t * dev)

Hal setup function for inference engine GLOW.

5.3.3.3.3.1 Parameters

in	*dev*	vision algo device to register

5.3.3.3.3.2 Returns

error code (0: success, otherwise: failure)

5.3.3.3.4 int hal_inference_dvrt_setup (vision_algo_dev_t * dev)

Hal setup function for inference engine DeepView RT.

5.3.3.3.4.1 Parameters

in	*dev*	vision algo device to register

5.3.3.3.4.2 Returns

error code (0: success, otherwise: failure)

5.3.3.3.5 int hal_display_setup (const char * name, display_dev_t * dev)

Register with a display device specified by name.

If name is NULL, return error.

5.3.3.3.5.1 Parameters

in	*name*	display name
in	*dev*	display device to register

5.3.3.3.5.2 Returns

error code (0: success, otherwise: failure)

5.3.3.3.6 int hal_camera_setup (const char * name, camera_dev_t * dev)

Register with a camera device specified by name.

If name is NULL, return error.

5.3.3.3.6.1 Parameters

in	*name*	camera name
in	*dev*	camera device to register

5.3.3.3.6.2 Returns

error code (0: success, otherwise: failure)

5.3.3.3.7 int hal_gfx_setup (const char * name, gfx_dev_t * dev)

Register with a graphic processing device specified by name.

If name is NULL, the first available graphic processing supported by Hw will be selected. The graphic device using CPU operations will be selected if name is not specified and if no graphic processing is available for the Hw.

5.3.3.3.7.1 Parameters

in	*name*	graphic processing device performing the image conversion
in	*dev*	graphic device to register

5.3.3.3.7.2 Returns

error code (0: success, otherwise: failure)

6 Revision history

Table 2 summarizes the changes done to this document since the initial release.

Table 2. Revision history

Revision number	Date	Substantive changes
0	30 June 2022	Initial release
1	06 September 2022	Updated for MCUXpresso SDK 2.12.1
2	08 December 2022	Updated for MCUXpresso SDK 2.13.0
3	27 July 2023	Updated for MCUXpresso SDK 2.14.0
4	10 January 2024	Updated for MCUXpresso SDK 2.15.000

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.

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.

Tables

Figures

Contents

For more information, please visit: https://www.nxp.com

Date of release: 10 January 2024
Document identifier: eIQ Media Processing Pipeline User's Guide