WTSD WTSD_OMAP4_MM_DUCATI_OMX_CAMERA_DESIGN

CompType OpenMAX Camera Component N.n OS Design Specification OMAP4

Document Revision: 0.5 DRAFT

Issue Date: 30 July 2009

Making**Wireless**

Table of Contents

List of Figures

List of Tables

Revision History

REV	DATE	AUTHOR	NOTES
0.1	07 Nov 2008	Devesh Pandey	Draft
0.2	05 Feb 2009	Sherin Sasidharan	Added/modified the component definition sections. Aligned on the number and nature of component ports.
0.3	12 Mar 2009	Sherin Sasidharan	Added camera internals section.
0.4	09 June 2009	Sherin Sasidharan	Partially updated OMX interface & incorporated LinuxMM teams' comments.
0.5	30 July 2009	Sherin Sasidharan	Updated deriving the intended use-case from OMX params, updated omx-indices, introduced omx component roles, introduced camera operating modes.

Plan Approvals

REV	APPROVAL 1	DATE	APPROVAL 2	DATE
	Person	dd Mmm YYYY	Person	dd Mmm YYYY

Please read the "Important Notice" on the next page.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

1 Products		2 Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

1 Introduction

The document represent the design of the OpenMAX Camera Component for Monica/OMAP4 platforms

1.1 Purpose

This document details the Design Specifications for OpenMAX Camera Component.

.

1.2 Scope

This document addresses the need of different ports in OpenMAX Camera Component, OpenMAX interface details, and internal design details and the behavior of OpenMAX Camera Component to different usecases. This document also discusses the multiple layers involved and how they interact to setup and achieve an usecase within the single component OpenMAX camera. For more OpenMAX related information from implementation perspective, please refer to the documents "wtsd_omx1.1_design_foils.ppt" and "wtsd_omx1.1_base_intfguide.doc" located \WTSD_DucatiMMSW\ omx\omx_il_1_x\omx_base\ docs\ .

1.3 File Path

This design specification document shall be captured in ClearCase® path defined in the project CM Plan:

The file path is WTSD_DucatiMMSW\ docs\design folder

1.4 File Name

The file name of this document is WTSD_OMAP4_MM_Ducati_OMX_Camera_Design.doc.

1.5 References

All References can be found on the Cellular Systems document repository (csdoc VOB) or the CSSD PMO Tools & Infrastructure web site.

1.6 Naming and Data Convention

This table below captures most of the acronyms used in this document.

Acronym	Description
ISIF	Image Sensor Interface

CSI-2	Camera Serial Interface 2
DPCM	Differential Pulse Code Modulation
FPS	frames per second
SIMCOP	Still Imaging coprocessor
IPIPE	Image Pipe
NSF	Noise filter
DPC	Defect Pixel Correction
LSC	Lens Shading Correction
MB	Megabytes
DDR-SDRAM	Double Data Rate - SDRAM
16 MP	16.6 MegaPixel – 4992x3328 pixels
CSI-2 RX	CSI-2 Receiver
CCPTx	CCP transmitter

2 Key Use Case Requirements

These requirements are driven for Monica which is applicable for OMAP4 also with some exceptions which would be mentioned explicity in place.

2.1 General requirements

The following are the key use case requirements that utilize the sensor.

1. Ability to support two separate sensors (simultaneously).
    a. One sensor shall be up to 16 MP (16:9) resolution. The use case descriptions will focus on a maximum sensor resolution of 16 MP (4:3).
    b. The second sensor can be up to 6 MP resolution
    c. Must be able to receive data from both sensors simultaneously and store them into SDRAM. This is to support stereo video and imaging.
2. Ability to support hardware features such as flash, mechanical shutter, neutral density filter, etc consistent with the support provided by the Xena.
3. Ability to control the auto exposure, auto focus and auto white balance (3A) of the sensor data. This includes automatic control of each of these as well as host configurable manual settings and macro modes of operation for each (such as night, sport, hyperfocal, etc) consistent with the support provided by the Xena.
4. Ability to prioritize focus, exposure, and white balance based on face detection and tracking. Ability to prioritize bitrate allocation based on face detection and tracking for video record use cases.
5. Ability to correct lens and optical shading using ISP5 hardware with no loss of performance.
6. Ability to suppress false color using ISP5 hardware with no loss of performance.
7. Ability to enhance edges using ISP5 hardware with no loss of performance.
8. Ancillary data must be attached to each frame of image data depending on the request. transferred to the host over the CCP2 or CSI2 interface in case of Monica. This includes all frames during preview and video encode.
9. Digital zoom capability which includes crop from 1x to 16x then resize (upsize or downsize) to the desired output resolution.
    a. All cropping stated as ‘nx’ is linear and is with respect to the full field-of-view of the sensor, independent of exactly how many pixels are being provided by the sensor. This is a linear reduction, not area. The horizontal and vertical dimensions are reduced by 1/n in the cropped image compared to the field of view of the sensor.
    b. Resizing is applied to the cropped image to bring it to the required output resolution.
    c. For still capture, the output resolution can be as large as the full sensor resolution
    d. For video capture, the output resolution is the size of the video to be captured
    e. The Monica/OMAP4 IPIPE resizer supports up to 16x maximum resize
        i. For still image capture, it is necessary to store the raw image in SDRAM and run the IPIPE using input from SDRAM because it is not possible to resize to 16x when running at 200 MP/s. However, such storage is not necessary if the sensor can crop and slow down the pixel rate. The maximum pixel rate that can be supported for horizontal and vertical upsampling ratios of H and V respectively is

200/ (H * V)

ii. For video capture from a 16MP sensor, 16x upsize is more than sufficient to generate all output video resolutions up to 1080p. To achieve the desired frame rate however, it will be necessary to downsample the image in the sensor.

iii. For purpose of analysis, sensors are assumed to support cropping and reasonable binning and/or skipping so that multiple passes through the resizer are not required. If multiple passes are needed for any particular sensor, imaging and video performance will be affected when using that sensor.

2.2 Still Image Recorder

The still record sequence begins with the host putting Monica into still preview mode such that Monica transfers preview frames (up to WVGA resolution) to the host over the CCP2 or CSI2 interface.No transfer over CCP2 or CSI2 is involved in OMAP4 case. The preview frames are displayed as a viewfinder for the user. Preview shall NOT be displayed on the TV out. If a demo in the store is desired, the encoded images may well be decoded and displayed on the TV. The user then initiates an image capture sequence through the appropriate host-application-specific user interface. The user-generated request is interpreted by the host and relayed to Monica.

Preview

The following requirements must be satisfied during still preview.

1. The preview output to the host must match the content that will be captured. This means that the cropping that will be applied to the frame that will be captured is also applied to each preview frame. This is essential so that the user can properly visualize the content to be captured and will not be ‘surprised’ by the result.
2. Preview frames are to be rotated properly before being sent to the host processor, such that no rotation of the preview frames is required by the host. This is independent of the phone/camera orientation. It is rather a function of the mechanical orientation of the sensor with respect to the display.
3. The preview frames must be resized as requested by the host before being sent to the host processor such that no resizing of the preview frames is required by the host.
4. The color format of the preview frames must be converted as per host’s request to one of the supported formats (YUV422, YUV420, RGB565, and RGB888) so that the host does not have to perform a color format conversion before sending the preview to the LCD.
5. If High Quality mode is signaled, Lens Distortion Correction must be applied to the preview. No other image enhancement algorithms need to be applied to the preview. High Quality mode is defined in the capture section.
6. The combination of the above requirements means that the resolution and content of the cropped, resized, rotated, preview frames is configured by the host such that the host can transfer the preview frames directly to the display without any required host image processing.

7. Once in still preview mode, a quick transition into shutter-half-press mode and image capture mode must be possible. This means that still preview, shutter-half-press, and image capture must be part of the same application build.
8. Preview output shall continue seamlessly during shutter-half-press if temporal bracketing is not commanded.
9. Preview output shall be the same as postview during shutter-half-press when temporal bracketing is commanded.
    a. A resized version of the captured frame in the resolution of the preview is called postview and is sent with every captured image. This serves as a viewfinder although it is actually a postview.
    b. During temporal bracketing, the preview framerate corresponds to the shot rate which is 10 sps for 16MP images.
    c. Preview in the original requested frame rate shall resume upon exit from temporal bracketing.
10. Preview output can be suspended during the image capture operation.
    a. This includes not only while Monica is receiving the frame that is captured but also while Monica is processing the captured frame.
    b. A preview representation of the captured frame must be provided to the host upon request.
    c. Preview could have automatically resume after the captured image is processed and transferred to the host.No transfer to Host in case of OMAP4. Or preview could resume once user triggers it.
    d. Resumption of preview between captured images in a multi-shot sequence is not required. If preview between captured images is resumed, it will most likely be at whatever frame rate is possible with the sensor in the mode required for image capture.
11. The worst case (most demanding) preview performance is WVGA resolution at 30 fps with rotation and Lens Distortion Correction enabled.

Shutter-Half-Press

Camera is put into shutter-half-press mode by the host. The following requirements must be satisfied

during shutter-half-press:

Camera shall support both locking and tracking of H3A and Faces during half press

If temporal bracketing is requested:

a. Camera shall reconfigure the sensor to output full size images.
b. Camera shall setup mechanical or electronic shutter to capture full size images at 10 shots per second.
c. Camera shall process the captured images and store in SDRAM the compressed JPEGs corresponding to the 5 most recent images.
d. In case of Monica, Ducati side shall transmit post view images as explained in the section on preview.

Capture

Still capture implies one of two things.

1. Unencoded image capture, where the unencoded image is transferred to the host via the CCP2 or CSI2 interface.No transfer to host via CCP2/CSI2 in OMAP4 case. This unencoded image can be either the raw sensor data or it can be the sensor data processed through the Monica/OMAP4 IPIPE. This scenario is NOT typically considered to be the worst case scenario since the throughput will be limited by the speed of the CCP2 or CSI2 interface to the host.
2. Encoded image capture, where the sensor data is processed through the IPIPE then JPEG encoded prior to being sent to the host via the CCP2 or CSI2 interface. No transfer to host via CCP2/CSI2.
    a. This includes the ability to capture and embed a thumbnail in the JPEG file, which is a small resolution representation of the captured image. The thumbnail can be optionally either unencoded or JPEG encoded. As stated in the JFIF specification, the maximum thumbnail resolution is 255x255 since the JFIF horizontal and vertical size parameters for the thumbnail are each 8 bit parameters
    b. No host processing of the JFIF/EXIF file from Monica will be required. The host therefore only performs the file store function.
    c. The host must be able to send any additional data to Camera component that is required to generate the JFIF/EXIF file such that the only task to be performed by the host is to store the file

Features that must be supported during both high speed and high quality image capture include the

following.

1. Monica must be able to apply host-specified cropping, rotation and resizing to the full size image from the sensor
    a. Rotation of 0°, 90°, 180° and 270° as well as mirror (180° rotation about vertical axis) and flip (180° rotation about horizontal axis). This rotation is independent of the rotation required for still preview.
    b. Resizing will never exceed the full size of the imager.
    c. The maximum cropping to be supported is 16x (in each dimension)
    d. The maximum upsize is 16x (in each dimension)
    e. This means that the full size image from the sensor can be cropped to a size that is $1/20^{th}$ of the original height and $1/20^{th}$ of the original width then upsized up to the original full size resolution of the sensor.
    f. Fine stepping of resizing factor is supported to mimic effect of optical zoom. ISP resizing factor is specified as 256/N, which specifies the ratio of the output dimension to the input dimension.
    g. The same rotation is applied to the captured image and to the thumbnail. This rotation is independent of the rotation applied to the preview frames.
2. Monica must apply edge enhancement and false color suppression to the captured images.
3. Monica must be able to apply LDC to the image from the sensor, when LDC is enabled by the host. LDC will also be applied to the thumbnail image.
4. Monica must be able to support an as yet undetermined software image enhancement prior to the hardware IPIPE to compensate for unanticipated lens or sensor anomalies. This algorithm

must run on the ARM and/or the iMX and the performance is limited by the complexity of the algorithm and the processing power available with the ARM and/or iMX.

5. Monica must be able to apply multi-pass processing to the captured image and thumbnail to assure that the maximum compressed file size (specified by the host) is not exceeded. It is anticipated that this feature will only be used in single shot capture mode due to the time it takes to process a single image multiple times.
6. In High Speed mode, Monica must be able to capture up to 10 shots per second indefinitely from a 16 MP sensor while operating the sensor at the 200 MP/s pixel readout rate. To achieve 10 shots per second, some features shall be restricted as below. This speed optimized mode is called “High Speed Mode”.
    a. Rotation is done on-the-fly in JPEG domain. However, the output does not need to be touched in any way by the host.
    b. Only JPEG (YUV420) format is supported because of the b/w limitations of the CCP2 and CSI2 interfaces used to send postview and captured to the host.
    c. Dark Frame Subtraction, Chromatic Aberration correction, Lens Distortion Correction, and High ISO Noise Filter are unavailable because of the high complexity.
    d. Lens Shading Correction, False Color Suppression, and Edge Enhancement are fully available with no restrictions.
7. In High Quality mode, Monica shall achieve a processing time of 1 second from the completion of readout to the start of JPEG file transfer over CCP2 or CSI2 with the following features. This enhanced feature set that optimizes for image quality is called “High Quality Mode.”
    a. Normal rotation is done with the entire image in memory.
    b. **Chromatic Aberration Correction** is done by the SIMCOP on raw Bayer data.
    c. Lens Distortion Correction is done by the SIMCOP on YUV data after ISP5 processing.
    d. High ISO Noise Filter is applied if necessary on YUV data.
    e. Up to 3 iterations of JPEG encoding are performed to control the size of the encoded file.
    f. Lens Shading correction, False Color Suppression, and Edge Enhancement are also done.
8. Monica shall be able to do multi-frame image stabilization when at least 64 MB of memory is available. It is expected that multi-frame image stabilization will take about 2 seconds after all the images have been captured.
9. Selectable capture and thumbnail data format including but not limited to YUV420, YUV422, RGB565, RGB888, BAYER and JPEG. Some thumbnail format limitations may be imposed by the design of the CCP transmitter in case of monica. These limitations have been experienced on Monica’s predecessors in that the thumbnail format has been limited to YUV422 and JPEG only.
10. Host configurable settings for both unencoded and encoded image capture including the following.
    a. Independent configuration of height and width of the image to capture.
    b. Configurable cropping of the full-size sensor image such that the cropped image can be resized to the configured output height and width. The cropped aspect ratio, the preview aspect ratio and the thumbnail aspect ratio must match the capture aspect ratio.
    c. Configurable effects including blemish correction, noise reduction, grey scale, sepia, negative, natural, vivid, contrast, sharpness, brightness, saturation, etc. Specifically, it

must have the ability to support all of these effects plus future application-specific effects.

d. Configurable rotation of 0, 90, 180 or 270 degrees as well as mirror (rotation about vertical axis) and flip (rotation about horizontal axis)

e. Number of shots to capture and the interval between the shots.

f. Control of a flash including flash selection, flash intensity, flash count, flash interval, flash duration and auto focus flash assistance.

11. Host configurable settings during encoded (JPEG) image capture including the following.

    a. Independently configurable image and thumbnail encode quality.

    b. Independently configurable encoded image and thumbnail maximum size (bytes).

    c. Application of an overlay to a portion of the image (for example, a date/time stamp). This is only applied to the captured image and the thumbnail. It is not applied to the preview frames.

Data Flow diagram

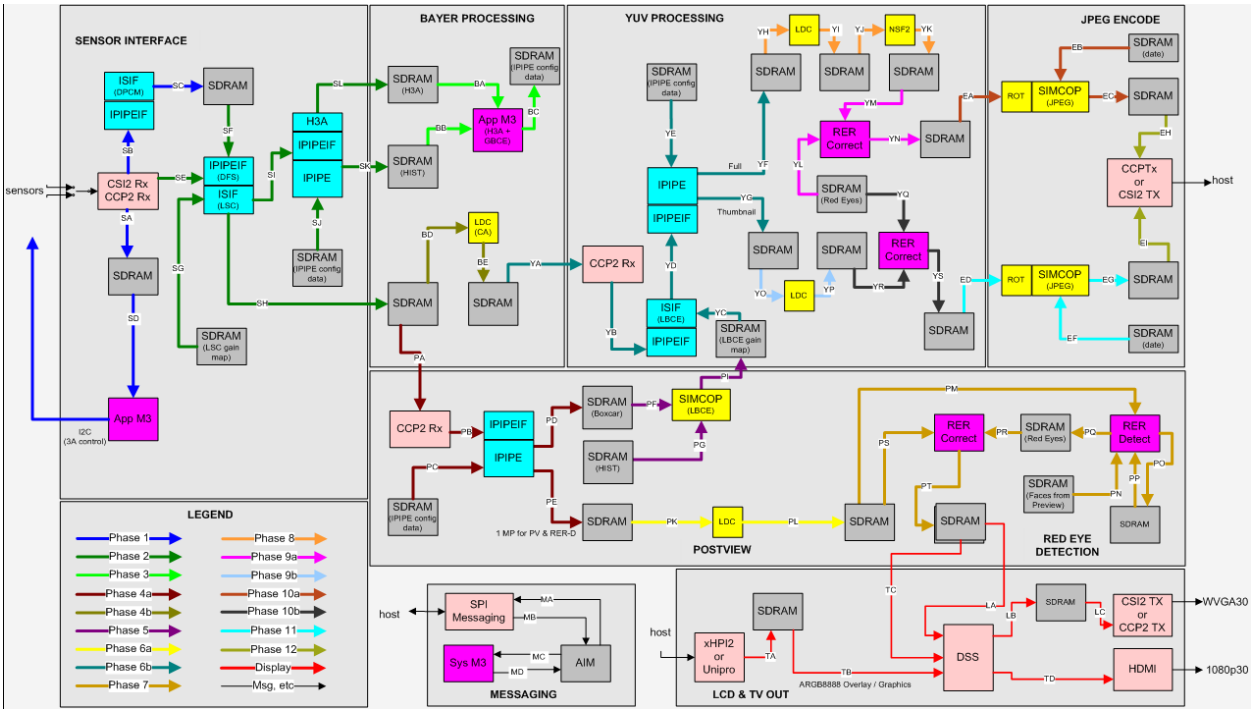


Figure 1 Advanced (High Quality) Still Image Recorder

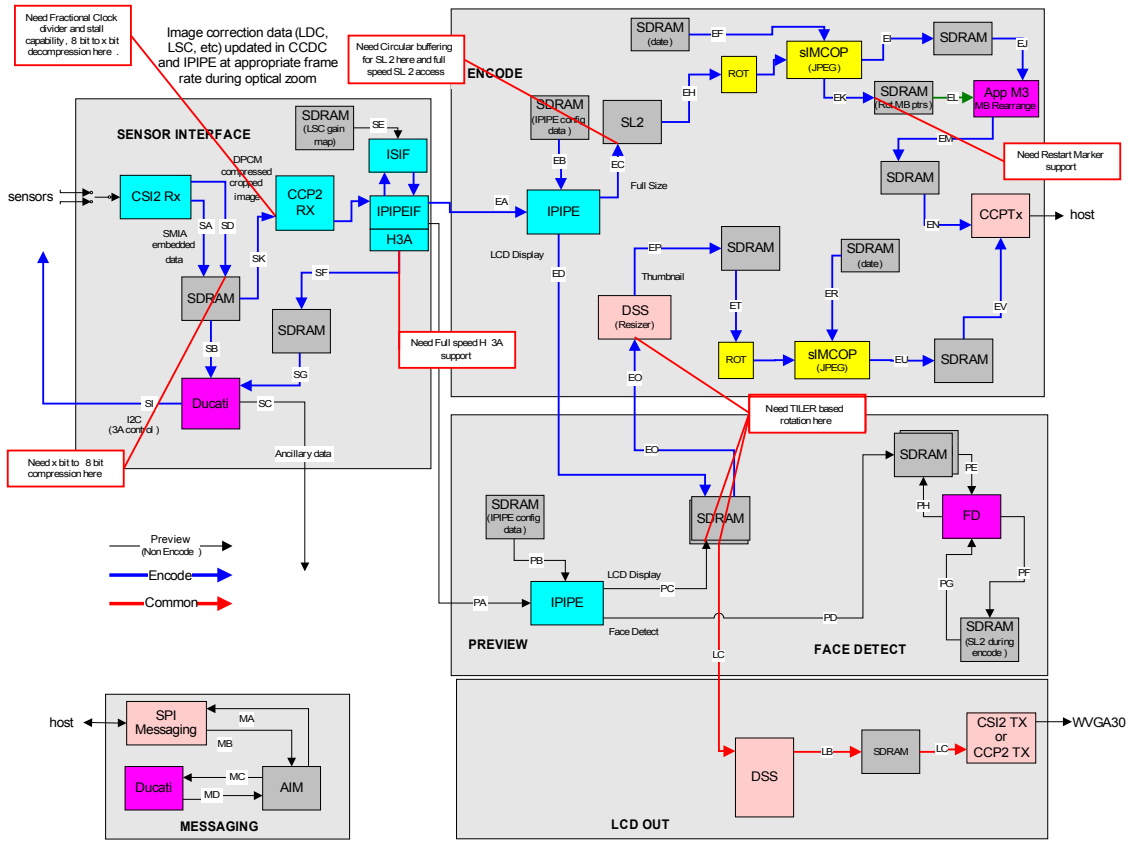


Figure 2 High Speed Still Image Recorder

2.3 Video Recorder

The video record sequence begins with the host putting Monica into video preview mode such that Monica transfers preview frames up to WVGA resolution to the host over the CCP2 or CSI2 interface. The

preview frames are displayed as a viewfinder for the user. Preview shall NOT be displayed on the TV. If a demo in the store is desired, the encoded video may well be decoded and displayed on the TV. The user then initiates video capture sequence through the appropriate host-application-specific user interface. The user-generated request is interpreted by the host and relayed to Monica.

1.1.1 Preview

The following requirements must be satisfied during video preview.

1. The preview output must match the content that will be captured. This means that the cropping that will be applied to the video that will be captured is also applied to each preview frame. This is essential so that the user can properly visualize the content to be captured and will not be 'surprised' by the result.
2. Preview frames are to be rotated as requested before being sent to the host processor so that the host does not need to rotate. This is independent of the phone/camera orientation. It is rather a function of the mechanical orientation of the sensor with respect to the display.
3. The preview frames must be resized to the size requested by the host before being sent to the host so that the host does not need to resize.
4. The color format of the preview frames must be converted as per host's request to one of the supported formats (YUV422, YUV420, RGB565, and RGB888) so that the host does not have to perform a color format conversion before sending the preview to the LCD.
5. The preview output must be stabilized to remove normal hand motion introduced while walking. The host must be able to enable/disable stabilization. The effect of stabilization during video preview must be identical to stabilization during video capture. Note that the process of stabilization requires some 'border' pixels around the region of interest that are no longer fully available for encoding. In other words, it is not possible to video encode the entire field-of-view of the sensor with stabilization enabled.
6. The combination of the above requirements means that the resolution and content of the cropped, resized, rotated, color format converted, and stabilized preview frames are configured by the host such that the host can transfer the preview frames directly to the display without any required host image processing.
7. Lens Shading Correction shall be applied to the preview.
8. False Color Suppression and Edge Enhancement shall be applied to the video.
9. Lens Distortion Correction shall be applied to the preview.
10. Video Noise Filter shall be applied to the video.
11. Once in video preview mode, a transition into video capture mode shall be possible. This means that video preview and video capture must be part of the same application build.
12. Preview output must continue during video capture. There must be no interruption in the video preview output when video capture is initiated.
13. The worst case (most demanding) preview performance is WVGA at 30 fps with 1080p30 video encoding with Rotation, LDC, Video Noise Filter, and Video Stabilization enabled.

1.1.2 Record

Video capture only applies to encoded video capture since unencoded video is the same as video preview. Features that must be supported during video capture include the following.

1. **Lens Shading Correction shall be applied prior to encoding.**
2. **False Color Suppression and Edge Enhancement shall be applied prior to encoding.**
3. **Lens Distortion Correction shall be applied prior to encoding.**
4. **Video Noise Filter shall be applied prior to encoding.**
5. **Monica must use face detection and face tracking algorithms to allocate more bitrate to faces and smoothly vary the bitrate around faces. Face optimized bitrate control must still meet all other bitrate and HRD requirements.**
6. Monica must send the encoded video bitstream to the host via the CCP2 TX, CSI2 TX or Unipro. In OMAP4 case this ia not valid.
7. Monica must attach the relative timing information to each encoded video frame sent to the host such that the host can perform audio/video synchronization. The technique currently used in the Xena is recommended since it is already proven. The method requires that Monica run a clock from which the time attached to each encoded frame is obtained. Monica also provides this clock value to the host periodically such that the host can maintain and update information that lets the host correlate the Monica clock value with its own clock value. With this information, the host can then translate the time at which an encoded frame occurred from the Monica time reference into the host time reference.
8. Host configurable settings for both unencoded and encoded video capture including the following.
    a. Independent configuration of height and width of the video to capture.
    b. Configurable cropping of the full-size sensor image such that the cropped video can be resized to the configured output height and width. The cropped aspect ratio and the preview aspect ratio must match the capture aspect ratio.
    c. Configurable effects including contrast, sharpness, brightness, saturation, etc.
    d. Configurable rotation of 0°, 90°, 180° or 270° degrees as well as mirror (rotation about vertical axis) and flip (rotation about horizontal axis)
    e. Control of an LED (illumination or indicator LED) including LED selection, LED intensity, flash count, flash interval and flash duration.
9. During video record, Monica shall support capturing a full size still image and compressing it using JPEG. Monica shall reconfigure the sensor to capture the full size image and then reconfigure it back for the video that was being encoded. The number of frames of video dropped is sensor dependent. The full size image shall be processed like in High Speed Still Image Capture mode. This is to minimize disruption of video. The video encoding shall start as soon as possible with an I-frame. The timestamps will account for the disruption of video so that A/V sync remains intact.
10. high speed video capture should be supported @ 120fps

1.1.3 Data Flow Diagram

The following data flow diagram shows a possible implementation for the H.264 video recorder. An H.263 or MPEG4 implementation would be similar, differing only in the encode algorithm within IVA-HD and the semantics of context data and reference storage.

3 OpenMAX Camera component definition

Existing OpenMAX IL 1.1.2 would need additional aspects to take care for catering to the high end cameras. This would need extending the 1.1.2 standard for fitting to our needs in terms of number of ports command indices, data starutures.

3.1 Need of different Ports in OpenMAX Camera Component

.

1.Memory to memory operation is used for Raw data processing, Statistic data collection. it is also used by SIMCOP. we need an **input buffer port** so that the input buffer can be shared (used for Raw data processing/Statistic data collection). Even uncompressed video frames also has to be supported. Thus this is a video domain port.
2. Need for **preview output port (video domain)** : dedicated port is required, usecases: vide finder and capture together. Always view finder is ON. Frame rate notion is needed here. Along with this preview data there is a need to append meta-data like, histogram and face detect information.
2.**An input time port is needed** for synchronization. Domain of this port would be OTHER.
3. This single OMX_CAM component shall support interface for Image/Video Encoder. This data could go out as captured data. Thus, need **1st Capture output port (image domain)**. Formats could be JPEG/YUV/RAW. Support for JPEG makes it to be in image domain. In video domain, there is no JPEG format.
4. Simultaneous request of different format output is a requirement, for this we would need a **2nd Capture output port (video domain)**. This port is meant for video capture. Formats supported could be YUV/RAW. This port won't be able to support JPEG in a native way. Please refer to the
5. It can handle applications such as face detection, object recognition, histogram data, distance measurement, etc. These data can be put together as in-band metadata along with the same capture port. That would increase

the data volume for that port. Capture port has to be handled in higher priority than the measurement and preview data. Which leads to separate the measurement data as separate port. **Measurement port**.
6. Still image capture during video record use-case. This would require preview port, video capture port, and image capture port.

3.2 OpenMAX Camera Component Diagram

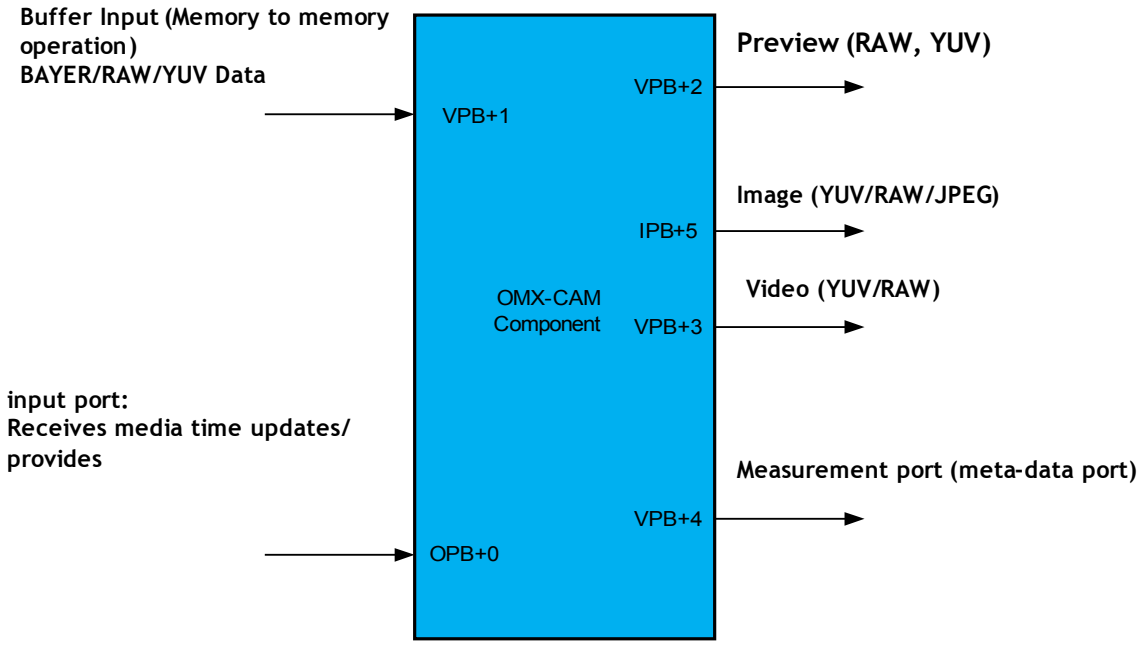


Figure 3 OpenMAX Component structure

Note: Image port and Video port are NOT of similar capability.

Note: All these ports share the same nTimeStamp value. This is used for synchronizing the meta-data received on measurement port with the preview frame.

Note: This single OpenMAX component could run at multiple instances depending on the usecase need.

Port index	Name	Format	Comment
1	Buffer Input	BAYER/YUV	Memory to memory operation is used for Raw data processing,Statistic data collection. it is also used by SIMCOP. we need an input port so that the input buffer can be shared (used for Raw data processing/Statistic data collection)
2	Preview	YUV Data	For Preview/viewfinder frames
5	Image	JPEG/YUV/RAW Data	This single OMX_CAM component shall support interface for Image/Video Encoder. The encoded data can be collected at an output port.
3	Video	YUV/RAW Data	This wont't support JPEG. Using but could be used for other uncompressed image capture also simulating one frame video capture using the flag in SensorMode.
4	Measurement	3A Stats, Histogram, face detect data, Data	Camera subsystem could provide metadata to client which could include 3A statistics, histogram data, face detect data. - in-band meta data – main data payload may be empty. This measurement datas which are less in size are sent along with the preview PORT for legacy reason as well as to reduce the complexity in the external system trying to use the preview frame and corresponding measured data.
0	Time	Gets the time for synchronization.	Accepts media time updates. Provides mechanism for camera component to query for media time. Camera component can detect drift between camera clock and media clock (which may use the audio capturer as a master) and correct timestamps on outgoing frames to compensate. In use case where two cameras are used (e.g. one pointed at user and one pointed away) this provides a consistent media time for timestamps across switches between cameras during capture.

3.3 OpenMAX Camera Component Port Information

Name	**Camera**			
Description	**Emits preview/viewfinder video and captured video/image according to settings.**			
Ports	**Index**	**Domain**	**Direction**	**Description**
	VPB+2	Video	output	Emits preview/viewfinder video and captured video according to settings.
	VPB+3	Video	output	Emits captured video when the camera

				component is capturing where the number of output frames depends on the sensor mode. If the sensor mode is set to one shot (OMX_PARAM_SENSORMODETYPE, bOneShot) then this port only emits a one frame per capture. Output may be interpreted as raw image
	IPB+5	Image	output	Emits captured raw format image when the camera component is capturing. Formats could be YUV/RAW/JPEG etc.
	VPB+4	Video	output	• Emits detected object video when the camera component is capturing. • This port emits in-band metadata on measurements • This port may be used to support fast changing in the frame configuration without the need of re-tunneling components and change settings (Quick parameter changes) directly at each instance. • Meta data are received at the measurement port. • We will get the FD coordinate at this port.
	VPB+1	Video	input	Used for RAW data processing/ Static Data collection
	OPB+0	other/time	input	Accepts media time updates. Provides mechanism for camera component to query for media time. Camera component can detect drift between camera clock and media clock and correct timestamps on outgoing frames to compensate. In use case where two cameras are used (e.g. one pointed at user and one pointed away) this provides a consistent media time for timestamps across switches between cameras during capture.

Note:

*****Measurement PORT is yet another image/video port with in-band metadata support. Utilizing OMX_OTHER_EXTRADATATYPE, which will allow to hold different types of ExtraData.**

***** This extra data could have been clubbed together with the main capture port – BUT, that will increase the data volume for capture port. And since capture port has to run at high priority, than this measurements and the preview port capture port is not burdened with the extra metadata.**

3.3.1 Capture ports and supported contents

Input buffer port, preview port, Video port and measurement ports are all in VIDEO domain. And Image port is in IMAGE domain. Preview port would be used always in video domain way, ie. Multiple frames in a streaming mode. But, other port could be used as image capture port also. Rational for making capture port to be of IMAGE domain is captured in the section 1.1.3.

OMX_PARAM_SENSORMODETYPE is the structure used to differentiate video and image in a port. If bOneShot = TRUE implies it is still image, or else video (streams).

Limitation with VIDEO domain ports is, though it can simulate image capture by using bOneShot flag, format of the video format doesn't have JPEG.

Special Implementation(if needeed): Worst case, if at all JPEG is needed through video domain port, there is MJPEG format with which we could support JPEG in VIDEO domain port.

To make this aspect correct there is an IMAGE domain port. Which has got JPEG format definition. This could emit RAW, YUV and JPEG format. Port definition doesn't have the frame rate notion. For multiple capture case, need to use OMX_CONFIG_EXTCAPTUREMODETYPE.

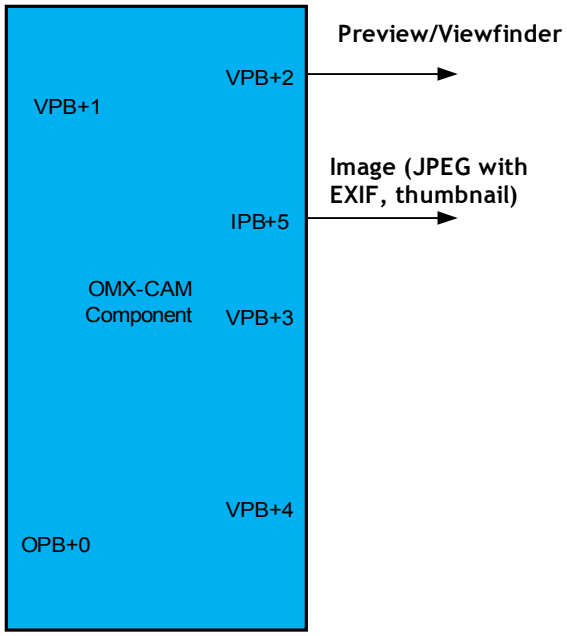
Special Implementation(if needed): And worst case, if at all streaming video is needed through IMAGE domain then need to have OMX_PARAM_SENSORMODETYPE nFrameRate

Figure 4 Extra data along with the main buffer payload

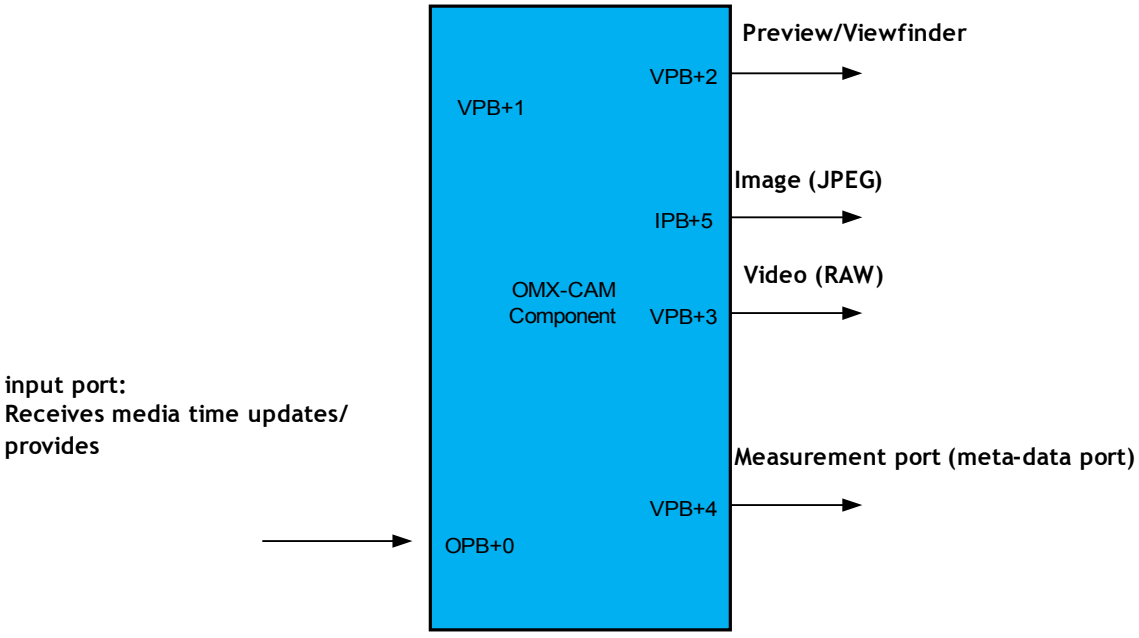
4 Typical camera component usage in Use-cases

4.1 image capture

4.1.1 Single shot JPEG image with embedded EXIF and thumbnail

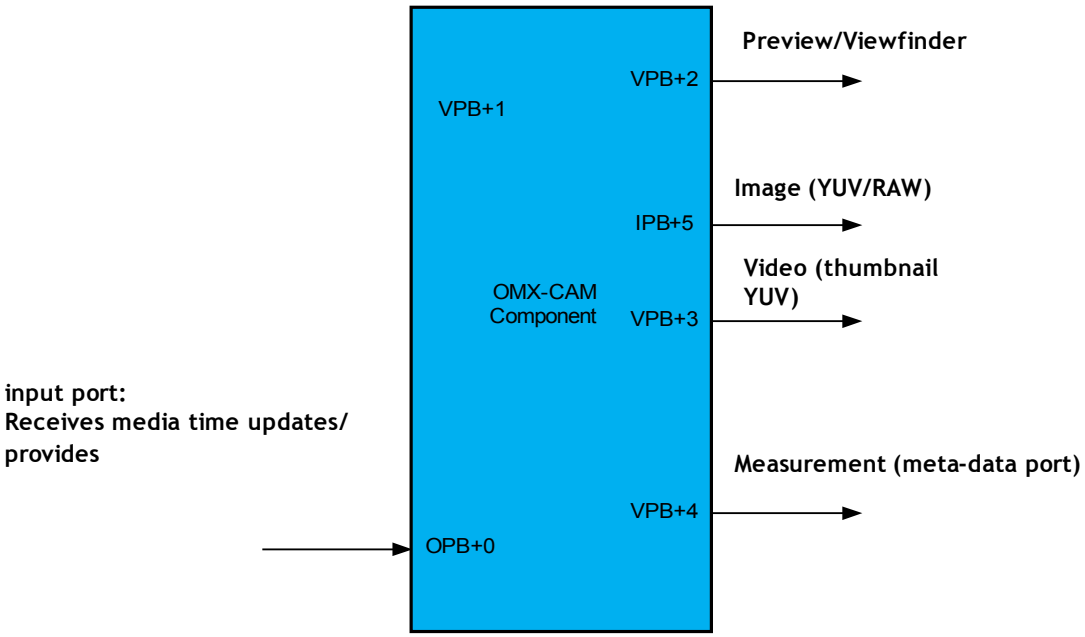


4.1.2 Simultaneous JPEG and RAW capture and store

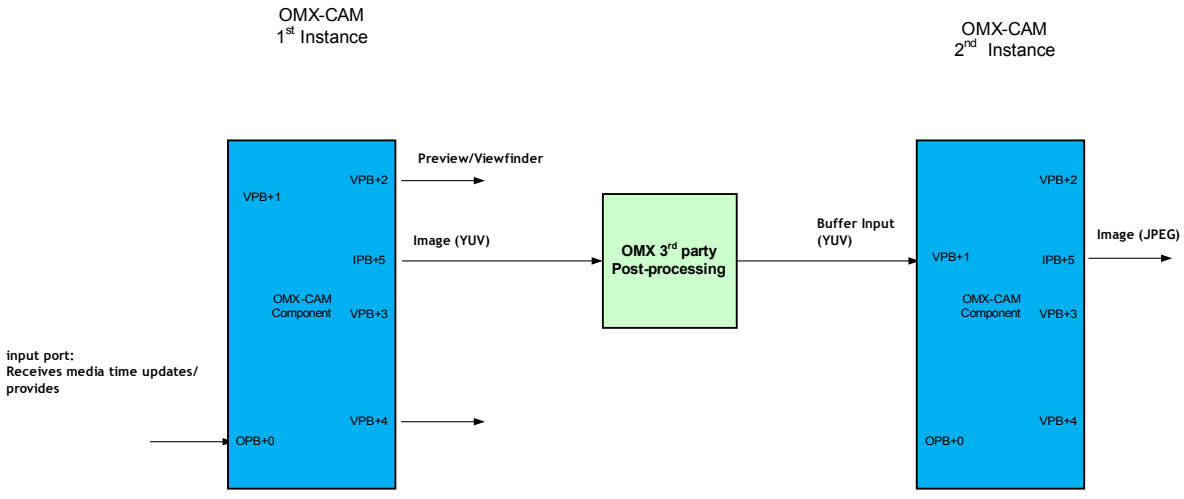


4.1.3 RAW image capture with explicit RAW/YUV thumbnail request

This is used when 3rd party want to do JPEG compression.

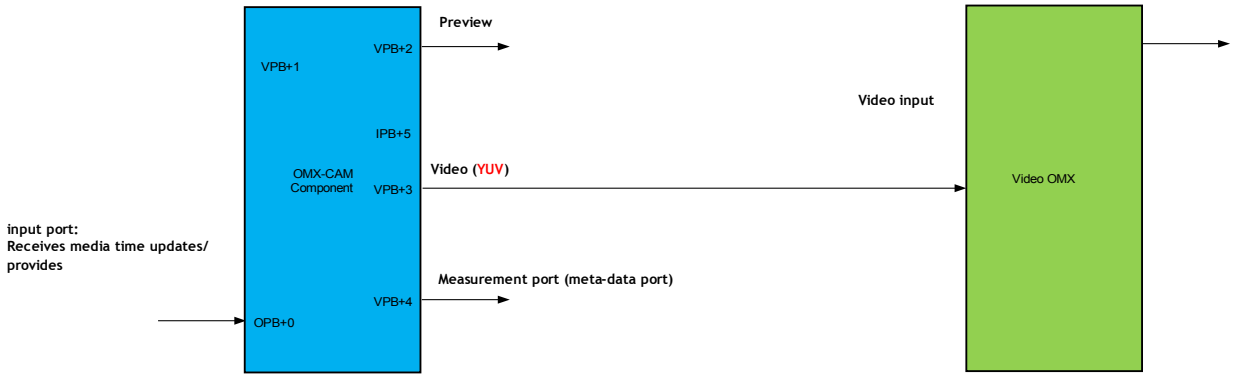


4.1.4 Image capture with 3rd party post-processing and back for JPEG Encoding



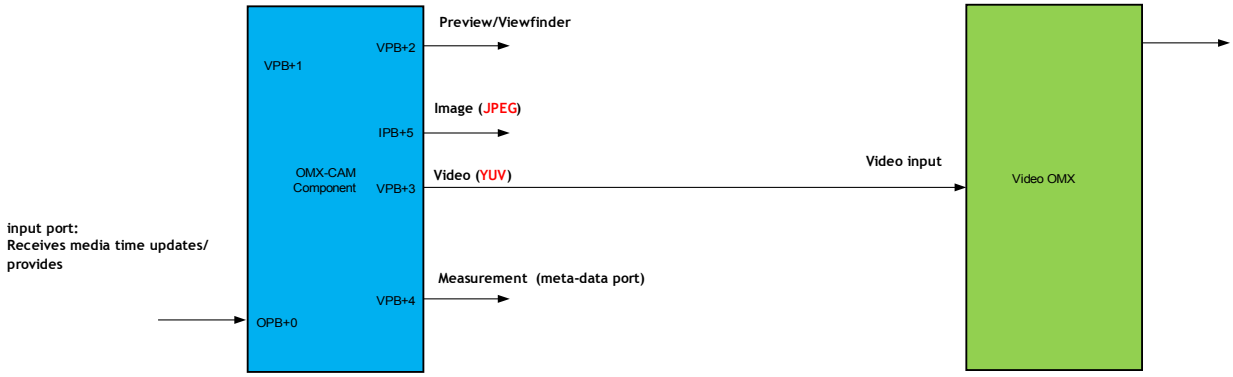
This 3rd party post-processing could be in Ducati or in a different processor (may be Modena in OMAP4).

4.2 Video capture



Note: VSTAB date from the OMX-CAM component could be send through the capture port as meta-data.

4.3 Still Image capture during video recording



5 Host to Ducati aspects

This section give how this OpenMAX camera component would fit in to the whole picture of OMAP4 OR Monica.

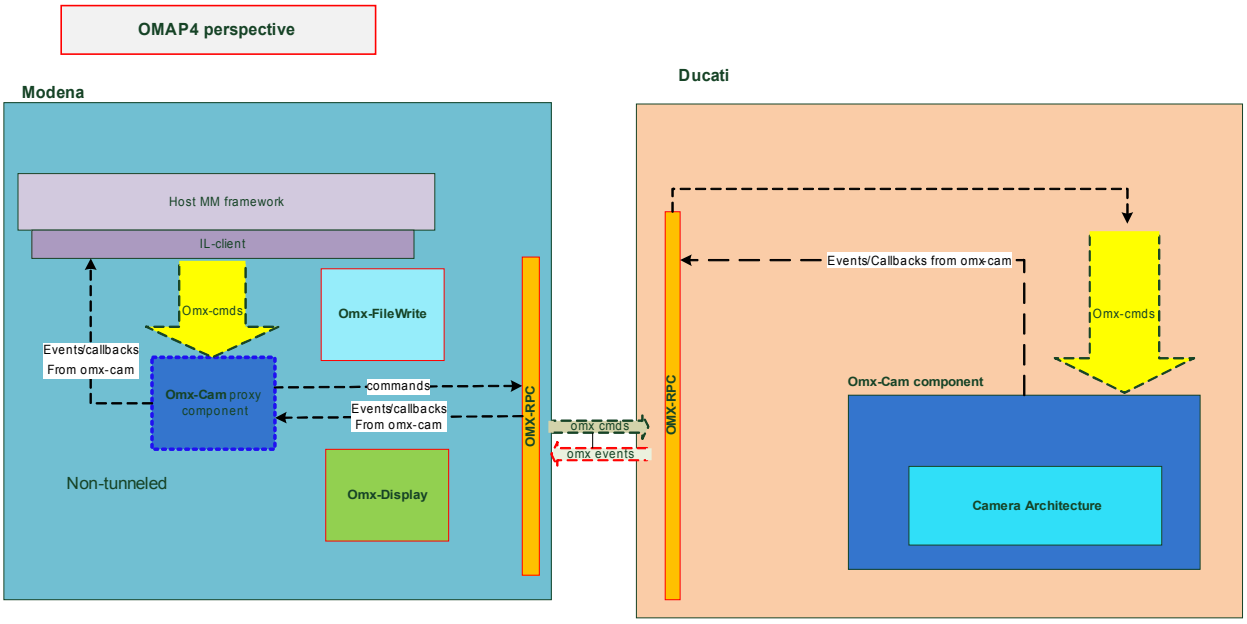


Figure 5 OMAP4 perspective

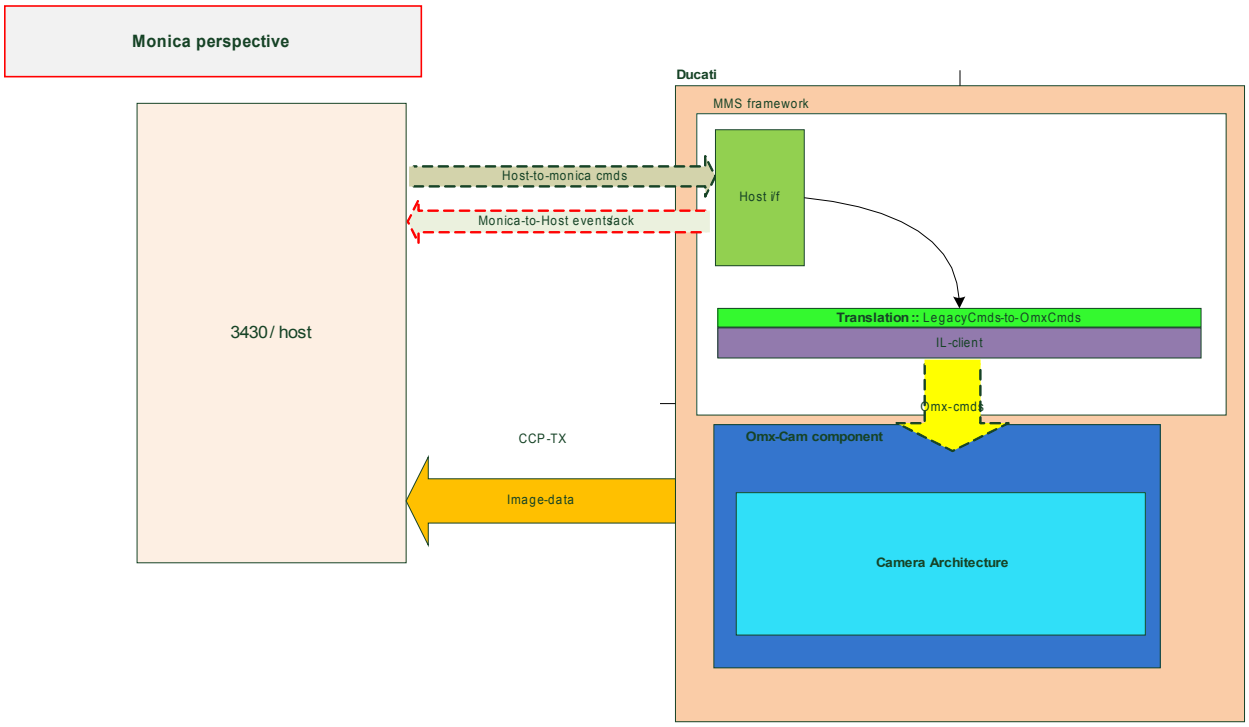


Figure 6 Monica perspective

6 Interface details

This gives the list of index values relevant to OpenMAX Camera and the description. For the respective data structures to be used along with this structure, refer to the next section.

Refer to:

1)\WTSD_DucatiMMSW\omx\omx_il_1_x\omx_core\omx_index.h - For standard indices.

2) \WTSD_DucatiMMSW\omx\omx_il_1_x\omx_core\omx_ti_index.h – For extended indices.

From respective indices there is a reference to the Data Structure(s).

Index	Extended (YES/NO)	Comments

OMX_IndexParamImageInit		
OMX_IndexParamVideoInit		
OMX_IndexParamOtherInit		
OMX_IndexParamPortDefinition		defines the port and domain of the port.
OMX_IndexParamImagePortFormat		Defines the image port formats etc.
OMX_IndexParamVideoPortFormat		For video port or for preview port.
OMX_IndexParamOtherPortFormat		for other port format. Used for time input.
OMX_IndexParamCommonSensorMode		for differentiating single shot or multiple shot in each port. Used to differentiate the ports content to be image or video. By default, all ports are defined to be video domain. bOneShot specifies single shot or multiple.
OMX_IndexConfigExtCapturing	YES	port specific capture command and status check.
OMX_IndexConfigExtPrepareCapturing	YES	Prepare command before actual capture. Shutter-Half press is also the same.
OMX_IndexConfigExtCaptureMode	YES	Specified the capture mode of the camera
OMX_IndexParamSensorSelect	YES	select sensor.
OMX_IndexParamSceneMode	YES	scene mode.
OMX_IndexConfigFlickerCancel	YES	Video cameras may suffer from interference leading to observable flicker, typically with a fixed frequency (50Hz or 60Hz). Algorithms exist to automatically compensate for this
OMX_IndexConfigSensorCal	YES	FULL, QUICK. Permits the host to tell the camera to recalibrate the sensor optics. Receipt of this function tells thecamera component to perform a sensor calibration. The host can use this function to recalibrate the sensor optics if there is any question about the validity of the current sensor calibration data. This configuration setting has no specific associated data values, but data is included for data parsing reasons. Note that sensor calibration may not be possible with some combinations of sensors and optics. This fact will be known at build time. Attempting to calibration such a sensor must result in an error being returned to the host. Sensor calibration is only available in

		application that use the sensor and only for the currently selected sensor.
OMX_IndexConfigCommonDigitalZoom		controls the digital zoom.
OMX_IndexConfigDigitalZoomSpeed	YES	mimics optical zoom.
OMX_IndexConfigDigitalZoomTarget	YES	mimics optical zoom.
OMX_IndexConfigCommonScaleQuality	YES	
OMX_IndexConfigCommonDigitalZoomQuality	YES	
OMX_IndexConfigSmoothZoom		Smooth Zoom control ::Mode, Rate, Quantize, Thresh, Min, Max
OMX_IndexConfigCommonOpticalZoom		controls optical zoom.
OMX_IndexConfigOpticalZoomSpeed		Optical zoom speed level. Special values: • 0 – stop current movement • values from 1 to 254 are mapped proportionally to supported zoom speeds inside optical zoom driver. So 1 is slowest available optical zoom speed and 254 is fastest available optical zoom speed • 255 – default optical zoom speed value
OMX_IndexConfigOpticalZoomTarget		This setting represents optical target zoom level. Levels are mapped to Zoom factors with a LUT inside IVE. LUT can be queried from IVE with CAM_QUERY_CAMERA_FEATURES message. When optical zoom reaches OpticalZoomTarget level, camera will stop driving lens.
OMX_IndexConfigCommonExposureValue		Exposure Value control.
OMX_IndexConfigCommonExposure		Exposure mode control type

OMX_IndexConfigCommonColorFormatConversion		Color conversion is used to specify the coefficients when converting image or video pixel data from YUV to RGB and visa-versa.
OMX_IndexConfigCommonImageFilter		Image filtering is used to apply digital effects to video or image data on the specified port.
OMX_IndexConfigCommonColorEnhancement		Color enhancement is applied to image or video data in YUV formats, where the U and V color components of each pixel are replaced with the specified values. Replacement occurs for each pixel and every frame. This enables a component to add specified color hues to the data. For example, this configuration can be used to convert color image or video data to sepia tone.
OMX_IndexConfigCommonSaturation		set saturation of images, if available.
OMX_IndexConfigBlemish	YES	Enable or disable Blemish correction.
OMX_IndexAutoPauseAfterCapture		Specifies whether the camera will automatically transition to OMX_StatePaused after the Capturing boolean is cleared (e.g. to facilitate a frozen viewfinder).
OMX_IndexConfigCommonContrast		contrast control.
OMX_IndexConfigCommonBrightness		Brightness control.
OMX_IndexConfigCommonFrameStabilisation		for enabling/disabling frame stabilization. For preview or video.
OMX_IndexConfigRedEyeRemoval	YES	Controls Red-Eye removal Algo.
OMX_IndexParamFlashControl		flash control type
OMX_IndexConfigExtFocusRegion	YES	
OMX_IndexConfigCommonFocusRegion		

OMX_IndexConfigCommonFocusStatus		
OMX_IndexConfigDigitalFlash	YES	Specifies whether the LED can be used to assist in autofocus, due to low lighting conditions. 'ENABLE' means use as determined by the auto exposure algorithm.
OMX_IndexConfigPrivacyIndicator	YES	Specifies whether the flash should be used to indicate image or video capture. When flash is not used for exposure, flash will be activated after exposure to indicate image capture. If video light is not used, the flash can be blinking or constant at low intensity to indicate capture but not affect exposure.
OMX_IndexConfigTorchMode	YES	This setting turns on LED at appropriate intensity
OMX_IndexConfigSlowSync	YES	DISABLE – Fire the xenon flash in the usual manner ENABLE – Reduce the light intensity of the main flash (ex 1EV)
OMX_IndexConfigFocusControl		control type and focus value.
OMX_IndexConfigFocusAssist	YES	Specifies whether the LED can be used to assist in autofocus, due to low lighting conditions. 'ENABLE' means use as determined by the auto exposure algorithm.
OMX_IndexConfigImageFocusLock	YES	Controls the focus by allowing to lock the current setting
OMX_IndexConfigImageWhiteBalanceLock	YES	Controls the white balance by allowing to lock the current setting including any AWB gain offset, color conversion matrix (both RGB2RGB and RGB2YUV), gamma curve, and additional contrast adjustment.
OMX_IndexConfigImageExposureLock	YES	Controls the exposure by allowing to lock the current setting.
OMX_IndexConfigImageAllLock	YES	Simultaneously lock focus, white balance and exposure (and relevant other settings).
OMX_IndexConfigImageDeNoiseLevel	YES	Controls the Noise reduction level. Automatic settings consider light conditions.
OMX_IndexConfigSharpeningLevel	YES	Controls the sharpening level. Automatic setting considers noisiness of the image.

OMX_IndexConfigDeBlurringLevel	YES	Controls the de-blurring level. Automatic setting considers noisiness of the image.
OMX_IndexConfigChromaCorrection	YES	Controls the chroma correction level. The automatic setting considers the chromatic errors.
OMX_IndexConfigDeMosaicingLevel	YES	Controls the de-mosaicing. Automatic setting considers chromatic errors.
OMX_IndexConfigCommonWhiteBalance		It performs the the color correction so that a white object appears truly white and not a tint of the light source.
OMX_IndexConfigImageWhiteBalanceLock	YES	Controls the white balance by allowing to lock the current setting including any AWB gain offset, color conversion matrix (both RGB2RGB and RGB2YUV), gamma curve, and additional contrast adjustment.
OMX_IndexConfigWhiteBalanceGain	YES	Controls the gain for white balance adjustment.
OMX_IndexConfigCommonRGB2RGB	YES	Controls conversion from sensor RGB to standard RGB.
OMX_IndexConfigCommonRGB2YUV	YES	Controls conversion from RGB to YUV.
OMX_IndexConfigCommonYUV2RGB	YES	controls conversion from YUV to RGB.
OMX_IndexConfigCommonGammaTable	YES	Controls the gamma value using loop up table.
OMX_IndexConfigImageFaceDetection	YES	
OMX_IndexConfigImageBarcodeDetection	YES	
OMX_IndexConfigImageSmileDetection	YES	
OMX_IndexConfigImageBlinkDetection	YES	
OMX_IndexConfigImageFrontObjectDetection	YES	
OMX_IndexConfigImageMotionEstimation	YES	this is sent as extra data to the specified port. When video stabilization is on this is helpful.
OMX_IndexConfigHistogramMeasurement	YES	The image Histogram is measured on the data image input and gives the number of pixels for each tonal value. The result is delivered as extra data
OMX_IndexConfigDistanceMeasurement	YES	
OMX_IndexConfigCommonRotate		Applied on the width and height of the image. 0/180 for the width and 90/270 for the height.

OMX_IndexConfigCommonMirror		Mirror effects image frames along the horizontal and the vertical axes.
OMX_IndexConfigCommonOutputCrop		Crop the image stream to the specified rectangular.
OMX_IndexConfigSnapshotToPreview	YES	Enables snapshot which is the captured image transformed to the same format and resolution as the preview and sent before captured image is processed further.
OMX_IndexConfigJpegHeaderType	YES	Specifies EXIF, JFIF or No header.
OMX_IndexParamQFactor		QFactor for JPEG/ and compressed output. 1- 100
OMX_IndexParamQuantizationTable,		Quantization table for compressed image output.
OMX_IndexParamHuffmanTable,		Huffman table for compressed image output.
OMX_IndexParamJpegMaxSize	YES	Sets the maximum size of the output image (only valid when capturing in JPEG format). This parameter does not affect the thumbnail or the ancillary data. This setting is automatically disabled during multi-shot image capture.
OMX_IndexConfigRestartMarker	YES	Restart Market insertion parameters for JPEG Encoding.
OMX_IndexParamImageStampOverlay	YES	For date time stamp overlay.
OMX_IndexParamThumbnail	YES	Configures the aspects of thumbnail needed for JPEG or as a separate output. And format of the thumbnail. Height, Width, port, format, wuality, max size.
OMX_IndexConfigImageStabilisation	YES	For image stabilization.
OMX_IndexConfigMotionTriggeredImageStabilisation	YES	Enable/Disable Motion triggered image stabilization.
OMX_IndexParamHighISONoiseFilter	YES	Making use of NSF2 module specifically for high ISO noise. For low light conditions. Enable/Disable NSF2-Algo. Applicable when low-light image noise reduction is requested, it shall support Auto mode also in which the use-case pipeline with in the camera can select depending on the light

		conditions.
OMX_IndexParamLensDistortionCorrection	YES	Making use of LDC module within SIMCOP. Functionality achieving is LenseDistortion correction.Enable/Disable LDC Algo.
OMX_IndexParamLocalBrightnessAndContrast	YES	Enable/Disable LBCE
OMX_IndexParamVideoCaptureYUVRange	YES	Indicates the range of pixel values. The application should ensure that the encoder input is constrained as per pixel range and codec should ensure that the pixel range is signaled in the bitstream.
OMX_IndexConfigChromaticAberrationCorrection	YES	Enable Disable Chromatic aberration correction.
OMX_IndexParamProcessingOrder	YES	TODO
OMX_IndexConfigOtherExtraDataControl	YES	Control extra data attachment to the port buffer.
OMX_IndexConfigVideoNoiseFilter	YES	Give ON/OFF/AUTO mode control for VNF.
OMX_IndexConfigISOSetting	YES	Camera shall support auto and manual control of ISO settings for the sensor.Values configurable by the host include AUTO and manual settings ranging from 1 to 3200
OMX_IndexConfigExtFocusRegion	YES	Defines the focus region as a rectangular region in the frame.
OMX_IndexParamStandardComponentRole		For set/get the standard roles of the component. (capture, iv_processor, jpeg_encode)
OMX_IndexCameraOperatingMode	YES	Specifies the capture use case .

6.1 Parameters/Configuration (including extended) applied to Port VPB+1

Port Index	VPB+1
Description	Emits preview/viewfinder video when the camera component is executing.

Required Parameters /Configs	Index	Access	Description
	OMX_IndexParamPortDefinition	r/w	Specifies preview's resolution and framerate. nFrameWidth = *320* nFrameHeight = *240* nStride = 320 nSliceHeight = *16* eCompressionFormat = OMX_VIDEO_CodingUnused eColorFormat = *OMX_COLOR_FormatYUV420Planar*
	OMX_IndexParamVideoPortFormat	r/w	eCompressionFormat = OMX_VIDEO_CodingUnused eColorFormat = *OMX_COLOR_FormatYUV420Planar*
	OMX_IndexConfigImageFaceDetection	r/w	
	OMX_IndexConfigureHistogramEstiamtion	r/w	nBins=64

6.2 Parameters/Configuration (including extended) applied to Port VPB+3

Port Index	VPB+3		
Description	Emits captured video when the camera component is capturing where the number of output frames depends on the sensor mode. If the sensor mode is set to one shot then this port only emits a one frame per capture.		
Formats	OMX_VIDEO_CodingUnused		
Required Parameters /Configs	**Index**	**Acces s**	**Description**
	OMX_IndexParamPortDefinition	r/w	Specifies emitted video's resolution and framerate. Eg: nFrameWidth = *640* nFrameHeight = *480* nStride = 640 nSliceHeight = *16* eCompressionFormat = OMX_VIDEO_CodingUnused eColorFormat = *OMX_COLOR_ FormatYUV420Planar*

6.3 Parameters/Configuration (including extended) applied to Port IPB+5

Port Index	IPB+5		
Description	Emits captured image when the camera component is capturing where the number of output frames depends on the OMX_CONFIG_EXTCAPTUREMODETYPE.		
Formats			
Required Parameters /Configs	**Index**	**Acces s**	**Description**
	OMX_IndexParamPortDefinition	r/w	Specifies emitted image resolution and compression and color format. Eg: nFrameWidth = *640*

			nFrameHeight = *480* nStride = 640 nSliceHeight = *16* eCompressionFormat = OMX_IMAGE_CodingJPEG eColorFormat = *OMX_COLOR_FormatYUV420Planar*

6.4 Parameters/Configuration (including extended) applicable to Port VPB+4

Port Index	VPB+4		
Description	This port emits in-band metadata on measurements.		
Required Parameters /Configs	**Index**	**Access**	**Description**
	OMX_IndexParamPortDefinition	r/w	nBufferSize >= largest metadata structure expected to be able to handle all the extra data that is passed in the buffers, needs to further be increased with the image formats size requirements. Depends on use case. When used with face detection and bSnapshot is true, the following apply: nFrameWidth = *64* nFrameHeight = *64* eCompressionFormat = OMX_VIDEO_CodingUnused

			eColorFormat = *OMX_COLOR_FormatYUV420Planar*
	OMX_IndexConfigImageBarcodeDetection	r/w	
	OMX_IndexConfigImageSmileDetection	r/w	
	OMX_IndexConfigImageBlinkDetection	r/w	
	OMX_IndexConfigImageFrontObjectDetection	r/w	
	OMX_IndexConfigImageMotionEstimation	r/w	

7 Data Structure used

7.1 OpenMAX Component Data structure

OMX_ParamSensorMode

Sensor mode is used to specify the frame rate and the resolution that an image sensor or the camera uses to capture image or video. The sensor mode is distantly separated from the port of the component, which may modify the data produced by the image sensor.

OMX_ParamSensorMode is defined as

typedef struct OMX_ParamSensorMode{

OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex ;

OMX_U32 nFrameRate ;

OMX_BOOL bOneShot ;

OMX_FRAMESIZETYPE sFrameSize ;

}OMX_ParamSensorMode ;

The parameters for **OMX_ParamSensorMode** are defined as follows.

• nPortIndex is the read-only value containing the index of the port.
• nFrameRate is the frame rate is in frames per second. This value is represented in Q16 format. The value 0x0 is used to indicate the frame rate is unknown, variable, or is not needed.

• bOneShot is a Boolean value that enables or disables one shot mode.
• sFrameSize is the resolution of the image sensor mode

OMX_IndexParamImagePortFormat

Port format parameter. This structure is used to enumerate the various data input/output format supported by the port.

typedef struct OMX_IMAGE_PARAM_PORTFORMATTYPE {

OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_U32 nIndex;

OMX_IMAGE_CODINGTYPE eCompressionFormat;

OMX_COLOR_FORMATTYPE eColorFormat;

} OMX_IMAGE_PARAM_PORTFORMATTYPE;

* STRUCT MEMBERS:

nSize : Size of the structure in bytes

nVersion : OMX specification version information

nPortIndex : Indicates which port to set

nIndex : Indicates the enumeration index for the format from 0x0 to N-1

eCompressionFormat : Compression format used in this instance of the component. When OMX_IMAGE_CodingUnused is specified, eColorFormat is valid

eColorFormat : Decompressed format used by this component

OMX_IndexConfigCommonWhiteBalance

Use OMX_GetConfig and OMX_SetConfig to access to access **OMX_CONFIG_WHITEBALANCECONTROLTYPE**. White balance performs the color correction so that a white object appears truly white and not a tint of color of the light source. The adjustment can be controlled automatically or manually.

OMX_CONFIG_ WHITEBALANCECONTROLTYPE is defined as

typedef struct OMX_ CONFIG _ WHITEBALANCECONTROLTYPE {

OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex ;

OMX_WHITEBALANCECONTROLTYPE eWhiteBalControl ;

}OMX_ CONFIG _WHITEBALANCECONTROLTYPE;

The parameters for **OMX_ CONFIG _WHITEBALANCECONTROLTYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- eWhiteBalControl is the enumerated valued indicating the type of white balance control used. The table given below gives the details of the values that can be selected for white balance control.

Table 1 Values that can be selected for white balance control

OMX_WHITEBALCONTROLTYPE Enumerated Value	**Description**
OMX_WhiteBalControlOff	Disables exposure control.
OMX_WhiteBalControlAuto	Automatic white balance control. The color temperature of the captured image or video stream is adjusted per frame using a white reference from within each frame.
OMX_WhiteBalControlSunLight	Manual white balance control when the sun provides the light source.
OMX_WhiteBalControlCloudy	Manual white balance control when the sun provides the light source through clouds.
OMX_WhiteBalControlShade	Manual white balance control when the light source is the sun and the scene is in the shade.
OMX_WhiteBalControlTungsten	Manual white balance control when the light source is tungsten.
OMX_WhiteBalControlFluorescent	Manual white balance control when the light source is fluorescent.
OMX_WhiteBalControlIncandescent	Manual white balance control when the light source is incandescent.
OMX_WhiteBalControlFlash	Manual white balance control when the light source is a flash.
OMX_WhiteBalControlHorizon	Manual white balance control when the light source is the sun on the horizon. 0
OMX_WhiteBalControlFacePriorityMode (Extended Enumerated value)	White balance control priority on face area. The color temperature of the captured image or video stream is adjusted per frame using a face reference from within each frame

OMX_IndexConfigExtCapturing

```
/**
 * Port specific capture trigger
 * useful for the usecases with multiple capture ports.
 *
 * STRUCT MEMBERS:
 *  nSize      : Size of the structure in bytes
 *  nVersion   : OMX specification version information
 *  nPortIndex : Port that this structure applies to
 *  bExtCapturing : Start Captre at the specified port. Can be queried to know the status of a specific port.
 */
```

```
typedef struct OMX_CONFIG_EXTCAPTURING {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bExtCapturing;
} OMX_CONFIG_EXTCAPTURING;
```

Even for preview port this could be applied.

OMX_IndexConfigExtPrepareCapturing

This is manly for shutter-half press sort of event from upper layer. This is not the capture command, but a command to let the camera component to know user is about to take picture. And frameworjk inside the camera could take care of the functionalities to be active from that moment.

OMX_IndexConfigCommonDigitalZoom

OMX_IndexConfigCommonOpticalBalance

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_SCALEFACTORTYPE.** (Scaling is used to stretch or shrink video or image data on the specified input or output port).

- Digital zoom implements a camera zoom feature digitally.
- Optical zoom “zooms” an image in or out using a lens on a camera.

OMX_Config_SCALEFACTORTYPE is defined as follows.

```
typedef  struct OMX_CONFIG_SCALEFACTORTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex ;
OMX_S32 xWidth ;
OMX_S32 xHeight ;
} OMX_CONFIG_SCALEFACTORTYPE;
```

The parameters for **OMX_CONFIG_SCALEFACTORTYPE** is defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- xWidth is the scaling in the horizontal direction in Q16 format (i.e., signed 15.16 fixed pointed format). For example, a scaling factor of 0x10000 would not change the width, but a scaling factor of 0x8000 would scale the width by 50%.

xHeight is the scaling in the vertical direction in Q16 format (i.e., signed 15.16 fixed pointed format). For example, a scaling factor of 0x10000 would not change the height, but a scaling factor of 0x20000 would scale the height by 200%.

OMX_CONFIG_EXPOSUREVALUETYPE

Exposure is the amount of light which falls upon the sensor of the digital camera. Sutter speed, sensitivity, and aperture are adjusted to achieve optimal exposure of scene. Most digital camera offers a verity of exposure modes, from fully-automatic to semi-automatic to full manual mode.

OMX_CONFIG_EXPOSUREVALUETYPE is defined as follows.

typedef struct OMX_CONFIG_EXPOSUREVALUETYPE {

OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex ;

OMX_METERINGTYPE emetering;

OMX_S32 xEVCompensation;

OMX_U32 nApertureFNumber;

OMX_BOOL bAutoAperture;

OMX_U32 nShutterSpeedMsec;

OMX_BOOL bAutoShutterSpeed;

OMX_U32 nSensitivity;

OMX_BOOL bAutoSensitivity;

} OMX_CONFIG_EXPOSUREVALUETYPE;

The parameters for **OMX_CONFIG_EXPOSUREVALUETYPE** is defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- eMetering is the metering type to be used. The table given below lists the valid metering modes that can be used.

Table 2 List of valid metering modes

OMX_METERINGTYPE Enumerated Value	Description
OMX_MeteringModeAverage	Center weight average metering
OMX_MeteringModeSpot	Spot (partial) metering
OMX_MeteringModeMatrix	Matrix or evaluative metering

- xEVCompensation is the Exposure Value compensation defined in Q16 format.
- nApertureFNumber is the aperture f-stop setting defined in Q16 format. A value of 2 implies a “f/2” setting. This setting is only valid for SetConfig if auto aperature mode is not set.
- bAutoAperture is a Boolean value indicating if auto-aperture is to be enabled and applied.
- nShutterSpeedMsec is the shutter speed specified in units of milliseconds. This setting is only valid for SetConfig if auto shutter speed is not set.
- bAutoShutterSpeed is a Boolean value indicating if auto shutter speed is to be enabled and applied.
- nSensitivity is the ISO sensitivity setting. A value of 100 implies a “ISO 100” setting. This setting is only valid for SetConfig if auto sensitivity is not set.

bAutoSensitivity is a Boolean value is indicating if auto sensitivity is to be enabled and applied.

OMX_IndexConfigCommonImageFilter

Used with OMX_GetConfig and OMX_SetConfig to assess **OMX_CONFIG_IMAGEFILTERTYPE**. Image filtering applies digital effects to a video or image stream.

OMX_CONFIG_IMAGEFILTERTYPE is defined as follows.

```
typedef  struct OMX_CONFIG_IMAGEFILTERTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex ;
OMX_IMAGEFILTERTYPE eImageFilter;
} OMX_CONFIG_IMAGEFILTERTYPE;
```

The parameters for **OMX_CONFIG_IMAGEFILTERTYPE** is defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- eImageFilter is the enumerated valued indicating the image filter used. The Table given below gives the details of the values that can be selected for the image filter.

Table 3 Values that can be selected for the image filter

OMX_IMAGEFILTERTYPE Enumerated Value	**Description**
OMX_ImageFilterNone	Used to disable image filtering.
OMX_ImageFilterNoise	Filters data to remove noise from the image.
OMX_ImageFilterEmboss	Filters data to give an embossed appearance (stamped from the rear for a raised effect along edges).
OMX_ImageFilterNegative	Filters data to negate colors.
OMX_ImageFilterSketch	Filters data to give the appearance of having been sketched by an artist.
OMX_ImageFilterOilPaint	Filters data to appear as if it were hand painted using a brush with oil paints.
OMX_ImageFilterHatch	Filters data to appear as if it were printed on a material with a grain.
OMX_ImageFilterGpen	Filters data to appear as if it were drawn with a pen.
OMX_ImageFilterAntialias	Filters data to anti-alias pixels so as to sharpen edges in the image or video stream.
OMX_ImageFilterDeRing	Filters data to remove erroneous artifacts introduced by inherent limitations of the numerical processing of digital image data.
OMX_ImageFilterSolarize	Filters data to create a solarization effect.

OMX_IndexConfigCommonRGB2RGB

OMX_IndexConfigCommonRGB2YUV

Used with OMX_GetConfig and OX_SetConfig to access **OMX_CONFIG_COLORCONVERSIONTYPE.**

Color conversion programs the coefficient used when coverting the pixel data from YUV to RGB and visa-versa.

OMX_CONFIG_COLORCONVERSIONTYPE is defined as follows.

typedef struct OMX_CONFIG_COLORCONVERSIONTYPE {
OMX_U32 nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex ;
OMX_S32 xColorMatrix[3][3] ;
OMX_S32 xColorOffSet[3];
OMX_BOOL bFullColorRange;
} OMX_CONFIG_COLORCONVERSIONTYPE;

The paramater for **OMX_CONFIG_COLORCONVERSIONTYPE** is defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- This structure represents linear color conversion from one space to another. For example, to conversion from one RGB color into another RGB color space can be represented as

  R' = xColorMatrix[1][1]*R + xColorMatrix[1][2]*G + xColorMatrix[1][3]*B + xColorOffset[1]

  G' = xColorMatrix[2][1]*R + xColorMatrix[2][2]*G + xColorMatrix[2][3]*B + xColorOffset[2]

  B' = xColorMatrix[3][1]*R + xColorMatrix[3][2]*G + xColorMatrix[3][3]*B + xColorOffset[3]

  Both xColorMatrix and xColorOffset are represented as Q16 value.

  bFullColorRange represents represents whether valid range of color is 0 to 255 (when set to TRUE) or 16 to 235 (for FALSE).

  Again all values assume that maximum value is 255. If internal implementation uses higher dynamic range, this value should be adjusted internally

OMX_IndexConfigCommonColorEnhancement

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_COLORENHANCEMENTTYPE.** Color enhancement replaces the U and V values of a YUV image with specified constant values to apply a color effect to an image or video stream.

OMX_CONFIG_COLORENHANCEMENTTYPE is defined as follows.

```
typedef  struct OMX_CONFIG_COLORENHANCEMENTTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bColorEnhancement;
OMX_U8 nCustomizedU;
OMX_U8 nCustomizedV;
}OMX_CONFIG_COLORENHANCEMENTTYPE;
```

The parameters for **OMX_CONFIG_COLORENHANCEMENTTYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- bColorEnhancement is the Boolean value that enables or disables color enhancement.
- nCustomizedU is a value for replacing the U color component of each pixel. The range of values is 0-255. Practical values are in the range of 16-240.

nCustomizedV is the value for replacing the V color component of each pixel. The range of values is 0-255. Practical values are in the range of 16-240.

OMX_IndexConfigCaptureMode

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_CAPTUREMODETYPE.**

Capture mode configuration is used to instruct the camera component how it shall behave during the course of capturing: continuous versus frame count limited capturing operatios.

OMX_Config_CAPTUREMODETYPE is defined as follows.

```
typedef  struct OMX_CONFIG_CAPTUREMODETYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bContinuous;
OMX_BOOL bFrameLimited;
OMX_U32 nFrameLimited;
} OMX_CONFIG_CAPTUREMODETYPE;
```

The parameters for **OMX_CONFIG_CAPTUREMODETYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.

- bContinuous is a Boolean used to indicate the frame rate emission. If true then ignore the port frame rate setting and emit captured frame data as quickly as possible otherwise obey the port's frame rate setting.
- bFrameLimited is a Boolean used to indicate if capturing shall be terminated after the specified number of frames if true frame limited capture is enabled; otherwise the port does not terminate capturing until instructed to do so by the client.

nFrameLimit is the limit on number of frames emitted during capturing, this parameter is only valid if bFrameLimited is enabled.

In the case of **multi shot**, for best picture selection, allow the camera to capture and emit a number of frames taken before and after the capture bit was set. This is possible by extending the capture mode type. This is defined as **OMX_CONFIG_EXTCAPTUREMODETYPE**.

```
typedef struct OMX_CONFIG_EXTCAPTUREMODETYPE {
   OMX_U32 nSize;
   OMX_VERSIONTYPE nVersion;
   OMX_U32 nPortIndex;
   OMX_U32 nFrameRate;
   OMX_U32 nFrameBefore;
   OMX_BOOL bPrepareCapture;
   OMX_BOOL bEnableBracketing;
   OMX_CONFIG_BRACKETINGTYPE tBracketConfigType;
} OMX_CONFIG_EXTCAPTUREMODETYPE;
```

The parameters for **OMX_CONFIG_EXTCAPTUREMODETYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- nFrameRate : when bContinuous is FALSE, need to define the frame rate of the muti-shot scenario. Since this would be applicable to IMAGE domain port, there is no port specific frame rate.

-
- nFrameBefore is specifying how many frames before the capture trigger shall be used. It is implementation dependent how many is supported. This shall only be supported for images and not for video frames.
- bPrepareCapture should be set to true when nFrameBefore is greater than zero and before capturing of before-frames should start. The component is not allowed to deliver buffers until capturing starts. This shall only be supported for images and not for video frames.
- bEnableBracketing should be enabled when bracketing is used. In bracketing mode, one parameter can be changed per each capture.
- tBracketTypeConfig specifies bracket mode configuration to use. Valid only when bEnableBracketing is set.

Table 4 Bracket Mode Type

OMX_ BRACKETTYPE Enumerated Value	Description
OMX_BracketExposureRelativeInEV	Exposure value is changed relative to the value set by automatic exposure. nBracketStartValue and nBracketStep are in Q16. Increment is additive.
OMX_BracketExposureAbsoluteMs	Exposure value is changed in absolute value in ms. nBracketStartValue and nBracketStep are in Q16. Increment is multiplicative.
OMX_BracketFocusRelative	Focus is adjusted relative to the focus set by auto focus. The value is S32 integer, and is the same as adjusting nFocusSteps of OMX_IMAGE_CONFIG_FOCUSCONTROLTYPE relatively. Increment is additive.
OMX_BracketFocusAbsolute	Focus position is adjusted absolutely. It is the same as setting nFocusSteps of OMX_IMAGE_CONFIG_FOCUSCONTROLTYPE relatively for each captures. The value should be interpreted as U32 value. Increment is additive.
OMX_BracketFlashPower	Power of flash is adjusted relative to the automatic level. Increment is multiplicative.
OMX_BracketAperture	Aperture number relative to the automatic setting. Data in Q16 format. Increment is multiplicative.

- nBracketStartValue represents the initial starting value for the bracketing. Its interpretation depends on the bracket mode chosen. In case of relative bracket modes, this value can be either additive or multiplicative to the default automatic value depending on the mode.
- nBracketStep represents incremental step between each captures. For each capture, the bracket value is added or multiplied by this value.

OMX_IndexAutoPauseAfterCapture

OMX_IndexConfigCapturing

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_Config_BOOLEANTYPE** to query the component if it is capturing the data. The **OMX_Config_BOOLEANTYPE** structure contains generic Boolean configuration information that may be used to set component level configuration settings rather than port level configuration settings.

OMX_CONFIG_BOOLEANTYPE is defined as follows.

```
typedef  struct OMX_CONFIG_BOOLEANTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_BOOL bEnabled;
} OMX_CONFIG_BOOLEANTYPE;
```

The parameters for **OMX_CONFIG_BOOLEANTYPE** are defined as follows.

- bEnabled is a Boolean used to indicate if a configuration is to be enabled. The configuration setting to be enabled is typically inherent in the name of the configuration or parameter indice used with this structure.

For example, the OMX_IndexAutoPauseAfterCapture index will use the OMX_CONFIG_BOOLEANTYPE structure to enable or disable the auto pause mechanism after a capture request is completed.

OMX_IndexConfigCommonGammaTable …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_GAMMATABLETYPE.**

OMX_CONFIG_GAMMATABLETYPE is defined as follows.

```
typedef  struct OMX_CONFIG_GAMMATABLETYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
```

OMX_U32 nPortIndex;

OMX_U32 xGamma[3][256];

} OMX_CONFIG_GAMMATABLETYPE;

The parameters for **OMX_CONFIG_GAMMATABLETYPE** are defined as follows.

- xGamma represent loop up tale for gamma correction in Q 16 format.
- All values assumed that the maximum value is 255. if internal uses higher dynamic range, this value should be adjusted internally.

OMX_IndexParamQFactor

```
/**
 * Q Factor for JPEG compression, which controls the tradeoff between image
 * quality and size.  Q Factor provides a more simple means of controlling
 * JPEG compression quality, without directly programming Quantization
 * tables for chroma and luma
 *
 * STRUCT MEMBERS:
 *  nSize      : Size of the structure in bytes
 *  nVersion   : OMX specification version information
 *  nPortIndex : Port that this structure applies to
 *  nQFactor   : JPEG Q factor value in the range of 1-100. A factor of 1
 *               produces the smallest, worst quality images, and a factor
 *               of 100 produces the largest, best quality images.  A
 *               typical default is 75 for small good quality images
 */

typedef struct OMX_IMAGE_PARAM_QFACTORTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nQFactor;
} OMX_IMAGE_PARAM_QFACTORTYPE;
```

OMX_IndexParamQuantizationTable

```
/**
 * JPEG quantization tables are used to determine DCT compression for
 * YUV data, as an alternative to specifying Q factor, providing exact
 * control of compression
 *
 * STRUCT MEMBERS:
```

```
 *  nSize               : Size of the structure in bytes
 *  nVersion            : OMX specification version information
 *  nPortIndex          : Port that this structure applies to
 *  eQuantizationTable  : Quantization table type
 *  nQuantizationMatrix[64] : JPEG quantization table of coefficients stored
 *                      in increasing columns then by rows of data (i.e.
 *                      row 1, ... row 8). Quantization values are in
 *                      the range 0-255 and stored in linear order
 *                      (i.e. the component will zig-zag the
 *                      quantization table data if required internally)
 */
typedef struct OMX_IMAGE_PARAM_QUANTIZATIONTABLETYPE {
   OMX_U32 nSize;
   OMX_VERSIONTYPE nVersion;
   OMX_U32 nPortIndex;
   OMX_IMAGE_QUANTIZATIONTABLETYPE eQuantizationTable;
   OMX_U8 nQuantizationMatrix[64];
} OMX_IMAGE_PARAM_QUANTIZATIONTABLETYPE;
```

OMX_IndexParamHuffmanTable

```
/**
 * JPEG Huffman table
 *
 * STRUCT MEMBERS:
 *  nSize                     : Size of the structure in bytes
 *  nVersion                  : OMX specification version information
 *  nPortIndex                : Port that this structure applies to
 *  eHuffmanTable             : Huffman table type
 *  nNumberOfHuffmanCodeOfLength[16] : 0-16, number of Huffman codes of each
 *                          possible length
 *  nHuffmanTable[256]          : 0-255, the size used for AC and DC
 *                          HuffmanTable are 16 and 162
 */
typedef struct OMX_IMAGE_PARAM_HUFFMANTTABLETYPE {
   OMX_U32 nSize;
   OMX_VERSIONTYPE nVersion;
   OMX_U32 nPortIndex;
   OMX_IMAGE_HUFFMANTABLETYPE eHuffmanTable;
   OMX_U8 nNumberOfHuffmanCodeOfLength[16];
   OMX_U8 nHuffmanTable[256];
}OMX_IMAGE_PARAM_HUFFMANTTABLETYPE;
```

```
typedef enum OMX_IMAGE_HUFFMANTABLETYPE {
    OMX_IMAGE_HuffmanTableAC = 0,
    OMX_IMAGE_HuffmanTableDC,
    OMX_IMAGE_HuffmanTableACLuma,
    OMX_IMAGE_HuffmanTableACChroma,
    OMX_IMAGE_HuffmanTableDCLuma,
    OMX_IMAGE_HuffmanTableDCChroma,
    OMX_IMAGE_HuffmanTableMax = 0x7FFFFFFF
} OMX_IMAGE_HUFFMANTABLETYPE;
```

OMX_IndexConfigProcessingOrder …….. Extended

Used with OMX_GetConfig to access **OMX_CONFIGPROCESSINGORDERTYPE**. It is used to query the order of the camera processing features.

```
typedef  struct OMX_CONFIGPROCESSINGORDERTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_U32 nIndex;
OMX_PROCESSINGTYPE eProc;
} OMX_CONFIGPROCESSINGORDERTYPE;
```

The parameters for **OMX_CONFIGPROCESSINGORDERTYPE** are defined as follows.

- nIndex is the ordered index number of the particular processing. A query with zero will return the number of processing items, I.e. the maximum index number. All other index values will return the processing enum value.
- eProc is an enumeration specifying the particular processing type, as shown in table.

Table 5 Processing Type Values

OMX_PROCESSINGTYPE Enumerated Value	Description
OMX_BLOOMING REDUCTION	
OMX_DENOISE	

OMX_SHARPENING	
OMX_DEBLURRING	
OMX_DEMOSAICING	
OMX_CONTRAST ENHANCEMENT	
OMX_...	

OMX_IndexConfigImageDeNoiseLevel …….. Extended

OMX_IndexConfigSharpening …….. Extended

OMX_IndexConfigDeBlurring …….. Extended

OMX_IndexConfigChromaCorrection …….. Extended

OMX_IndexConfigDeMosaicing …….. Extended

OMX_IndexConfigSoftening …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_IMAGE_CONFIG_PROCESSINGLEVELTYPE**.

```
typedef  struct OMX_IMAGE_CONFIG_PROCESSINGLEVELTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_S32 nLevel;
OMX_BOOL bAuto;
} OMX_IMAGE_CONFIG_PROCESSINGLEVELTYPE;
```

The parameters for **OMX_CONFIG_PROCESSINGLEVELTYPE** are defined as follows.

- nLevel: hinting processing amount. Range of values is -100 to 100. 0 causes no change to the image. Increased values cause increased processing to occur, with 100 applying maximum processing. Negative values have the opposite effect of positive values.
-

-

- bAuto sets if the processing should be applied according to input data. It is allowed to combine the hint level with the auto setting, i.e. to give a bias to the automatic setting. When set to false, the processing should not take input data into account.

OMX_IndexConfigImageFaceDetection …….. Extended
OMX_IndexConfigImageBarcodeDetection …….. Extended
OMX_IndexConfigImageFrontObejectDetection …….. Extended
OMX_IndexConfigImageMotionEstimation …….. Extended
OMX_IndexConfigImageSmileDetection …….. Extended
OMX_IndexConfigImageFaceDetection …….. Extended
OMX_IndexConfigImageBlinkDetection …….. Extended
OMX_IndexConfigImageFaceDetection …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_OBJETDETECTIONTYPE**.

```
typedef  struct OMX_CONFIG_OBJETDETECTIONTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bFrameLimited;
OMX_U32 nFrameLimit;
OMX_BOOL bDetect;
OMX_U32 nMaxNbrObjects;
OMX_S32 nLeft;
OMX_S32 nTop;
OMX_U32 nWidth;
OMX_U32 nHeight;
OMX_OBJECTDETECTQUALITY eObjDetectQuality;
OMX_BOOL bSnapshot;
OMX_U32 nExtraPixels;
OMX_BOOL bTransperent;
OMX_U32 nPriority;
} OMX_CONFIG_OBJETDETECTIONTYPE;
```

The parameters for **OMX_CONFIG_OBJETDETECTIONTYPE** are defined as follows.

- nPortIndex is an output port. The port index decides on which port the extra data structur is sent on as well as snapshot images of detected object.

- bFrameLimited is a Boolean used to indicate if detection shall be terminated after the specified number of frames if true frame limited detection is enabled; otherwise the port does not terminate detection until instructed to do so by the client.
- nFrameLimit is the limit on number of frames detection is executed for, this parameter is only valid if bFrameLimited is enabled.
- bDetect is a Boolean that should be set to true when detection shall begin, otherwise set to false. Query will give status information on if detection is ongoing or not.
- nMaxNbrObjects specifies the maximum number of objects that should be found in each frame. It is implementation dependent which objects are found.
- nLeft is the leftmost coordinate of the detection area rectangle.
- nTop is the topmost coordinate of the detection area rectangle.
- nWidth is the width of the detection area rectangle in pixels.
- nHeight is the height of the detection area rectangle in pixels.
- eObjDetectQuality is an enumeration specifying the quality desired by the detection. The given table lists the supported values.

Table 6 Object Detection Quality Type Values

OMX_ OBJDETECTQUALITY Enumerated Value	**Description**
OMX_FASTDETECTION	A detection that prioritizes speed
OMX_DEFAULT	The default detection, should be used when no control of the detection quality is given.
OMX_BETTERDETECTION	A detection that levels correct detection with speed
OMX_BESTDETECTION	A detection that prioritizes correct detection
OMX_...	…
OMX_AUTODETECTION	Automatically decide which object detection quality is best.

- bSnapshot when true cropped images of the detected objects are delivered, when false only the location of the detected image in the larger image frame is delivered. When this is true and used on the measurement port of the camera component each detected object is delivered as a separate frame with reference to its placement in the large image (and timestamp of the frame). The image is placed starting from the top left corner of the image frame delivered. Image port frame size, scaling etc shall be meet.
- nExtraPixels specifies minimum extra pixels surrounding the object is needed only when used in combination with bSnapshot.
- bTransparent is only used when front object detection is supported. When this is true the pixels not part of the object are set to transparent (color). When used with bSnapshot set to true the cropped image is made transparent outside the object, otherwise the full image is made transparent outside the object.

- nPriority represents priority of each object when there are multiple objects detected.

OMX_IndexConfigImageHistogramMeasurement …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_HISTOGRAMTYPE**. This controls the histogram measurement.

```
typedef  struct OMX_CONFIG_HISTOGRAMTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bFrameLimited;
OMX_U32 nFrameLimit;
OMX_BOOL bMeasure;
OMX_U32 nBins;
OMX_S32 nLeft;
OMX_S32 nTop;
OMX_U32 nWidth;
OMX_U32 nHeight;
OMX_HISTTYPE eHistType;
} OMX_CONFIG_HISTOGRAMTYPE;
```

The parameters for **OMX_CONFIG_HISTOGRAMTYPE** are defined as follows.

- nPortIndex is an output port. The port index decides on which port the extra data structur is sent on.
- bFrameLimited is a Boolean used to indicate if measurement shall be terminated after the specified number of frames if true frame limited measurement is enabled; otherwise the port does not terminate measurement until instructed to do so by the client.
- nFrameLimit is the limit on number of frames measured, this parameter is only valid if bFrameLimited is enabled.
- bMeasure is a Boolean that should be set to true when measurement shall begin, otherwise set to false. Query will give status information on if measurement is ongoing or not.
- nBins specifies the number of histogram bins. When queried with set to zero, the respons gives the maximum number of bins allowed.
- nLeft is the leftmost coordinate of the measurement area rectangle.
- nTop is the topmost coordinate of the measurement area rectangle.
- nWidth is the width of the measurement area rectangle in pixels.
- nHeight is the height of the measurement area rectangle in pixels.

- eHistType is an enumeration specifying the histogram type, as shown in the table.

Table 7 Histogram Control Type Values

OMX_ HISTTYPE Enumerated Value	Description
OMX_HISTCONTROLLUMINANCE	Luminance histogram is calculated (Y)
OMX_HISTCONTROLCOLORCOMPONENTS	A histogram per color component (R, G, B) is calculated
OMX_HISTCONTROLCHROMINANCECOMPONENTS	A histogram per chrominance component (Cb, Cr) is calculated.

OMX_IndexConfigImageDistanceMeasurement …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_DISTANCETYPE**.

```
typedef  struct OMX_CONFIG_DISTANCETYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bStarted;
OMX_S32 nLeft;
OMX_S32 nTop;
OMX_U32 nWidth;
OMX_U32 nHeight;
OMX_DISTTYPE eDistType;
} OMX_CONFIG_DISTANCETYPE;
```

The parameters for **OMX_CONFIG_DISTANCETYPE** are defined as follows.

- bStarted is a Boolean. The IL client sets it to true to start the measurement . the IL client sets to false to stop the measurement. The IL client can query it to check if the measurement is ongoing.
- nLeft is the leftmost coordinate of the rectangle.
- nTop is the topmost coordinate of the rectangle.
- nWidth is the width of the rectangle in pixels.
- nHeight is the height of the rectangle in pixels.
- eDistType is an enumeration specifying the distance measurement type, as shown in the table.

Table 8 Distance Control Type Values

OMX_ DISTTYPE Enumerated Value	Description
OMX_DISTANCECONTROL_FOCUS	Distance evaluated based on the object in focus. May require that autofocus is active to obtain measurements. The defined rectangle can be used to select which focus region if several focus distance measurements available. Set region to cover whole image to obtain all measurments.
OMX_DISTANCECONTROL_RECT	Evaluated distance to the object found in the rectangelar area indicated as input region.

OMX_IndexConfigImageFocusLock …….. Extended

OMX_IndexConfigImageWhiteBalanceLock …….. Extended

OMX_IndexConfigImageExposureLock …….. Extended

OMX_IndexConfigImageAllLock …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_IMAGE_CONFIG_LOCKTYPE**.

```
typedef  struct OMX_IMAGE_CONFIG_LOCKTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bLock;
OMX_BOOL bAtCapture;
} OMX_IMAGE_CONFIG_LOCKTYPE;
```

The parameters used for **OMX_IMAGE_CONFIG_LOCKTYPE** are defined as followed.

- bLock controls if the camera control config/param are updated or not. When bLock is true no update is performed.
- bAtCapture controls if the lock is performed immediately (false) or from next captured image (true). The former corresponds to a half-press button the later corresponds to maintaining image settings between snapshots. This value is sampled when bLock is set to true. This value is ignored when bLock is false.

OMX_IndexConfigCommonWhiteBalanceGain …….. Extended

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_WHITEBALGAINTYPE**.

```
typedef  struct OMX_CONFIG_WHITEBALGAINTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_S32 xWhiteBalanceGain[4];
OMX_S32 xWhiteBalanceOffset[4];
OMX_S32 nWhiteThreshold[4];
} OMX_CONFIG_WHITEBALGAINTYPE;
```

The parameters used for **OMX_CONFIG_WHITEBALGAINTYPE** are defined as follows.

- xWhiteBalanceGain and xWhiteBalanceOffset represents gain and offset for R, Gr, Gb, B channels respectively in Q16 format. For example, new red pixel value = xWhiteBalanceGain[1]* the current pixel value + xWhiteBalanceOffset[1]. All values assume that maximum value is 255. If internal implementation uses higher dynamic range, this value should be adjusted internally.
- nWhiteThreshhold represents thresholds for “white” area measurments in Q16 format.

OMX_OTHER_EXTRADATATYPE

- The **OMX_OTHER_EXTRADATATYPE** structure is used to describe the additional buffer payload information included within the buffer. A buffer may contain multiple blocks of extra data and thus multiple instances of this structure.
- Each additional EXTRADATATYPE structure shall be required to be 32 bit address aligned, and padding bytes may need to inserted in order to ensure this alignment.
- The order of the additional information is not required to be pre-determined since a component is expected to traverse the **OMX_OTHER_EXTRADATATYPE** structures to determine the additional information of interest.
- Meta data utilizes the **OMX_OTHER_EXTRADATATYPE** structure.

```
typedef  struct OMX_OTHER_EXTRADATATYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
```

OMX_EXTRADATATYPE eType;

OMX_U32 nDataSize;

OMX_U8 Data[1];

} **OMX_OTHER_EXTRADATATYPE;**

The parameters for **OMX_OTHER_EXTRADATATYPE** are defined as follows.

- nSize is the size of the structure including data bytes and any padding necessary to ensure 32bit alignment of the next OMX_OTHER_EXTRADATATYPE structure.
- nPortIndex is the read-only value containing the index of the port.
- eType identifies the extra data payload type.

Table 9 Extended Extra Data Payload Type Enumerated Values

Enumerated Value	Description
OMX_ExtraDataNone	Indicates that this terminates the list of extra data sections.
OMX_ExtraDataQuantization	Indicates that the data payload contains quantization data.
OMX_EXIFAttributes	Indicates that the data payload contains EXIF formatted meta data as described in EXIF 2.2 (ref) for the main attributes, potentially covering: image size, format, title, make/model, orientation, creater, copyright. Image data structure, data characteristics, time, model, title, copyright, and picture taking conditions such as exposure, F-number, exposure prg, spectra lsensitivity, ISO speed rating, OECF, shutter speed, aperture, brightness, exposure bias, max lens aperature, subject distance, metering mode, light source, flash, lens focal length, subject area, flash energy, spatial freq respons, focal plane x/y resolution, focal plane resolution unit, subject location, exposure index, sensing method, file source, scene type, CFA pattern, custom image processing, exposure mode, white balance, digital zoom ratio, scene capture type, gain control, saturation, sharpness, subject distance range, etc potentially covering GPS location, direction, time, speed, etc.
OMX_FACEDETECTION	Indicates that the data payload contains face detection data
OMX_BARCODEDETECTION	Indicates that the data payload contains barcode detection data
OMX_FRONTOBJDETECTION	Indicates that the data payload contains the front/back object detection data
OMX_MOTIONESTIMATION	Indicates that the data payload contains the motion estimation measured data
OMX_DISTANCEESTIMATION	Indicates that the data payload contains a distance measurement.

OMX_HISTOGRAM	Indicates that the data payload contains the histogram measured data
OMX_FOCUSREGION	Indicates that the data payload contains the focus region data
OMX_ExtraDataPanAndScan	Indicates that the data payload contains the pan&scan cropping information data. Croping is only applied by components for which pan&scan cropping extended config/param is activated.
OMX_RAWFormat	Vendor specific identifier for the custom RAW format
OMX_SensorType	Vendor and model of the sensor being used.
OMX_SensorCustomDataLength	Length of vendor specific custom data for the sensor
OMX_SensorCustomData	vendor specific custom data for the sensor

- nDataSize identifies the size of supporting data in units of bytes. For the OMX_OTHER_EXTRADATATYPE structure that terminates the list of extra data sections, nDataSize will be zero.
- Data is an array of one or more bytes of data as indicated by the nDataSize field.

Each of these will now be described with separate structures that starts at the data[1] element position of the OMX_OTHER_EXTRADATATYPE structure. All the EXIF related extra data format is covered by the EXIF specification.

OMX_IndexConfigOtherExtraDataControl

```
/**
 * The OMX_EXTRADATATYPE enumeration is used to define the
 * possible extra data payload types.
 */
typedef enum OMX_EXT_EXTRADATATYPE
{
  OMX_ExifAttributes = 0x7F000001, /**< Reserved region for introducing Vendor Extensions */
  OMX_FaceDetection, /**< face detect data */
  OMX_BarcodeDetection, /**< bar-code detct data */
  OMX_FrontObjectDetection, /**< Front object detection data */
  OMX_MotionEstimation, /**< motion Estimation data */
  OMX_DistanceEstimation, /**< disctance estimation */
  OMX_Histogram, /**< histogram */
  OMX_FocusRegion, /**< focus region data */
  OMX_RawFormat, /**< custom RAW data format */
  OMX_SensorType, /**< vendor & model of the sensor being used */
  OMX_SensorCustomDataLength, /**< vendor specific custom data length */
  OMX_SensorCustomData /**< vendor specific data */
} OMX_EXT_EXTRADATATYPE;
```

```
/**
 * Enable Extra-data on a specific port.
 *
 *
 *
 * STRUCT MEMBERS:
 *  nSize       : Size of the structure in bytes
 *  nVersion    : OMX specification version information
 *  nPortIndex  : Port on which this extra data to be assosiated
 *  eExtraDataType :  Extra data type
 *  bEnable     : Eneble/Disable this extra-data through port.
 *
 */
typedef struct OMX_CONFIG_EXTRADATATYPE {
   OMX_U32 nSize;
   OMX_VERSIONTYPE nVersion;
   OMX_U32 nPortIndex;
   OMX_EXT_EXTRADATATYPE eExtraDataType;
   OMX_BOOL bEnable;
} OMX_CONFIG_EXTRADATATYPE;
```

Face detection data

The extra data having face detection data is described with the following structure. Multiple faces can be found and then this structure is repeated.

```
typedef  struct OMX_FACEDETECTIONTYPE {
OMX_U32  nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_U32 nScore;
OMX_S32 nLeft ;
OMX_S32 nTop ;
OMX_U32 nWidth;
OMX_U32 nHeight;
OMX_S32 nOrientationRoll;
OMX_S32 nOrientationYaw;
OMX_S32 nOrientationPitch;
```

OMX_U32 nPriority;

OMX_FACEATTRIBUTE nFaceAttr;

} **OMX_FACEDETECTIONTYPE;**

The parameters for **OMX_FACEDETECTIONTYPE** are defined as follows.

- nScore is a detection score between 0 and 100, where 0 means unknown score, 1 means least certain and 100 means most certain the detection is correct.
- nLeft is the leftmost coordinate of the detected area rectangle.
- nTop is the topmost coordinate of the detected area rectangle.
- nWidth is the width of the detected area rectangle in pixels.
- nHeight is the height of the detected area rectangle in pixels.
- nOrientationRoll/Yaw/Pitch is the orientation of the axis of the detected object. Here roll angle is defined as the angle between the vertical axis of face and the horizontal axis. All angles can have the value of -180 to 180 degree in Q16 format. Some face detection algorithm may not be able to fill in the angles, this is denoted by the use of MAX_INT value.
- nPriority represents priority of each object when there are multiple objects detected.
- nFaceAttr describe the attributes of the detected face object with the following structure:

typedef struct OMX_FACEATTRIBUTE {
OMX_U32 nARGBEyeColor;

OMX_ U32 nARGBSkinColor;

OMX_ U32 nARGBHairColor;

OMX_ U32 nSmileScore;

OMX_ U32 nBlinkScore;

OMX_U32 xIdentity[4];

} OMX_FACEATTRIBUTE;

The parameters within OMX_FACEATTRIBUTE are defined as follow:

- nARGBEyeColor is the indicates a 32-bit eye color of the person, where bits 0-7 are blue, bits 15-8 are green, bits 24-16 are red, and bits 31-24 are for alpha.
- nARGBSkinColor is the indicates a 32-bit skin color of the person, where bits 0-7 are blue, bits 15-8 are green, bits 24-16 are red, and bits 31-24 are for alpha.
- nARGBHairColor is the indicates a 32-bit hair color of the person, where bits 0-7 are blue, bits 15-8 are green, bits 24-16 are red, and bits 31-24 are for alpha.
- nSmileScore is a smile detection score between 0 and 100, where 0 means not detecting, 1 means least certain and 100 means most certain a smile is detected.
- nBlinkScore is a eye-blink detection score between 0 and 100, where 0 means not detecting, 1 means least certain and 100 means most certain an eye-blink is detected.

- xIdentity represents the identity of the face. With identity equal to zero this is not supported. This can be used by a face recognition application. The component shall not reuse an identity value unless the same face. Can be used to track detected faces when it moves between frames. Specific usage of this field is implementation dependent. It can be some kind of ID.

Barcode detection data

The extra data having in barcode detection data is described with the following structure.

typedef struct OMX_BARCODEDETECTIONTYPE {

OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_S32 nLeft;
OMX_S32 nTop;

OMX_U32 nWidth;
OMX_U32 nHeight;

OMX_S32 nOrientation;

OMX_BARCODETYPE eBarcodetype;

} OMX_BARCODEDETECTIONTYPE;

The parameters for OMX_BARCODEDETECTIONTYPE are defined as follows.

- nLeft is the leftmost coordinate of the detected area rectangle.
- nTop is the topmost coordinate of the detected area rectangle.
- nWidth is the width of the detected area rectangle in pixels.
- nHeight is the height of the detected area rectangle in pixels.
- nOrientation is the orientation of the axis of the detected object. This refers to the angle between the vertical axis of barcode and the horizontal axis.
- eBarcodetype is an enumeration specifying the barcode type, as listed in the given table.

Table 10 Barcode Type Values

OMX_ BARCODETYPE Enumerated Value	Description
OMX_BARCODE1D	1D barcode
OMX_BARCODE2D	2D barcode

Front object detection

The extra data having front object detection data is described with the following structure. Multiple front objects may be found and then this structure is repeated. This may beneficially be combined with the bTransparent setting and bSnapshot setting to obtain images of the front object separated from the background.

typedef struct OMX_FRONTOBJDETECTIONTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_S32 nLeft;

OMX_S32 nTop;
OMX_U32 nWidth;
OMX_U32 nHeight;
} **OMX_FRONTOBJDETECTIONTYPE**;

The parameters for OMX_FRONTOBJDETECTIONTYPE are defined as follows.

- nLeft is the leftmost coordinate of the detected area rectangle.
- nTop is the topmost coordinate of the detected area rectangle.
- nWidth is the width of the detected area rectangle in pixels.
- nHeight is the height of the detected area rectangle in pixels.

Distance estimation

The extra data having in disatnce estimation data is described with the following structure.

typedef struct OMX_DISTANCEESTIMATIONTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_U32 nDistance;

OMX_U32 nLargestDiscrepancy;
} **OMX_DISTANCEESTIMATIONTYPE**;

The parameters for OMX_DISTANCEESTIMATIONTYPE are defined as follows.

- nDistance is the estimated distance to the object in millimeters.
- nLargestDiscrepancy is the estimated largest discrepancy of the distance to the object in millimeters. When equal to MAX_INT the discrepancy is unknown.

Motion estimation

The extra data having in motion estimation data is described with the following structure.

typedef struct OMX_MOTIONESTIMATIONTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_S32 nPanX;
OMX_S32 nPanY;
} OMX_MOTIONESTIMATIONTYPE;

The parameters for OMX_MOTIONESTIMATIONTYPE are defined as follows.

- nPanX is the detected translation in horizontal direction. The value is represented as pixels in Q16-format.
- nPanY is the detected translation in vertical direction. The value is represented as pixels in Q16-format.

Histogram measurement data

The extra data having histogram estimation data is described with the following structure.

typedef struct OMX_HISTOGRAMTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_U32 nBins;

OMX_HISTCOMPONENTTYPE eComponentType;

OMX_U8 data[1];
} OMX_HISTOGRAMTYPE;

The parameters for OMX_HISTOGRAMTYPE are defined as follows.

- nSize is the size of the structure including the length of data field containing the histogram data.
- nBins is the number of bins in the histogram.
- eComponentType specifies the type of the histogram bins according to enum. It can be selected to generate multiple component types, then the extradata struct is repeated for each component type.

Table 11 Histogram Type Values

OMX_ HISTTYPE Enumerated Value	Description
OMX_HISTCOMP_Y	Luminance histogram (Y)
OMX_HISTCOMP_YLOG	Logarithmic luminance histogram (Y)
OMX_HISTCOMP_R	Red histogram component (R)
OMX_HISTCOMP_G	Green histogram component (G)
OMX_HISTCOMP_B	Blue histogram component (B)
OMX_HISTCOMP_Cb	Chroma blue histogram component (Cb)
OMX_HISTCOMP_Cr	Chroma red histogram component (Cr)

- data[1] first byte of the histogram data

Focus region data

The extra data having focus region detection data is described with the following structure.

typedef struct OMX_FOCUSREGIONTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_U32 nRefPortIndex;

OMX_S32 nLeft;

OMX_S32 nTop;
OMX_U32 nWidth;
OMX_U32 nHeight;
} OMX_FOCUSREGIONTYPE;

The parameters for OMX_FOCUSREGIONTYPE are defined as follows.

- nRefPortIndex is the port the image frame size is defined on. This image frame size is used as reference for the focus region rectangle.
- nLeft is the leftmost coordinate of the focus region rectangle.
- nTop is the topmost coordinate of the focus region rectangle.
- nWidth is the width of the focus region rectangle in pixels.
- nHeight is the height of the focus region rectangle in pixels.

OMX_Index_Config_CommonRotation

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_ROTATIONTYPE**. Rotation rotates video or image frames clockwise by a specified angle.

OMX_CONFIG_ROTATIONTYPE is defined as

typedef struct OMX_CONFIG_ROTATIONTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_S32 nRotation;

} OMX_CONFIG_ROTATIONTYPE;

The parameters for **OMX_CONFIG_ROTATIONTYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- nRotation is an integer value that represents the angle of rotation. Some components may only support rotation on right angles such as 0°, 90°, 180°, and 270°. Rotation is clockwise.

OMX_Index_Config_CommonMirror

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_MIRRORTYPE**. Mirroring is applied to pixel or image data on a specified port. The data can be mirrored in the horizontal direction, vertical direction, or both horizontal and vertical directions.

OMX_CONFIG_MIRRORTYPE is defined as

typedef struct OMX_CONFIG_MIRRORTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_MIRRORTYPE eMirror;

} OMX_CONFIG_MIRRORTYPE;

The parameters for **OMX_CONFIG_MIRRORTYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- eMirror contains the enumerated values indicating the mirroring applied to image or video data. OMX_MirrorNone is used to disable mirroring or have no mirroring. Table 4-35 identifies the mirroring values.

Table 12 Mirror Type Values

OMX_MIRRORTYPE Enumerated Value	Description
OMX_MirrorNone	Disables mirroring (i.e., no mirroring).
OMX_MirrorHorizontal	Mirrors pixels in the horizontal direction. Hence, pixel at 0,1 is swapped with pixel W,1 where W is the width of the image.
OMX_MirrorVertical	Mirrors pixels in the vertical direction. Hence, pixel at 1,0 is swapped with pixel 1,H where H is the height of the image.
OMX_MirrorBoth	Mirrors pixels in the horizontal and vertical directions. Hence, pixel at 0, 0 is swapped with pixel W,H where W is the width of the image and H is the height of the image.

OMX_IndexConfigCommonInputCrop

OMX_IndexConfigCommonOutputCrop

Used with OMX_GetConfig and OMX_SetConfig to access **OMX_CONFIG_RECTTYPE**. Crops the image or video streams to the specified angle.

Rectangles are used with several configuration types to indicate orientation, position, inclusion, or exclusion.

OMX_CONFIG_RECTTYPE is defined as follows.

typedef struct OMX_CONFIG_RECTTYPE {
OMX_U32 nSize;

OMX_VERSIONTYPE nVersion;

OMX_U32 nPortIndex;

OMX_S32 nLeft;

OMX_S32 nTop;

OMX_U32 nWidth;

OMX_U32 nHeight;

} OMX_CONFIG_RECTTYPE;

The parameters for **OMX_CONFIG_RECTTYPE** are defined as follows.

- nPortIndex is the read-only value containing the index of the port.
- nLeft is the leftmost coordinate of the rectangle.
- nTop is the topmost coordinate of the rectangle.
- nWidth is the width of the rectangle in pixels.
- nHeight is the height of the rectangle in pixels.

OMX_IndexParamHighISONoiseFilter:

```
/**
 * High ISO Noise filter mode range enum
 */
typedef enum OMX_ISONOISEFILTERMODETYPE{
   OMX_ISONoiseFilterModeOff = 0,
   OMX_ISONoiseFilterModeOn,
   OMX_ISONoiseFilterModeAuto,
   OMX_ISONoiseFilterModeExtensions = 0x6F000000,    /** Reserved region for
introducing Khronos Standard Extensions */
   OMX_ISONoiseFilterModeStartUnused = 0x7F000000,   /** Reserved region for
introducing Vendor Extensions */
   OMX_ISONoiseFilterModeMax = 0x7FFFFFFF
} OMX_ISONOISEFILTERMODETYPE;


/**
 * Enable ISO noise filter.
 *
 * STRUCT MEMBERS:
 *  nSize      : Size of the structure in bytes
 *  nVersion   : OMX specification version information
 *  nPortIndex  : Port that this structure applies to
 *  eMode      : ISO noise filter (NSF2 is used) mode (on/off/auto)
 */
typedef struct OMX_PARAM_ISONOISEFILTERTYPE {
   OMX_U32 nSize;
   OMX_VERSIONTYPE nVersion;
   OMX_U32 nPortIndex;
   OMX_ISONOISEFILTERMODETYPE eMode;
} OMX_PARAM_ISONOISEFILTERTYPE;
```

OMX_IndexConfigISOSetting:

For setting the ISO for sensor.

```
/**
 * OMX_ISOSETTINGTYPE: specifies its auto or manual setting
 *
 */
```

```
typedef enum OMX_ISOSETTINGTYPE{
    OMX_Auto = 0, /**<          */
    OMX_Manual      /**< */
}OMX_ISOSETTINGTYPE;

/**
 *  nSize is the size of the structure including the length of data field containing
 *  the histogram data.
 *  eISOMode:
 *      specifies the ISO seetting mode - auto/manual
 *  nISOSetting:
 *      for manual mode client can specify the ISO setting.
 */

typedef struct OMX_CONFIG_ISOSETTINGTYPE{
        OMX_U32 nSize;
        OMX_VERSIONTYPE nVersion;
        OMX_U32 nPortIndex;
        OMX_ISOSETTINGTYPE eISOMode;
        OMX_U32 nISOSetting;
}OMX_CONFIG_ISOSETTINGTYPE;
```

8 OpenMAX and use case interaction

This section tries to go through some sample use-case. This is covering mostly the way IL-Client tries to setup and run the use-case. For more details on the internal architecture refer to

8.1 Use-cases nature in this OpenMAX Camera component.

All use-cases which are going to execute would be pre-defined usecases. This is due to the face that, it is not possible to get one total new usecase pipeline up in a module level pick choose and connect manner.

There are lot of hidden details of interconnection especially in and around sensor, ISP which would be very difficult to generalize. Flexibility wise what this architecture would help is to enable integration of a 3rd party algorithm through a standard interface (MSP).

Different use-cases which are supported in this component would be,

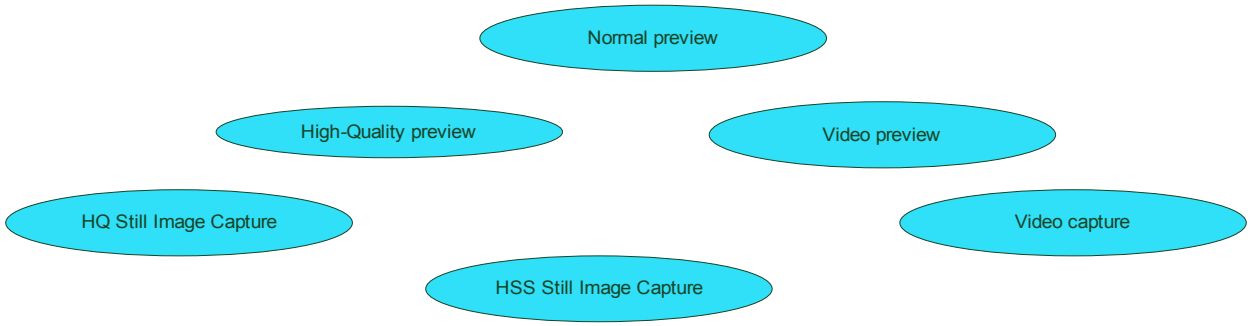


Figure 7 Different basic use-cases

Of these HSS (High Speed Still Image Capture) usecase and all very much tightly coupled implementation. There won't be any option of adding any extra algorithm along this piepeline. It would be to the HQ (High Quality) usecase pipeline, a new 3rd party processing algorithm would be inserted (statically).

8.2 Arriving at the intended use-case from OMX parameters

OpenMAX natively doesn't have any use-case specific parameters. Each of the parameters are independent configuration/parameters. So, need to derive the intended use-case from those OMX-parameters passed from IL-client. So, when moving from LOADED to IDLE, this parsing logic would analyze and detects which is the intended use-case.

Another important point is that, this component is a multiple functionality component. Which could pereform capture, post-processing and JPEG encoding functionality. Thus it would be easier from the internal logic point of view to classify this functionalities to be classified as the OpenMAX component standard roles. Three roles are,

- Camera.yuv (w. jpeg encode support)
- Iv_processor.yuv
- Image_encoder.jpeg (Memory IO)

The same can be set or queried by ***OMX_IndexParamStandardComponentRole.***

At the beginning all ports would be enabled, then depending on the use-case need client disable which ever ports are not needed. This would give a hint on the probable use-case. And

.1

.2

after one of the roles is set, its easier to go ahead with setting up the pipeline for a specific use-case. But in the capture use-cases there would be many parameter/configs such that, it would be difficult to parse each dependency, analyze each combination and derive the intended use-case. More over, IL-client's small mistake in filling in some parameter would be misinterpreted as a wrong use-case. In this case, defining the camera operating mode is a better solution to reduce the complexity of the underlying camera manager in setting up the use-case pipeline. In the future there could be additional use-case pipelines coming in.

Suggested interface for this would look like this,

/* enum types --- sample */
typedef enum ***OMX_CAMOPERATINGMODETYPE*** *{*

OMX_CaptureImageHighSpeedBurst = 0,

```
    OMX_CaptureImageHighSpeedTemporalBracketing,
    OMX_CaptureImageHighQualityProfile1,
    OMX_CaptureImageHighQualityProfile2,
    OMX_CaptureImageHighQualityProfile3,
    OMX_CaptureMemoryInput,
    OMX_CaptureVideo,
    OMX_CaptureHighSpeedVideo,
    OMX_CamOperatingModeMax = 0x7fffffff
} OMX_CAMOPERATINGMODETYPE;

typedef struct OMX_CONFIG_CAMOPERATINGMODETYPE {
  OMX_U32 nSize;
  OMX_VERSIONTYPE nVersion;
  OMX_CAMOPERATINGMODETYPE eCamOperatingMode;
} OMX_CONFIG_CAMOPERATINGMODETYPE;
```

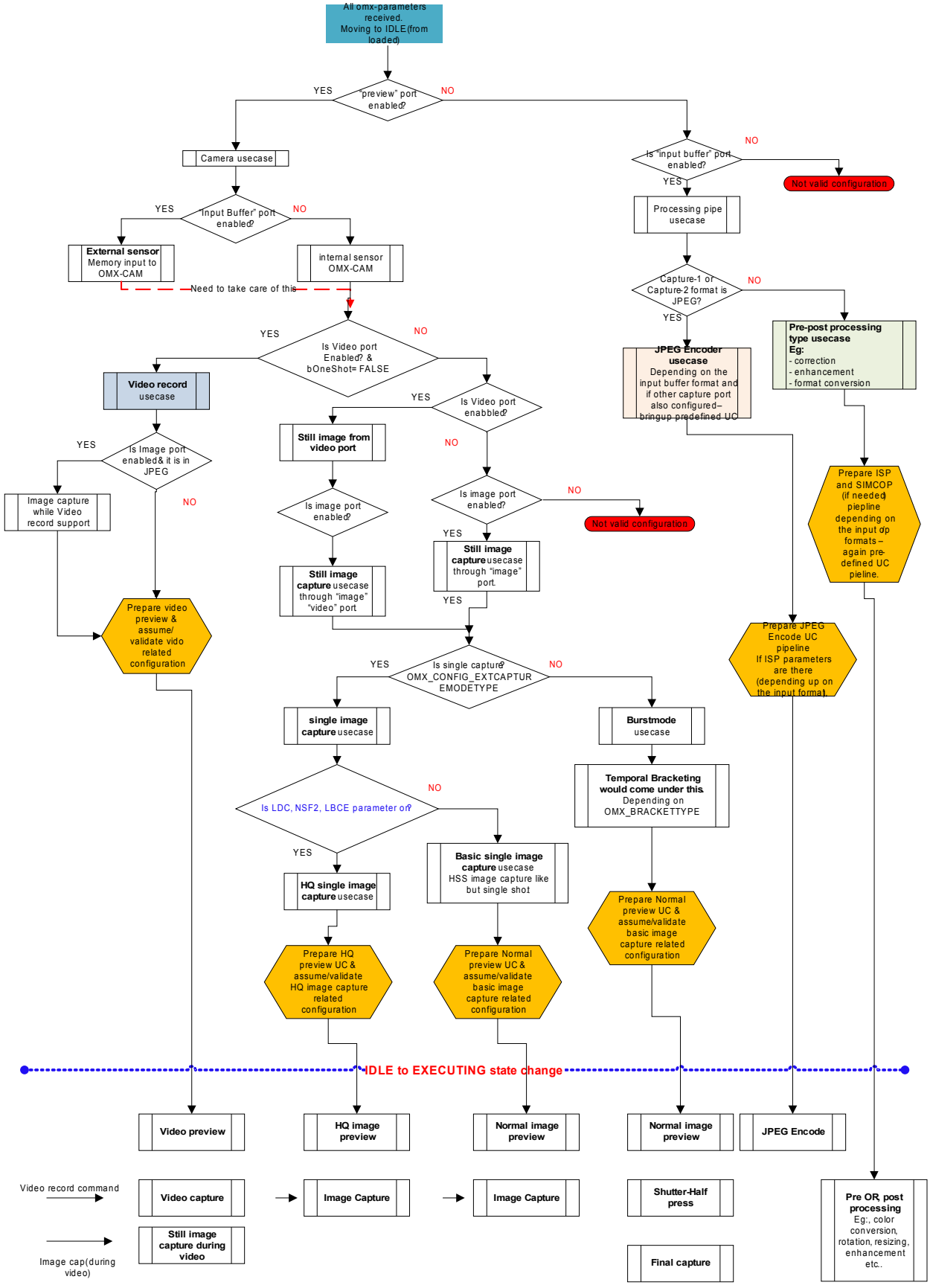

All omx-parameters received. Moving to IDLE(from loaded)
"preview" port enabled?
YES
NO
Camera usecase
"Input Buffer" port enabled?
YES
NO
External sensor Memory input to OMX-CAM
internal sensor OMX-CAM
Need to take care of this
Is Video port Enabled? & bOneShot= FALSE
YES
NO
Video record usecase
Is Image port enabled& it is in JPEG
YES
NO
Image capture while Video record support
Prepare video preview & assume/ validate vido related configuration
Still image from video port
Is image port enabled?
Still image capture usecase through "image" "video" port
Is Video port enabbled?
Is image port enabled?
Not valid configuration
Still image capture usecase through "image" port.
Is single capture? OMX_CONFIG_EXTCAPTUREMODETYPE
single image capture usecase
Is LDC, NSF2, LBCE parameter on?
HQ single image capture usecase
Basic single image capture usecase HSS image capture like but single shot
Prepare HQ preview UC & assume/validate HQ image capture related configuration
Prepare Normal preview UC & assume/validate basic image capture related configuration
"Input buffer" port enabled?
Not valid configuration
Processing pipe usecase
Capture-1 or Capture-2 format is JPEG?
JPEG Encoder usecase Depending on the input buffer format and if other capture port also configured– bringup predefined UC
Pre-post processing type usecase Eg: - correction - enhancement - format conversion
Prepare ISP and SIMCOP (if needed) piepline depending on the input dp formats – again pre-defined UC pieline.
Prepare JPEG Encode UC pipeline If ISP parameters are there (depending up on the input format)
Burstmode usecase
Temporal Bracketing would come under this. Depending on OMX_BRACKETTYPE
Prepare Normal preview UC & assume/validate basic image capture related configuration
IDLE to EXECUTING state change
Video preview
HQ image preview
Normal image preview
Normal image preview
JPEG Encode
Video record command
Image cap(during video)
Video capture
Still image capture during video
Image Capture
Image Capture
Shutter-Half press
Final capture
Pre OR post processing Eg:, color conversion, rotation, resizing, enhancement etc..

Figure 8 Parsing routine to arrive at the intended usecase.(THIS IS OUT OF THE DESIGN since, camera operating mode is introduced which would reduce the detailed parsing)

8.3 Still Image Capture Use case

8.3.1 Capture trigger for the usecase

Traditionally, OMX_IndexConfigCapturing {OMX_CONFIG_BOOLEANTYPE} is used to trigger the camera for capture. There is no trigger for the preview usecase to start. And things get more complex when multiple capture ports are there. For eg: usecase like, still image capture during video record, there has to be a separate trigger for video and image. Better solution could be to associate capture trigger to the specific port. So that the implementation could decide which is the trigger for. For this there is an extended structure defined referred by OMX_IndexConfigExtCapturing.

In addition to this there is a need to differentiate Sutter Half-Press event from IL-client. This is achieved by incorporating a new BOOLEAN type configuration index OMX_IndexConfigShutterHalfPress. Refer to section 1.1.3for details.

8.3.2 Single image capture use-case

This could be the most commonly used scenario, where JPEG file is saved as the final image on receiving the capture command.

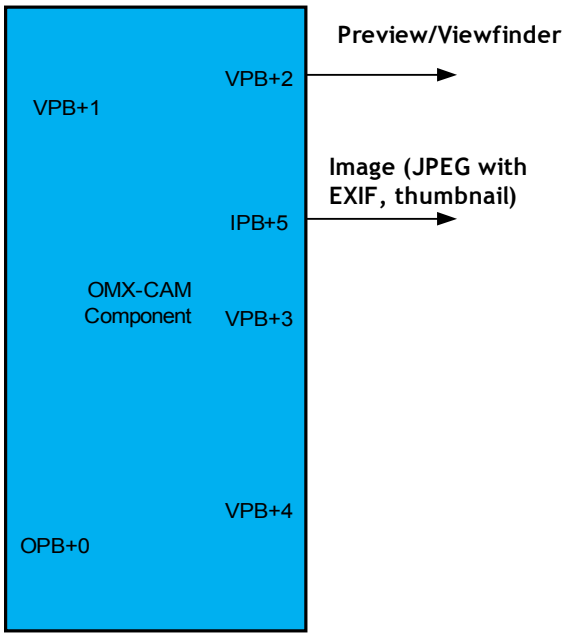


Figure 9

An IL client using a camera to capture an image may fallow the following steps:

1. Instantiate the camera component and any co-operating components
2. Set camera parameters:
   In loaded state, all camera ports would be enabled. So, for specific use-case disable ports which are not needed using OMX_CommandPortDisable.

   - Set image port resolution & format according to desired values of captured image.

   - Set preview/viewfinder port resolution and frame-rate (e.g. according to desired values of preview windows).

- Set the one shot bit of the sensor mode specific to “Preview port” is set to FALSE to indicate that the camera should emit multiple frames depending on thee frame rate specified on that port.

The IL client should leave the sensor resolution at the default allowing the camera to pick a sensor resolution at the default allowing the camera to pick a sensor resolution appropriate to the resolution settings of the preview/viewfinder and the capture ports.

- Set other camera settings (e.g. exposure value compensation, white balance, zoom, etc)

- Set or clear auto pause after capture accordingly. If auto pause is set the component will stop emitting viewfinder (preview port) after a capture

3. Establish any necessary tunnels between the camera component and other components (e.g. a display component tunneling with the viewfinder port or a image encoder tunneling with the capture port).
4. Transition all components to the OMX_StateIdle state this place derived layer parses the omx parameters and come to know the usecase to be executed.
5. and then to the OMX_StateExecuting state. The viewfinder port should now be actively emitting preview frames and the capture port is not transmitting any frames, it is paused.
6. With the viewfinder port enabled, the IL client now has the opportunity to performing any zoom and focus related actions or to change the capture resolution.
7. If shutter half press is commanded through OMX_IndexConfigShutterHalfPress, then respective shutter half press mode is executed. Mostly, face detect, 3A algorithms are triggered in this phase.
8. To signal image capture set the capturing bit. The capture port will emit a single captured frame and then the component will immediately clear the capturing bit. If set to auto pause after capture the component will cease the emission of frames through preview port. This effectively freezes any associated preview window to the captured image frame. If auto pause is clear then the viewfinder continues emitting preview frames.

8.3.2.1 Example sequence call from IL-client

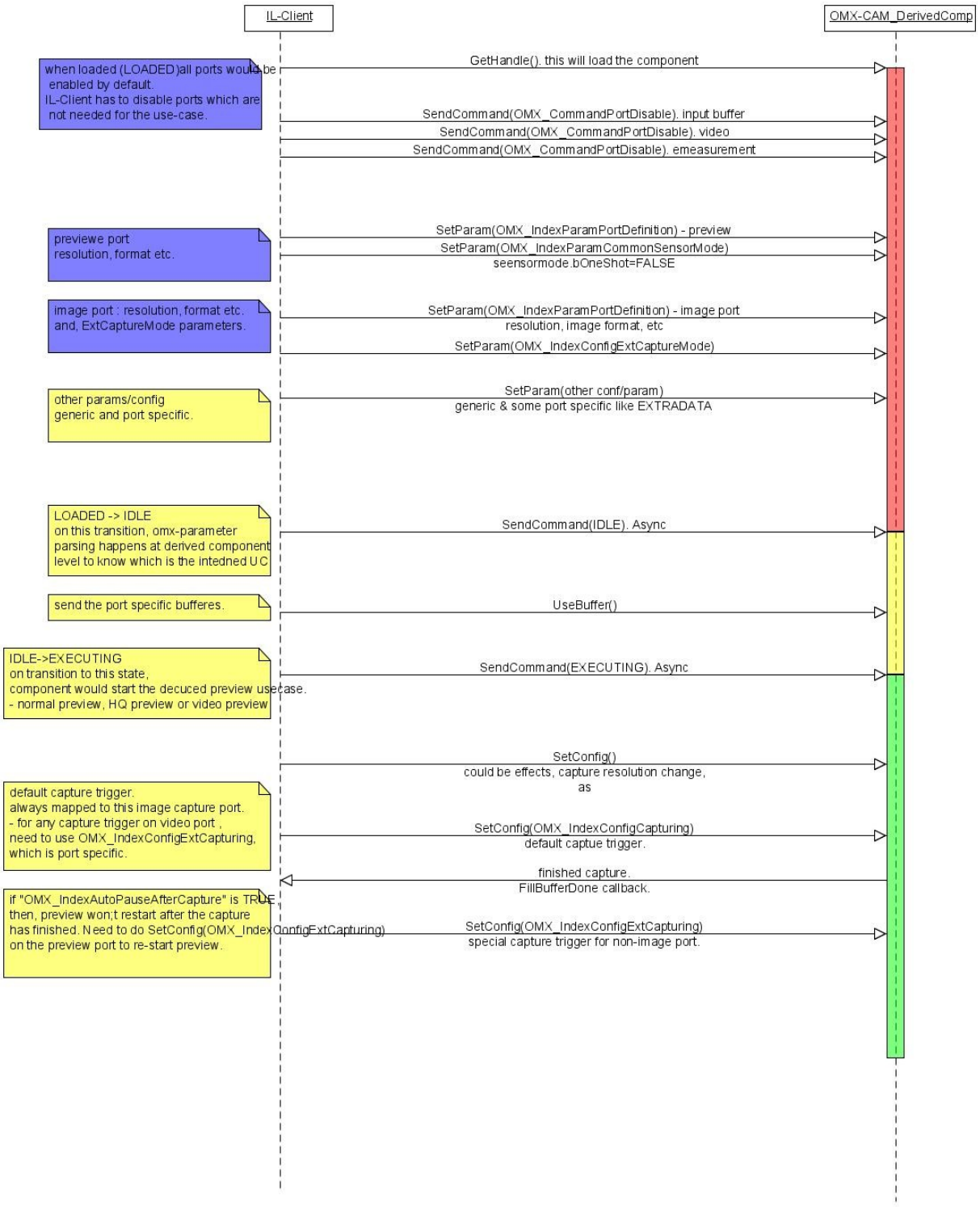


Figure 10

8.3.3 Burst image capture

This case is same as single image capture except that additional parameters has to be filled in OMX_CONFIG_EXTCAPTUREMODETYPE. So, on receiving the capture command, camera component would start emitting specified number of frames.

Depending on the eBracketMode different bracketing modes are executed on receiving capture command. If this mode is, OMX_BracketTemporal then Temporal Bracketing phase of execution has to be launched on half-press.

8.4 Face Detection Use Cases

Face detect Algorithm is controlled from Il-Client through OMX_IndexConfigImageFaceDetection and the respective structure used is, OMX_CONFIG_OBJDETECTIONTYPE. Which controls ON/OFF nature of this algorithm. Number of faces to be detected, quality of detection etc. Output of this face detect info is emitted out of the component as OTHER_EXTRADATA through the port specified through OMX_CONFIG_OBJDETECTIONTYPE.

8.5 Video Capture Use Cases

1. In loaded state, disable inut buffer, image ports.
2. set sensor mode flag for preview port, video port and measurement port to be FALSE.
3. set other config params needed for the video usecase.
4. if Video stabilization or preview viewfinder stabilization has to be ON, configure omponent with OMX_IndexConfigCommonFrameStabilisation with the value as TRUE.
5. Moving from loaded to idle would analyze and decude that this is video usecase.
6. while moving from idle to executing it launches video preview usecase.
7. for start capturing video, IL-client issues OMX_IndexConfigExtCapturing to video port with valus as TRUE.
8. To stop capturing, IL-client issues OMX_IndexConfigExtCapturing to video port with valus as FALSE.
9. If PauseAfterCapture is enabled, then IL-client needs to issue OMX_IndexConfigExtCapturing to preview port with value as TRUE. This would start emitting viewfinder frames from preview port.
10. To exit out of the application, IL-client moves the component from executing to idle to loaded.

8.6 Image capture while video record use-case

1. In loaded state, disable inut buffer.
2. set sensor mode flag for preview port, video port and measurement port to be FALSE.
3. set other config params needed for the video usecase.

4. Moving from loaded to idle would analyze and decude that this is video usecase.
5. while moving from idle to executing it launches video preview usecase.
6. for start capturing video, IL-client issues OMX_IndexConfigExtCapturing to video port with valus as TRUE.
7. During this video capture IL-client issues OMX_IndexConfigCapturing to image port with the value as TRUE.
8. To stop capturing, IL-client issues OMX_IndexConfigExtCapturing to video port with values as FALSE.
9. If PauseAfterCapture is enabled, then IL-client needs to issue OMX_IndexConfigExtCapturing to preview port with value as TRUE. This would start emitting viewfinder frames from preview port.
10. To exit out of the application, IL-client moves the component from executing to idle to loaded.

9 Internal Architecture Details

This shows the internal module division for the camera architecture. This has taken care how the legacy architecture is organized also.

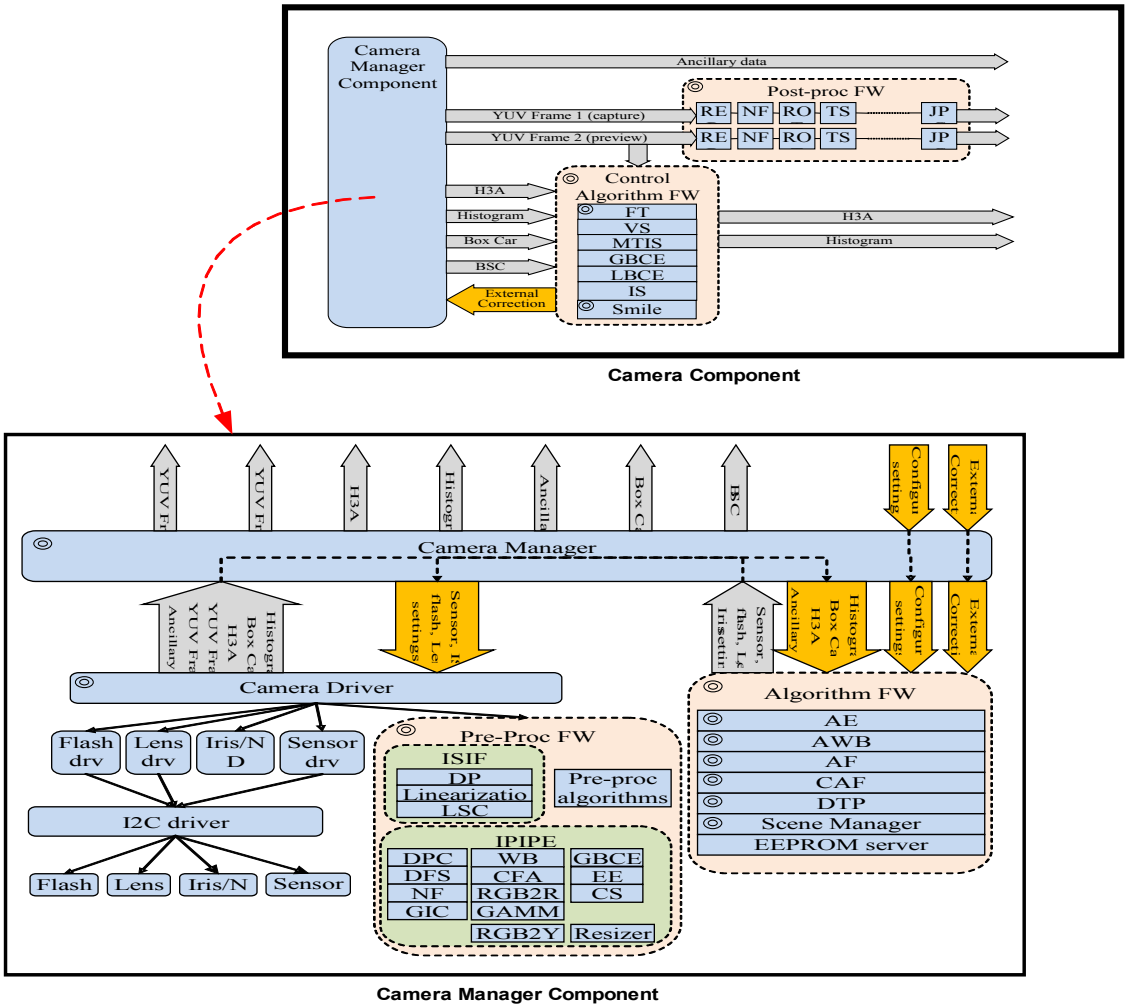


Figure 11 Internal modules information

All internal module interface is going to be MSP. OpenMAX configurations and parameters received from the IL-client would reach OMX-Derived layer (refer to OMX-Base, Derived concepts) and the same would be populated in component private area. Once the component moved to IDLE from LOADED, it does the parsing to get to know the intended use-case to be run.

And on receiving IDLE to ECECUTING state, derived layer selects the usecase and creates it. Infact is nothing newly created, its all predefined, just preparing the use-case by selecting the predefined use-case, which in turn initializes underlined MSP components. After that, it kick start the usecase in a predefined manner. Once this pipeline is in running, the specific use-case manager would be listening to the events from the contained MSP components and passing it on to the next. At that time derive layer context could be receiving commands from the IL-client, derived layer could decide on those commands is to be passed on to the currently running specific usecase-manager layer or not to service it.

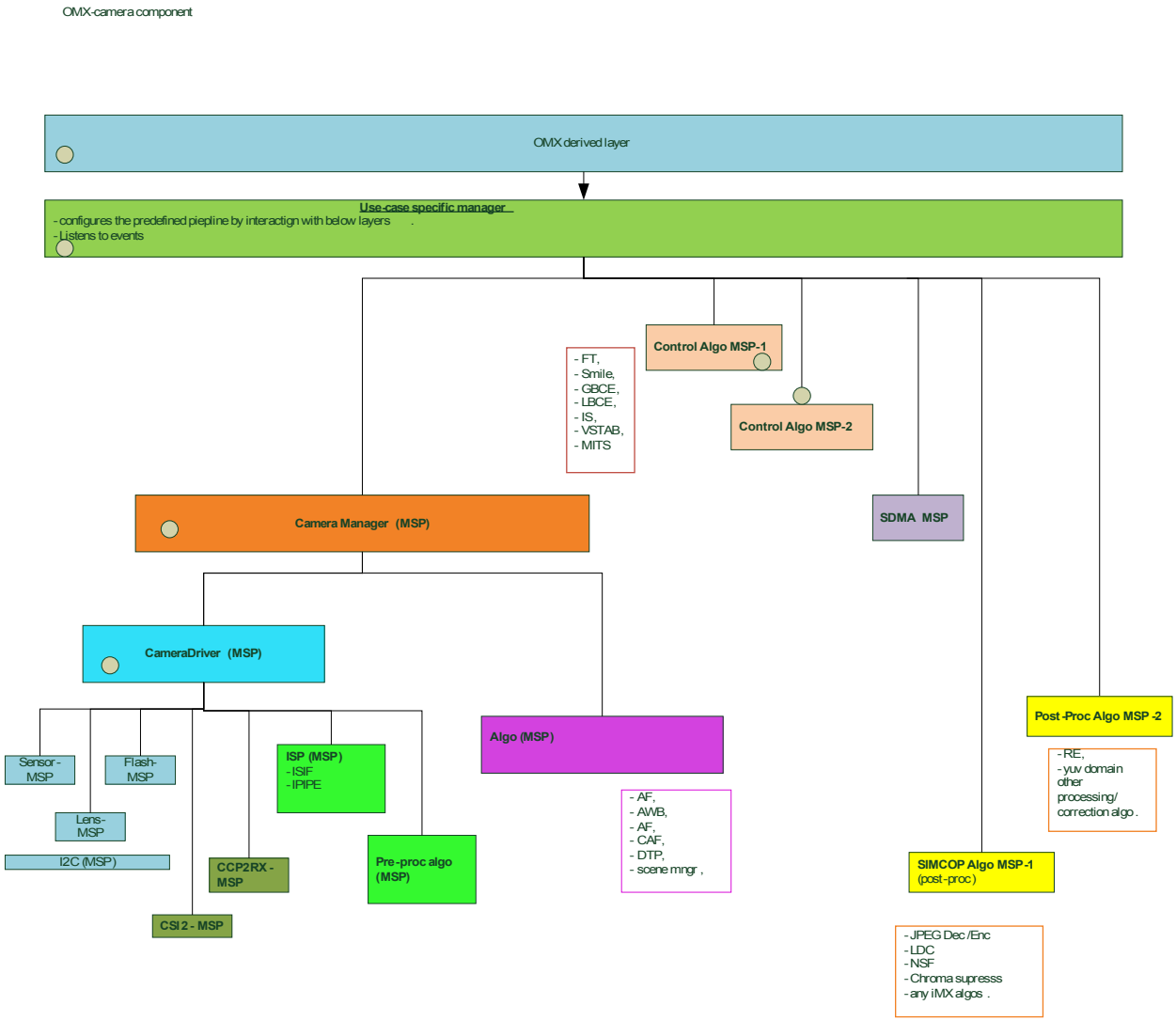


Figure 12 Camera internal architecture

Since there is specific details and lot of hidden items related to interconnections are there, Camera manager abstracts Camera Driver, ISP processing and even ISP specific feed back Algos. So, this Camera Manager acts as a s logical driver. There is no need to standardize the MSP parameter interface. This layer would be using the MSP interface. since, lot of setting would be very specific, at this moment won't be able to standardize the parameter. What comes out of this is used for post-processing as well as for control Algorithm kind of processing(like, Smile detect, Face detect etc). Typically those are the place where lot of 3rd party algorithms would come in.

Post-processing and control algorithms, again, buffer input buffer output kind of aspect could be standardized, but at the same time, there could be some control variations. So, what we restrict would be to stick on to MSP interface and what API to call. For 3rd party integration, anyway the code modification would be needed at the place you want to insert the algorithm. But the function calls would be based on MSP.

Note: For more information on MSP interface refer to \WTSD_DucatiMMSW\framework\msp.

10 Interface Header Files

10.1 OpenMAX interface

The OpenMAX Integration Layer API is defined in a set of header files, namely:

- **omx_types.h**: Data types used in the OpenMAX IL
- **omx_core.h**: OpenMAX IL core API
- **omx_component.h**: OpenMAX IL component API
- **omx_audio.h**: OpenMAX IL audio domain data structures
- **omx _ivcommon.h**: OpenMAX IL structures common to image and video domains
- **OMX_ti_ivcommon**.h : Extended OpenMAX IL structures common to image and video domains
- **omx _video.h**: OpenMAX IL video domain data structures
- **omx _image.h**: OpenMAX IL image domain data structures
- **omx _other.h**: OpenMAX IL other domain data structures (includes A/V synchronization)
- **omx _index.h**: Index of all OpenMAX IL-defined data structures
- **Omx_ti_index.h :** Extended of all Extended Data structure.
- **omx _contentpipe.h**: Content pipe definition