OMAP™ SS&P DESIGN SPECIFICATION

OpenMAX™ 1.0 Nucleus® TMS2300 IMG Component

Document Revision: 1.1

Issue Date: 31 January 2006

Making**Wireless**

Table of Contents

List of Figures

List of Tables

Revision History

REV	DATE	AUTHOR	NOTES
1.0	28 January 2006	Narendran M R	Initial release
1.1	31 January 31, 2006	Anandhi Ramesh	Updated section 3.3 and 4.3

Approvals

REV	APPROVAL 1	DATE	APPROVAL 2	DATE
1.0	S Prabhavathy			

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

This document describes the design of OpenMAX™ 1.0 IMG Component. The environment for OpenMAX™ 1.0 IMG Component is:

- Nucleus® on OMAPV1030.
- OpenMAX™ 1.0 IMG Component.
- Revision 2.x or later I-Sample board.

1.1 Purpose

This document details the design specifications for OpenMAX™ 1.0 Nucleus® IMG Component on OMAPV1030.

1.2 Scope

This document addresses only design specifications.

Additional technical data can be found by referring to the OMAP™SS&P Technical Perspective and Data Package document.

The document provides information about technical data artifacts, including their title, standard ClearCase® VOB location, a brief description and the System or Software Checkpoint where the artifact is first introduced into the development process.

1.3 File Path

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

```
\\OMAPSW_SysDev\OMAPV1030\Multimedia\docs\
```

1.4 File Name

The file name of this document is CSSD_DesignSpec_Locosto_OMX_Img.doc

1.5 References

All References can be found on the Cellular Systems web site or the World Wide Process and Tools Group web site.

1.6 Definitions

Terms used in this document can be found in the Cellular Systems Glossary Document.

Terms that are introduced in this document are detailed below:

Table 1 Terms and Acronyms

ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machines
GPP	General Purpose Processor
JPEG	ISO 11172-3 Image compression standard
OMAP™	Open Multimedia Application Platform
OMX	OMX and OpenMAX™ 1.0 are used interchangeably in the document.
OS	Operating System
OSAL	Operating System Adaptation Layer.

2 Architectural Overview

OpenMAX™ 1.0 IMG Component facilitates the interaction between an Application and the Emuzed Imaging Codec. Communication between OpenMAX™ 1.0 IMG Component and Emuzed Imaging Codec happens via non-blocking message interface between the OMX client (the application which calls OMX functions) and IMG core component which runs as a separate thread in the system.

OpenMAX™ 1.0 IMG Component is a type of OpenMAX™ 1.0 component. It conforms to the guidelines defined by OpenMAX™ 1.0 specifications and implements a standard set of functions. An Application will access the OpenMAX™ 1.0 IMG Component's interfaces to send commands to the underlying Emuzed Imaging Codec to perform operations such as initialize, start, and stop of different imaging operations. In this usage, OpenMAX™ 1.0 IMG Component is said to wrap the underlying Emuzed Imaging Codec.

To offer services to an application, OpenMAX™ 1.0 IMG Component works in unison with the OpenMAX™ 1.0 core. For details on OpenMAX™ 1.0 core, please refer to OpenMaxIL1.0 Specification.

2.1 Features

The table1 lists the high-level features that OpenMAX™ 1.0 IMG Component. A given instance of this OMX client can support only one feature. i.e. If application needs a JPEG encode feature and a color conversion feature, then, two instance of IMG Component needs to be created – one instance configured for JPEG encoding and the other instance configured for color conversion. Details on initialization, configuration and data flow for each of these features are explained later in the document.

Table 2 Feature List

Feature	Desciption
JPEG encode	Encode raw input images to baseline jpeg format conforming to ISO/IEC 11172-3 standard
JPEG decode	Decode JPEG images to raw YUV or RGB formats
Color conversion	Color convert images from different YUV formats to RGB formats
Rescale	Rescale RGB or YUV images using bilinear interpolation
Overlay	Overlay two RGB images depending on color key provided
Alpha blending	Alpha blending of two RGB images depending on the transparency color key provided.
Rotation	Rotate an image by 90, 180, 270 degress
Grayscale Effect	Convert color images to gray scale
Sepia Effect	Add sepia tone to an image to give antique look

2.2 System Design

The IMG module is divided into two separate components

1) IMG Core.

2) IMG OMX client

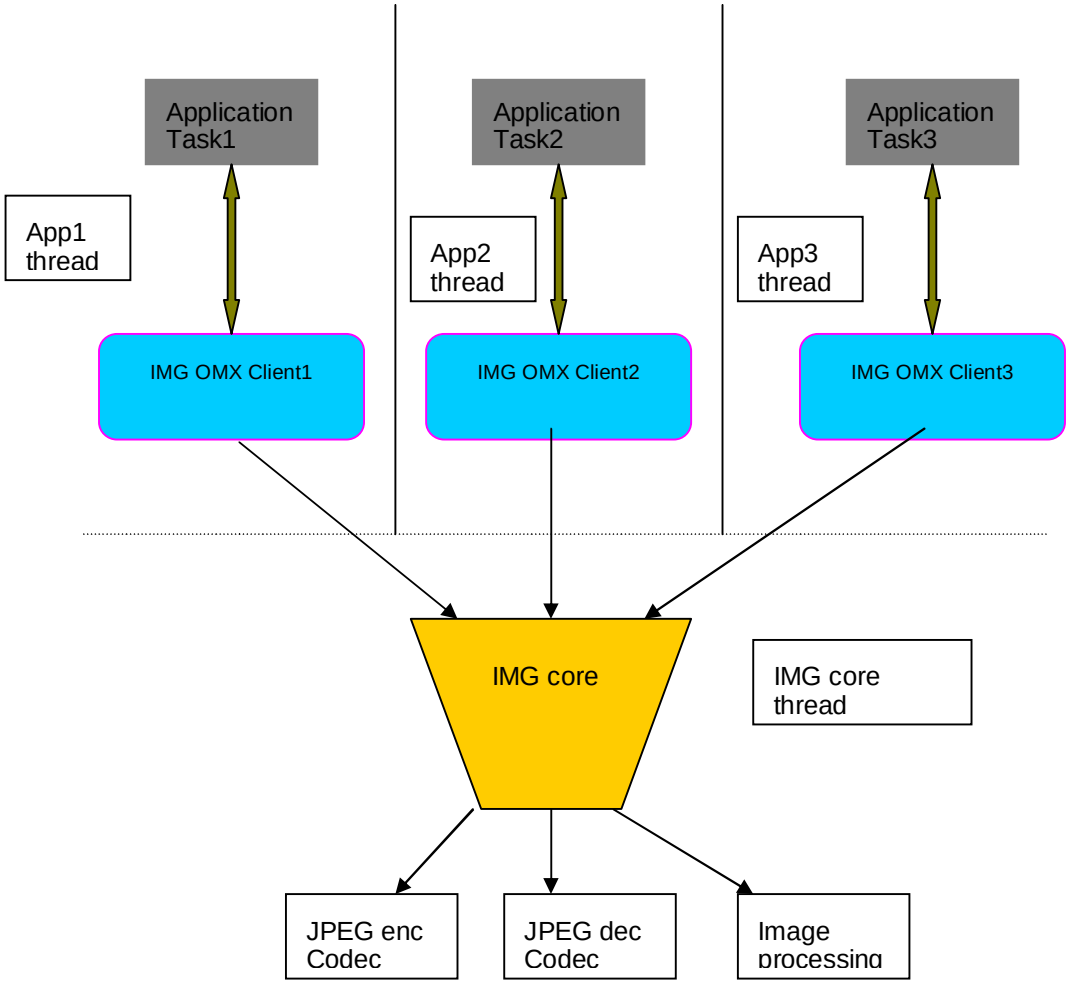
IMG core is a separate thread running in the system and facilitates actual image processing of data. IMG core is started when the system boots up and it waits for commands from different IMG OMX clients. There is only one instance of this thread running in the system and it uses the Emuzed Imaging Codecs to do the actual data processing. IMG core interface is not exposed to application space and only IMG OMX clients interface with IMG core.

IMG OMX client is the interface which enables applications to facilitate different image processing features. This client exposes standard OpenMAX™ 1.0 complaint API interface. The client runs in the context (same thread) as that of application which invokes these APIs. An application can invoke multiple IMG OMX clients as per the requirement. All these clients will interface with IMG Core to carry out the actual data processing. But this client to Core communication is hidden from the application and the application just needs to invoke the OMX client APIs to carry out the functionality.

Figure 1 shows the architecture of OpenMAX™ 1.0 IMG Component in context of a complete system. Since the actual data processing happens in a separate IMG core thread, the OMX client data processing calls are non-blocking. This diagram illustrates how communication will happen between an application, the OpenMAX™ 1.0 Core, the OpenMAX™ 1.0 IMG Component and the Emuzed Imaging Codec.

The component instances are classified according to the operations they perform as encoder, decoder and image processing component. The encoding and decoding components cannot be used for any other image processing operations when they are in the executing state. But the image processing component can be used interchangeably for rotation, overlay, color conversion, rescaling, or image effects.

Figure 1 IMG System Diagram



2.3 Design Rational

The IMG client is designed keeping the below mentioned design aspects in mind.

1. Low -latency or non-blocking processing call operation. This means that the application which invokes processing APIs should not be blocked and should immediately return. The actual processing should happen in a separate thread and inform the application regarding the process completion through a separate call back function. This is important from the perspective that the application thread should not be blocked and should be responsive to other user interaction while the processing is carried out.
2. Industry Standard interface. The OMX interface is industry accepted interface and by exposing an OMX 1.0 compliant interface, it becomes easier for application developers to integrate this component with rest of the system.
3. Minimum code footprint. Theoretically each of these encode, decode and image processing functions could be provided as a separate OMX client component. But in this implementation, only one OMX IMG client implementation is provided which can be configured for doing any of the above mentioned operations. The rational being each of these operation involve similar operations and the actual interface with the codecs happens only in the core level.

2.4 Limitations

TBD

3 Component Interface

This section describes the data types, data structures, callbacks and macros, which are required for an application interfacing with the IMG OMX component.

3.1 Include Files For Parent Application

An Application which uses OpenMAX™ 1.0 IMG OMX component must include the files listed in Table 4.

Table 3 Include Files for application

File	Function
OMX_Core.h	Contains prototypes of the core functions and definitions used by both the application and IMG Component to access common items.
OMX_Index.h	Contains the definitions of parameter and configuration indices for both application and IMG Component.
OMX_ImgComponent.h	This file contains the structures needed by Image components to exchange parameters and configuration data between the application and the IMG Component.
OMX_TIImage.h	This file contains TI defined extended interface types

3.2 Defined Types

3.2.1 Basic Data Types

Please refer to section 9.2.1 of OpenMaxIL1.0 Specification. OpenMAX™ 1.0 IMG Component uses these data types instead of fundamental C-types like int, char etc. in order to provide portability across different platforms, compilers and operating systems. These data types are defined in OMX_Types.h file.

3.2.2 OMX_TIIMG_COMPONENT

This is the name of component that needs to be passed to the OMX_getHandle core function. This function will obtain the handle of the OpenMAX™ 1.0 IMG Component. This string is defined in OMX_TIImage.h file.

3.3 Data Structures

3.3.1 OMX_TICOLOR_FORMATTYPE

This enum type defines the color formats which are passed as parameter to different configuration structures.

Table 4 OMX_TICOLOR_FORMATTYPE Structure

Enum Type	**Description or Evaluation**
OMX_TICOLOR_Monochrome	Black and White
OMX_TICOLOR_8bitRGB332	Red 7:5, Green 4:2, Blue 1:0
OMX_TICOLOR_12bitRGB444	Red 11:8, Green 7:4, Blue 3:0
OMX_TICOLOR_16bitARGB4444	Padding 15:12, Red 11:8, Green 7:4, Blue 3:0
OMX_TICOLOR_ARGB1555	Padding 15, Red 14:10, Green 9:5, Blue 4:0

OMX_TICOLOR_RGB565	Red 15:11, Green 10:5, Blue 4:0
OMX_TICOLOR_BGR565	Blue 11:8, Green 7:4, Red 3:0
OMX_TICOLOR_RGB666	Red 17:12, Green 11:6, Blue 5:0
OMX_TICOLOR_ARGB1665	Padding 17 Red 16:11, Green 10:5, Blue 4:0
OMX_TICOLOR_ARGB1666	Padding 18 Red 17:12, Green 11:6, Blue 5:0
OMX_TICOLOR_RGB888	Red 23:16, Green 15:8, Blue 7:0
OMX_TICOLOR_BGR888	Blue 15:11, Green 10:5, Red 4:0
OMX_TICOLOR_ARGB1887	Padding 23 Red 22:15, Green 14:7, Blue 6:0
OMX_TICOLOR_ARGB1888	Padding 24 Red 23:16, Green 15:8, Blue 7:0
OMX_TICOLOR_BGRA8888	Padding 31:24, Blue 23:16, Green 15:8, Red 7:0
OMX_TICOLOR_ARGB8888	Padding 31:24, Red 23:16, Green 15:8, Blue 7:0
OMX_TICOLOR_YUV411Planar	
OMX_TICOLOR_YUV411PackedPlanar	
OMX_TICOLOR_YUV420Planar	
OMX_TICOLOR_YUV420PackedPlanar	
OMX_TICOLOR_YUV420SemiPlanar	
OMX_TICOLOR_YUV422Planar	
OMX_TICOLOR_YUV422PackedPlanar	
OMX_TICOLOR_YUV422SemiPlanar	
OMX_TICOLOR_YCbYCr	
OMX_TICOLOR_YCrYCb	
OMX_TICOLOR_CbYCrY	
OMX_TICOLOR_CrYCbY	
OMX_TICOLOR_YUV444Interleaved	
OMX_TICOLOR_RawBayer8bit	
OMX_TICOLOR_RawBayer10bit	
OMX_COLOR_FormatRawBayer8bitcompressed	
OMX_TICOLOR_RGB444	Red 11:8, Green 7:4, Blue 3:0
OMX_TICOLOR_RGB565	Red 15:11, Green 10:5, Blue 4:0
OMX_TICOLOR_BGR565	Blue 15:11, Green 10:5, Red 4:0
OMX_TICOLOR_RGB666	Red 17:12, Green 11:6, Blue 5:0
OMX_TICOLOR_RGB888	Red 24:16, Green 15:8, Blue 7:0
OMX_TICOLOR_ARGB444	Extended RGB 444.
OMX_TICOLOR_YUV444	Y, U, V sampled at every pixel.
OMX_TICOLOR_YUV422H	Y sample at every pixel, U and V sampled at every second pixel horizontally on each line

OMX_TICOLOR_YUV422V	Y sample at every pixel, U and V sampled at every second pixel vertically on each line
OMX_TICOLOR_YUV422I	Y sample at every pixel, U and V sampled at every second pixel and interleaved(YUYV)
OMX_TICOLOR_YUV420P	8 bit Y plane followed by 8 bit 2x2 sub-sampled U and V planes.
OMX_TICOLOR_MONOCHROME	Black and White

3.3.2 OMX_TIIMAGE_EFFECTTYPE

This enum type defines the effects type. This passed as a configuration parameter to the effects API.

Table 5 OMX_TIIMAGE_EFFECTTYPE Structure

Enum Type	Description or Evaluation
OMX_TIIMAGE_SEPIAEFFECT	Adds a sepia tone to the image to give an impression of aged film
OMX_TIIMAGE_GRAYEFFECT	Converts the colour image to a gray tone.

3.3.3 OMX_TIIMAGE_OVERLAYTYPE

This enum type defines the effects type. This passed as a configuration parameter to the overlay API.

Table 6 OMX_TIIMAGE_OVERLAYTYPE Structure

Enum Type	Description or Evaluation
OMX_TIIMAGE_OVERLAY	Overlays one image on another without any blending, based on the colour key. All areas of the same colour as the colour key are replaced with the overlay image
OMX_TIIMAGE_ALPHABLEND	Blends two images with the formula alpha * p1 + (1-alpha) * p2 where p1 and p2 are the pixel values at a location for the two images.

3.3.4 OMX_TIIMAGE_WINDOWTYPE

This structure defines the window type. This passed as a configuration parameter to the decode and rescale API.

Table 7 OMX_TIIMAGE_WINDOWTYPE Structure

Enum Type	Description or Evaluation
nXOffset	X Coordinate offset for the portion of the image to be windowed.
nYOffset	Y Coordinate offset for the portion of the image to be windowed.
nWidth	Width of the window.
nHeight	Height of the window.

3.3.5 OMX_TIIMAGE_ROTATETYPE

This enum type defines the rotation type. This passed as a configuration parameter to the rotation API.

Table 8 OMX_TIIMAGE_ROTATETYPE Structure

Enum Type	Description or Evaluation
OMX_TIIMAGE_ROTATE0	Rotate by 0 deg.
OMX_TIIMAGE_ROTATE90	Rotate by 90 deg.
OMX_TIIMAGE_ROTATE180	Rotate by 180 deg.
OMX_TIIMAGE_ROTATE270	Rotate by 270 deg.

3.3.6 OMX_TIIMAGE_ENCODE_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetParameter function call. This param type is sent along with the OMX_IndexParamEncode index for set/get param call as well as the OMX_IndexConfigEncode index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for JPEG encoding. The OMX_TIIMAGE_ENCODE_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 9 OMX_TIIMAGE_ENCODE_PARAMTYPE Structure

Data field Name	Description or Evaluation
tImageCodingType	This is an OMX_IMAGE_CODINGTYPE enum type. The only supported format in this version is OMX_IMAGE_CodingJPEG
nQualityFactor	Quality value varies from 1-100. 1 represents best compression, but worst quality and 100 represents best quality, but worst compression.
nImageWidth	Width of the image to be encoded
nImageHeight	Height of the image to be encoded.
nStride	Number of bytes per span of an image. Imagewidth could be smaller than nStride
tInputImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the input image.
tImageEncodeFormat	This OMX_TICOLOR_FORMATTYPE enum type defines the format of the actual encoding.
nEncodeAUSize	If this value is set to 0, encoding happens in one shot. The output buffer provided should be sufficient enough to hold the full encoded image. Else encoder will encode only the specified access units in one process call.
bInsertHeader	IF this flag is set to OMX_TRUE, header will be inserted to the generated bitstream, else not.

3.3.7 OMX_TIIMAGE_DECODE_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetConfig function call. This param type is sent along with the OMX_IndexParamDecode index for set/get param call as well as the OMX_IndexConfigDecode index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for JPEG decoding. The OMX_TIIMAGE_DECODE_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 10 OMX_TIIMAGE_DECODE_CONFIGTYPE Structure

Data field Name	Description or Evaluation
nStride	This is the width of the buffer provided to decoder. This value could be bigger than image width.
nImageScalingFactor	The decoder has capability to scale down the image during the decode process. The value is used to specify the required scale value.
nDecodeAUSize	If this value is set to 0, decoding happens in one shot. The input buffer provided should contain the full encoded image. Else decoder will decode only the specified access units in one process call.
tCropWindow	Window specifying the portion of the image to be decoded.
tImageOutPutFormat	This OMX_TICOLOR_FORMATTYPE enum type defines the format of output decoded data expected.

3.3.8 OMX_TIIMAGE_ROTATE_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetParameter function call. This param type is sent along with the OMX_IndexParamRotate index for set/get param call as well as the OMX_IndexConfigRotate index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for image rotation. The OMX_TIIMAGE_ROTATE_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 11 OMX_TIIMAGE_ROTATE_PARAMTYPE Structure

Data field Name	Description or Evaluation
nStride	Buffer width.
nImageWidth	Width of the image
nImageHeight	Height of the image
tInputImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the input image.
tRotate	This OMX_TIIMAGE_ROTATETYPE enum type field indicates the rotation value.

nRotateManual	If tRotate is not populated, this field contains the rotation degree.
bInplace	IF this flag is set to true, operation happens in place. No separate buffer needs to be supplied for output buffer.

3.3.9 OMX_TIIMAGE_RESCALE_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetParameter function call. This param type is sent along with the OMX_IndexParamRescale index for set/get param call as well as the OMX_IndexConfigRescale index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for image rotation. The OMX_TIIMAGE_RESCALE_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 12 OMX_TIIMAGE_RESCALE_PARAMTYPE Structure

Data field Name	**Description or Evaluation**
nInputImageWidth	Width of the input image
nInputImageHeight	Height of the input image
tInputImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the input image.
nOutputImageWidth	Width of the output image
nOutputImageHeight	Height of the output image
nStride	Buffer width.
tCropWindow	OMX_TIIMAGE_WINDOWTYPE field specifying the portion of the image to be zoomed
nZoomFactor	Zoom Factor value.
nAutoZoom	If this flag is true the component performs auto zoom.
nStartZoomFactor	In auto zoom mode, the starting zoom facto value.
nFinalZoomFactor	In auto zoom mode, the Final zoom facto value.
nIncrement	In auto zoom mode, the zoom factor increment value.
bZoomReset	If this flag is set to true, the zoom factor is reset every time it reaches the final zoom factor in auto reset mode.
bInplace	IF this flag is set to true, operation happens in place. No separate buffer needs to be supplied for output buffer.

3.3.10 OMX_TIIMAGE_EFFECT_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetParameter function call. This param type is sent along with the OMX_IndexParamEffect index for set/get param call as well as the OMX_IndexConfigEffect index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for image effects (gray scale, sepia etc). The OMX_TIIMAGE_EFFECT_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 13 OMX_TIIMAGE_EFFECT_PARAMTYPE Structure

Data field Name	Description or Evaluation
nImageWidth	Width of the image
nImageHeight	Height of the image
tInputImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the input image.
tEffect	This OMX_ TIIMAGE_EFFECTTYPE enum type defines the effect.
nStride	Buffer Width
bInplace	IF this flag is set to true, operation happens in place. No separate buffer needs to be supplied for output buffer.

3.3.11 OMX_TIIMAGE_OVERLAY_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetParameter function call. This param type is sent along with the OMX_IndexParamOverlay index for set/get param call as well as the OMX_IndexConfigOverlay index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for image overlay. The OMX_TIIMAGE_OVERLAY_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 14 OMX_TIIMAGE_OVERLAY_PARAMTYPE Structure

Data field Name	Description or Evaluation
nSrcImageWidth	Width of the image to be overlayed
nSrcImageHeight	Height of the image to be overlayed.
tSrcImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the input source mage.
nOverlayImageWidth	Width of the overlay image
nOverlayImageHeight	Height of the overlay image
tOverlayImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the overlay image.
nSrcXOffset	X- coordinate offset for overlay

nSrcYOffset	Y-coordinate offset for overlay
tOverlay	This OMX_TIIMAGE_OVERLAYTYPE enum type field indicates the choice of alpha blending or color key substitution.
tTransmissivity	Transmissivity Value for alpha blending
nAlpha	Alpha Value if the type is manual in tTransmissivity
nOverlay	Index value for transparency
nStride	Buffer Width
bInplace	If this flag is set to TRUE, the output would be overwritten to source image.

3.3.12 OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE

This parameter is passed to the IMG client using the standard OMX SetParameter function call. This param type is sent along with the OMX_IndexParamColorConversion index for set/get param call as well as the OMX_IndexConfigColorConversion index for set/getconfig call. (defined in OMX_TIImage.h). This parameter needs to be set if the OMX IMG client needs to be configured for color conversion of images. The OMX_TIIMAGE_COLORCONVERSION_DEFAULTS structure is passed for the initialization of the component. Once the component is past the loaded state, the parameters are changed through a OMX_SetConfig call. The structure members are detailed in the table below.

Table 15 OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE Structure

Data field Name	**Description or Evaluation**
nImageWidth	Width of the image
nImageHeight	Height of the image
nSrcXOffset	X- coordinate offset for the input image
nSrcYOffset	Y-coordinate offset for the input image
tInputImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the input image.
tOutputImageFormat	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the output image.
nStride	Buffer Width
bInplace	IF this flag is set to true, operation happens in place. No separate buffer needs to be supplied for output buffer.

3.3.13 OMX_TIIMAGE_ENCODE_IMAGEINFOTYPE

This parameter is passed to the IMG client using the standard OMX GetConfig function call. The index value for this param type is OMX_IndexConfigEncodeImgInfo (defined in OMX_TIImage.h). This parameter is used by the application to allocate the output buffer for the encoding operation. The structure members are detailed in the table below.

Table 16 OMX_TIIMAGE_ENCODE_IMAGEINFOTYPE Structure

Data field Name	Description or Evaluation
nOutputSize	Size of the output buffer

3.3.14 OMX_TIIMAGE_DECODE_IMAGEINFOTYPE

This parameter is passed to the IMG client using a call for getConfig with the OMX_IndexConfigDecodeImgInfo with this structure as the third parameter. This is used for allocating the output buffer for a decoding operation. The structure members are detailed in the table below.

Table 17 OMX_TIIMAGE_DECODE_IMAGEINFOTYPE Structure

Data field Name	Description or Evaluation
nWidth	Extended Width of the output image
nHeight	Extended Height of the output image
nActWidth	Actual Width of the output image
nActHeight	Actual Height of the output image
tImageFormat	This OMX_TIICOLOR_FORMATTYPE enum type field indicates the format of the decoded image.

3.4 API Requirements Coverage

The OpenMAX™ 1.0 core provides a set of macros that are used by the application to perform various operations like loading the component, communicating with IMG Component etc. These macros are defined in OMX_Core.h. Each macro maps to a function implemented by the OpenMAX™ 1.0 IMG Component. Detailed description of each function implemented by the IMG Component is given in following sub sections. The application must not call any of these functions directly and instead use the macros provided by the OMX core.

3.4.1 OMX_Img_ComponentInit

```
OMX_ERRORTYPE OMX_Img_ComponentInit(OMX_HANDLETYPE hComp)
```

Description

This function is called by OMX core to initialize the component. The core makes the call when the Application needs to call OMX_GetHandle and a new component must be instantiated.. The OMX core maintains a table that lists all the OMX components and their ComponentInit functions. The OMX_Img_ComponentInit function must be present in this component table.

Parameters

Name	Type	Description
hComp	OUT	The component fills the handle structure with pointers to functions that it implements. After this, the application can use these function pointers to access the functionality of this OMX component

Return

OMX_ErrorNone	This is returned if macro executes successfully.
OMX_ErrorBadParameter	This error is returned if one of the input parameters is wrong.

Requirement Coverage

This method addresses requirement:

Implementation

- Blocking function.
- Entry function to the OMX component.
- Populates hComp with function pointers to functions implemented by this particular OMX component.
- Allocates private data of the OMX component.
- Sets the current state of the IMG Component as OMX_StateLoaded.

3.4.2 OMX_Img_SetCallbacks

```
OMX_ERRORTYPE OMX_Img_SetCallbacks (OMX_HANDLETYPE hComp,
                                    OMX_CALLBACKTYPE* pCallBacks,
                                    OMX_PTR pAppData)
```

Description

This function is called by OMX core. The application needs to call OMX_GetHandle. After calling OMX_Img_ComponentInit, the OMX core calls this function to provide application callbacks to the OpenMAX™ 1.0 IMG Component.

Parameters

Name	Type	Description
hComp	IN	Handle of the component to be accessed. This is the component handle returned by the call to the GetHandle function
`pCallBacks`	IN	pointer to an OMX_CALLBACKTYPE structure used to provide the callback information to the component
`pAppData`	IN	pointer to an application defined value. It is anticipated that the application will pass a pointer to a data structure or a "this pointer" in this area to allow the callback (in the application) to determine the context of the call

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

Implementation

- Blocking function.
- Copies the contents of the callback structure into the private data area of the component for later use.

3.4.3 OMX_Img_GetComponentVersion

```
OMX_ERRORTYPE OMX_Img_GetComponentVersion (OMX_HANDLETYPE hComp,
                                          OMX_STRING* pComponentName,
                                          OMX_VERSIONTYPE* pComponentVersion,
                                          OMX_VERSIONTYPE* pSpecVersion,
                                          OMX_UUIDTYPE* pComponentUUID)
```

Description

Queries the component and returns information about the component. The application calls this function to get the version of the component.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pComponentName	OUT	Pointer to an area where the component writes the name of the component at the successful return from this macro. The maximum length of the name is 128 including the null terminating character.
pComponentVersion	OUT	This is a pointer to the OMX_VERSIONTYPE structure which is filled by the component. The component fills the component version information in this structure
pSpecVersion	OUT	This is a pointer to the OMX_VERSIONTYPE structure which is filled by the component. The component fills the OMX specification version information in this structure.
pComponentUUID	OUT	This is a pointer to the UUID structure which is be filled by the component. Currently, the UUID is not used and the component does not update this value.

Return

OMX_ErrorNone	This is returned if the macro executes successfully.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.

Requirement Coverage

This method addresses requirement:

Implementation

- Blocking function.
- OpenMAX™ 1.0 IMG Component fills component version, OMX specification version and UUID information in the structures passed by application.

3.4.4 OMX_Img_SendCommand

```
OMX_ERRORTYPE OMX_Img_SendCommand (OMX_HANDLETYPE hComp,
                                   OMX_COMMANDTYPE Cmd,
                                   OMX_U32 nParam)
```

Description

Sends a command to the component. Currently there is only one command that is recognized by the component, the command to change the state of the component.

Parameters

Name	Type	Description
hComp	IN	This input argument is component handle.
Cmd	IN	This specifies the command type/category. The value of this argument can be OMX_CommandStateSet. Currently in OpenMAX™ 1.0, the only valid command is to set the component state.
nParam	IN	The value of this argument is dependent on the cmd argument. If the value of the Cmd is: OMX_CommandStateSet: this argument contains the state that is to be set for the component. The value can be one of the values defined by OMX_STATETYPE.

Return

OMX_ErrorNone	This is returned if the macro executes successfully.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.
OMX_ErrorInvalidState	Indicates that the state specified by the argument nParam is invalid for the current state of the component i.e. the component cannot change to the state specified by the argument nParam.

Requirement Coverage

This method addresses requirement:

Implementation

- Non-blocking function.
- Calls ProcessFunction to handle input and output buffers.
- For successful state transitions, calls the EventHandler callback with a state change notification, calls EventHandler with an error othersise.

3.4.5 OMX_Img_GetParameter

```
OMX_ERRORTYPE OMX_Img_GetParameter (OMX_HANDLETYPE hComp,
                                    OMX_INDEXTYPE nParamIndex,
                                    OMX_PTR pCompParam)
```

Description

The application should call this function to get the currently in effect parameter settings of the component. The correct sequence used to call this function is for the application to allocate an empty structure, set the structure version and size information and then call this function with the structure pointer as the third parameter and the structure index as the second parameter. The component will fill in the values of the structure from the component's internal state information. This function may be used to get the initialization parameters of the component, when the component is in any state except the INVALID state.

Parameters

Name	Type	Description
`hComp`	IN	This input argument is the component handle.
`nParamIndex`	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
`pCompParam`	OUT	This is a pointer to the structure that needs to be filled in by the component.

Return

OMX_ErrorNone	This value is returned when the component gives the required information.
OMX_ErrorBadParameter	This is returned if any arguments are invalid.

Pre Condition

The structure specified by the third argument must have the structure size and version information filled in before invoking the function.

Requirement Coverage

This method addresses requirement:

Implementation

- Blocking function.
- Copies the appropriate values from the component's private data area and populates the structure passed in the third argument.
- Performs basic parameter checking by comparing the size passed in the structure (third argument to this function) to the actual size of the structure.

Valid Parameters

OMX_IndexParamEncode, OMX_TIIMAGE_ENCODE_PARAMTYPE
OMX_IndexParamEncodeDefault (output is OMX_TIIMAGE_ENCODE_DEFAULT)
OMX_IndexParamDecode, OMX_TIIMAGE_DECODE_PARAMTYPE
OMX_IndexParamDecodeDefault , (output is OMX_TIIMAGE_DECODE_DEFAULT)
OMX_IndexParamRotate, OMX_TIIMAGE_ROTATE_PARAMTYPE
OMX_IndexParamRotateDefault , (output is OMX_TIIMAGE_ROTATE_DEFAULT)
OMX_IndexParamRescale, OMX_TIIMAGE_RESCALE_PARAMTYPE
OMX_IndexParamRescaleDefault , (output is OMX_TIIMAGE_RESCALE_DEFAULT)
OMX_IndexParamEffect, OMX_TIIMAGE_EFFECT_PARAMTYPE

OMX_IndexParamEffectDefault , (output is OMX_TIIMAGE_EFFECT_DEFAULT)

OMX_IndexParamOverlay, OMX_TIIMAGE_OVERLAY_PARAMTYPE

OMX_IndexParamOverlayDefault , (output is OMX_TIIMAGE_OVERLAY_DEFAULT)

OMX_IndexParamColorConversion, OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE

OMX_IndexParamColorConversionDefault ,. (output is OMX_TIIMAGE_COLORCONVERSION_DEFAULT)

These structures are defined OMX_TIImage.h.

OMX_Img_SetParameter

```
OMX_ERRORTYPE OMX_Img_SetParameter (OMX_HANDLETYPE hComp,
                                    OMX_INDEXTYPE nParamIndex,
                                    OMX_PTR pCompParam)
```

Description

The application should call this function to set the parameters of the component. The correct sequence to call this function is for the application to allocate and initialize a structure to be sent to the component and then call this function with the structure as the third parameter and the structure index as the second parameter. The component will make a local copy of this structure and uses the stored data at the time of component initialization. This function should be used to set the initialization parameters of the component, when the component is in the LOADED state.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
nParamIndex	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
pCompParam	IN	This input argument is a pointer to a structure which the component uses to make its local copy.

Return

OMX_ErrorNone	This is returned when component gives the required information.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.

Pre Condition

The structure specified by the third argument must have its structure size and version information filled in before invoking the macro.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with Texas Instruments OpenMAX TI 1.5 specification.

Implementation

- Blocking function.
- Copies the structure passed into the component's private data area for later use.
- Performs basic parameter checking by comparing the size passed in the structure (third argument to this function) to the actual size of the structure.

Valid Parameters

OMX_IndexParamEncode, OMX_TIIMAGE_ENCODE_PARAMTYPE
OMX_IndexParamDecode, OMX_TIIMAGE_DECODE_PARAMTYPE
OMX_IndexParamRotate, OMX_TIIMAGE_ROTATE_PARAMTYPE
OMX_IndexParamRescale, OMX_TIIMAGE_RESCALE_PARAMTYPE
OMX_IndexParamEffect, OMX_TIIMAGE_EFFECT_PARAMTYPE
OMX_IndexParamOverlay, OMX_TIIMAGE_OVERLAY_PARAMTYPE
OMX_IndexParamColorConversion, OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE

These structures are defined OMX_TIImage.h.

3.4.6 OMX_ Img_GetConfig

```
OMX_ERRORTYPE OMX_Img_GetConfig (OMX_HANDLETYPE hComp,
                                 OMX_INDEXTYPE nConfigIndex,
                                 OMX_PTR ComponentConfigStructure)
```

Description

This API enables the application to query the configuration parameters at run time. This function can be invoked at any time after the component has been loaded. The application allocates the required structure and passes it to this function. The component fills this structure with the required information.

Parameters

Name	Type	Description
`hComp`	IN	This input argument is the component handle.
`nConfigIndex`	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
`ComponentConfigStructure`	OUT	This output argument is the pointer to the structure to be filled by the component.

Supported nParamIndex values and their corresponding structures are:

- There are no supported configuration structures for the IMG Component.

Return

OMX_ErrorNone	This is returned when the component gives the required information.
OMX_ErrorBadParameter	This is returned when the one of the arguments is invalid.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with Texas Instruments OpenMAX TI 1.5 specification.

Implementation

Always OMX_ErrorBadParameter is returned as IMG Component not supports any configuration structures.

Valid Parameters

OMX_IndexConfigEncode, OMX_TIIMAGE_ENCODE_PARAMTYPE
OMX_IndexConfigDecode, OMX_TIIMAGE_DECODE_PARAMTYPE
OMX_IndexConfigRotate, OMX_TIIMAGE_ROTATE_PARAMTYPE
OMX_IndexConfigRescale, OMX_TIIMAGE_RESCALE_PARAMTYPE
OMX_IndexConfigEffect, OMX_TIIMAGE_EFFECT_PARAMTYPE
OMX_IndexConfigOverlay, OMX_TIIMAGE_OVERLAY_PARAMTYPE
OMX_IndexConfigColorConversion, OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE
OMX_IndexConfigEncodeImgInfo, OMX_TIIMAGE_ENCODE_IMAGEINFOTYPE
OMX_IndexConfigDecodeImgInfo, OMX_TIIMAGE_DECODE_IMAGEINFOTYPE

These structures are defined OMX_TIImage.h.

3.4.7 OMX_Img_SetConfig

```
OMX_ERRORTYPE OMX_Img_SetConfig (OMX_HANDLETYPE hComp,
                                 OMX_INDEXTYPE nConfigIndex,
                                 OMX_PTR ComponentConfigStructure)
```

Description

This API enables the application to change the configuration at run time. This function can be invoked at any time after the component has been loaded. The application should allocate memory for the correct structure, fill it with the required values and pass it to this function. The component makes a local copy of this structure and uses it to configure the codec at the appropriate moment.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
nConfigIndex	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
ComponentConfigStructure	OUT	This input argument is a pointer to a structure that holds the values with which codec is to be configured

Supported nParamIndex values and their corresponding structures are:

- There are no supported configuration structures for the IMG Component.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

Implementation

Always OMX_ErrorBadParameter is returned as IMG Component not supports any configuration structures.

Valid Parameters

OMX_IndexConfigEncode, OMX_TIIMAGE_ENCODE_PARAMTYPE
OMX_IndexConfigDecode, OMX_TIIMAGE_DECODE_PARAMTYPE
OMX_IndexConfigRotate, OMX_TIIMAGE_ROTATE_PARAMTYPE
OMX_IndexConfigRescale, OMX_TIIMAGE_RESCALE_PARAMTYPE
OMX_IndexConfigEffect, OMX_TIIMAGE_EFFECT_PARAMTYPE
OMX_IndexConfigOverlay, OMX_TIIMAGE_OVERLAY_PARAMTYPE
OMX_IndexConfigColorConversion, OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE

These structures are defined OMX_TIImage.h.

3.4.8 OMX_Img_GetState

```
OMX_ERRORTYPE OMX_Img_GetState (OMX_HANDLETYPE hComp,
                                OMX_STATETYPE* pState)
```

Description

The application calls this function to get the current state of the component.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pState	OUT	This is the output argument, which points to the memory location where the component should store its current state. This argument should not be NULL.

Return

OMX_ErrorNone	This is returned when the component gives the required information.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.

Requirement Coverage

This method addresses requirement:

Implementation

- Blocking function
- Copies the current state value stored in the component's private data area into the structure passed to the function.
- During state transitions (e.g.when the component is in the process of going to the idle state) the state returned will be the old state until the requested state change is completed.

3.4.9 OMX_Img_EmptyThisBuffer

```
OMX_ERRORTYPE OMX_Img_EmptyThisBuffer (OMX_HANDLETYPE hComp,
                                       OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

The application calls this function to send a buffer filled with input data to the input port of the component. This function will write the buffer pointer into the input data pipe of the component and then call the component's ProcessFunction to complete the buffer processing.

Parameters

Name	Type	Description
HComp	IN	This input argument is the component handle.
Pbuffer	IN	This is a pointer to the buffer header whose buffer is to be emptied.

Return

OMX_ErrorNone	This is returned when the component gives the required information.
OMX_ErrorPortsNotCompatible	This is returned if the specified port index is not valid.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.

Requirement Coverage

This method addresses requirement:

Implementation

- Stores the buffer provided by the application into the input data pipe.
- Calls ProcessFunction, which is an internal function that processes the buffers stored in the input and output pipes.

3.4.10 OMX_Img_FillThisBuffer

```
OMX_ERRORTYPE OMX_Img_FillThisBuffer (OMX_HANDLETYPE hComp,
                                      OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

The application calls this function to send an empty buffer to the output port of the component. Before invoking this function, the application must have received the buffer with the FillBufferDone callback from the component for the case that the IMG Component allocated the input buffers. This function will write the buffer into the output data pipe of the component and then call the ProcessFunction to complete processing of the buffer.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pBuffer	OUT	This is a pointer to the buffer header whose buffer is to be filled.

Return

OMX_ErrorNone	This is returned when the component gives the required information.
OMX_ErrorPortsNotCompatible	This is returned if the specified port index is not valid.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid

Requirement Coverage

This method addresses requirement:

Implementation

- Stores the buffer provided by the application into the output data pipe.
- Calls ProcessFunction, which is an internal function that processes the buffers stored in the input and output pipes.

3.4.11 OMX_Img_ComponentTunnelRequest

```
OMX_ERRORTYPE OMX_Img_ComponentTunnelRequest (OMX_HANDLETYPE hComp,
                                              OMX_U32 nPortInput,
                                              OMX_HANDLETYPE hTunneledComp,
                                              OMX_U32 nTunneledPort,
                                              OMX_DIRTYPE eDir,
                                              OMX_CALLBACKTYPE* pCallbacks)
```

A call to this function returns with OMX_ErrorNotImplemented error code.

3.4.12 OMX_Img_ComponentDeInit

```
OMX_ERRORTYPE OMX_Img_ComponentDeInit(OMX_HANDLETYPE hComp)
```

Description

This function is called by OMX core when application calls OMX_FreeHandle.

Parameters

Name	Type	Description
hComp	OUT	Handle of the component to be accessed. This is the component handle returned by the call to the GetHandle function

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

Implementation

- Blocking function.
- Releases private data of the OMX component.

3.5 Application Callbacks

The OpenMAX™ 1.0 specification allows the application to provide three callbacks for buffer exchange and event handling. At loading time, the OpenMAX™ 1.0 IMG Component receives a structure containing pointers to the callback functions. The IMG Component makes a copy of this structure in the private data area. This section describes the callback functions.

3.5.1 EventHandler

```
void (*EventHandler)(
     OMX_HANDLETYPE hComp,
     OMX_PTR pAppData,
     OMX_EVENTTYPE eEvent,
     OMX_U32 Data,
     OMX_STRING cExtraInfo)
```

Description

The EventHandler method is used to notify the application when an event of interest occurs. This event may be change of state, an error occurred etc. In the OpenMAX™ 1.0 IMG Component, this callback is invoked from SendCommand and ProcessFunction.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pAppData	IN	Pointer to data which was defined by application when the component was loaded. Using this data, application identifies who invoked this callback.
eEvent	IN	One of the component events that are defined in OMX_EVENTTYPE enumeration. This can be state change, an error etc.
Data	IN	Used only if an error event occurs. Data will be OMX_ERRORTYPE.
cExtraInfo	IN	String which may carry some more explanation about the error. It is not always required for a component to use this argument.

Return

None

Requirement Coverage

This method addresses requirement:

Implementation

- This callback happens when application at the completion of a state change (e.g. in response to a call to the OMX_Img_SendCommand function) or when an error occurs.
- The application can use this information to update it's component status information. The application should not block or perform processing within this call.

3.5.2 EmptyBufferDone

```
void (*EmptyBufferDone)(
      OMX_HANDLETYPE hComp,
      OMX_PTR pAppData,
      OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

This is the callback function of the application that a component uses to return an empty input buffer for the application for use. There is always a callback EmptyBufferDone from the component for each OMX_EmptyThisBuffer call from the application. In the case where the component is required to allocate the buffers, all the buffers are initially sent to application by calling EmptyBufferDone for each buffer during the transition to Executing from Idle. The Application should fill these buffers and call EmptyThisBuffer.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pAppData	IN	Pointer to the data which was defined by the application when the component was loaded. Using this data, the application identifies who invoked this callback.
pbuffer	IN	Pointer to buffer header structure which contains pointer to emptied buffer, its size etc.

Return

None

Requirement Coverage

This method addresses requirement:

Implementation

- EmptyBufferDone is invoked after the buffer has been emptied and is ready for the application to refill.
- The application can use this information to update it's buffer status information. The applicaton should not block or perform processing within this call.

3.5.3 FillBufferDone

```
void (*FillBufferDone)(
     OMX_HANDLETYPE hComp,
     OMX_PTR pAppData,
     OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

This is the application callback function that the component uses to return a filled output buffer to application. There is always a callback FillBufferDone from the component for each call OMX_FillThisBuffer from the application. Unlike EmptyBufferDone, FillBufferDone will be invoked for the first time only after a data is available in the output buffer.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
PappData	IN	Pointer to data which was defined by the application when the component was loaded. Using this data, the application identifies who invoked this callback.
Pbuffer	IN	Pointer to the buffer header structure which contains a pointer to the filled buffer, its size etc

Return

None

Requirement Coverage

This method addresses requirement:

Implementation

- FillBufferDone is invoked when the buffer is filled with encoded data.
- OMX component sets the EOS flag in the last buffer header that is given to application.
- The application can use this information to update it's buffer status information. The applicaton should not block or perform processing within this call.

4 Control and Data Flow

4.1 IMG Component States

OpenMAX™ 1.0 IMG Component will exist in one of five states at any given time. Figure 2 represents the state diagram for the OpenMAX™ 1.0 IMG Component. The IMG Component states are controlled by application via the OMX_SendCommand macro. The OMX core does not maintain states for the component. The core is involved in only two state transitions; which are from INVALID state to LOADED state (using function OMX_GetHandle) and component unloading (using OMX_FreeHandle). The Application controls the remainder of all state transitions via OMX_SendCommand macro.

Figure 2 State Diagram of a IMG Component

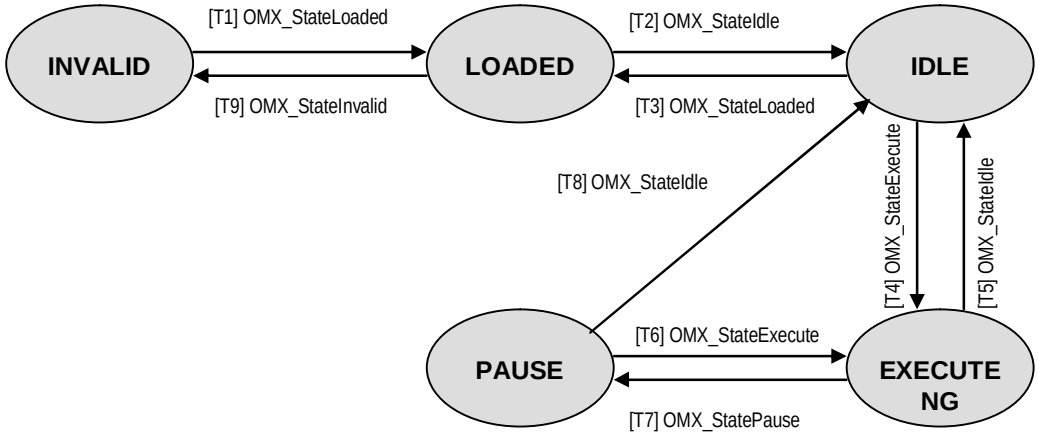


Table 18 shows state transitions for the OpenMAX™ 1.0 IMG Component and the functions/triggers that initiate these transitions.

Table 18 IMG Component State Transition

State	State change	Function/trigger
T1	Invalid->Loaded	OMX_GetHandle() Get the Handle to IMG Component Allocate resources needed for component execution
T2	Loaded->Idle	OMX_Img_SendCommand (stateIdle) Allocate buffer headers for input buffers Allocate buffer header for output buffer Allocate memory for input buffers Allocate memory for output buffer
T3	Idle->Loaded	OMX_Img_SendCommand (stateLoaded) Deallocate input buffers Deallocate output buffers Deallocate buffer headers for input buffers Deallocate buffer headers for output buffers
T4	Idle->Execute	OMX_Img_SendCommand (stateExecute) Call EmptyBufferDone for each input buffer if buffers are allocated by component. Queue all output buffers if allocated by component When input as well as output buffers are ready, the component will start processing data.
T5	Execute->Idle	OMX_Img_SendCommand (stateIdle) Reclaim all component allocated buffers and return all application allocated buffers.
T6	Pause->Execute	OMX_Img_SendCommand (stateExecute) 'Execute' State request from Application results in change of current state of IMG Component from pause to Execute.
T7	Executing->Pause	OMX_Img_SendCommand (statePause) 'Pause' State request from Application results only in change of current state of IMG Component to 'pause'. Data processing will be halted until it transitions to execute state.
T8	Pause->Idle	OMX_Img_SendCommand (stateIdle) Reclaim all component allocated buffers and return all application allocated buffers.
T9	Loaded->Invalid	OMX_FreeHandle Release (Deallocate) all resources allocated by the component

The above transitions in IMG Component state are illustrated below in Figure 3.

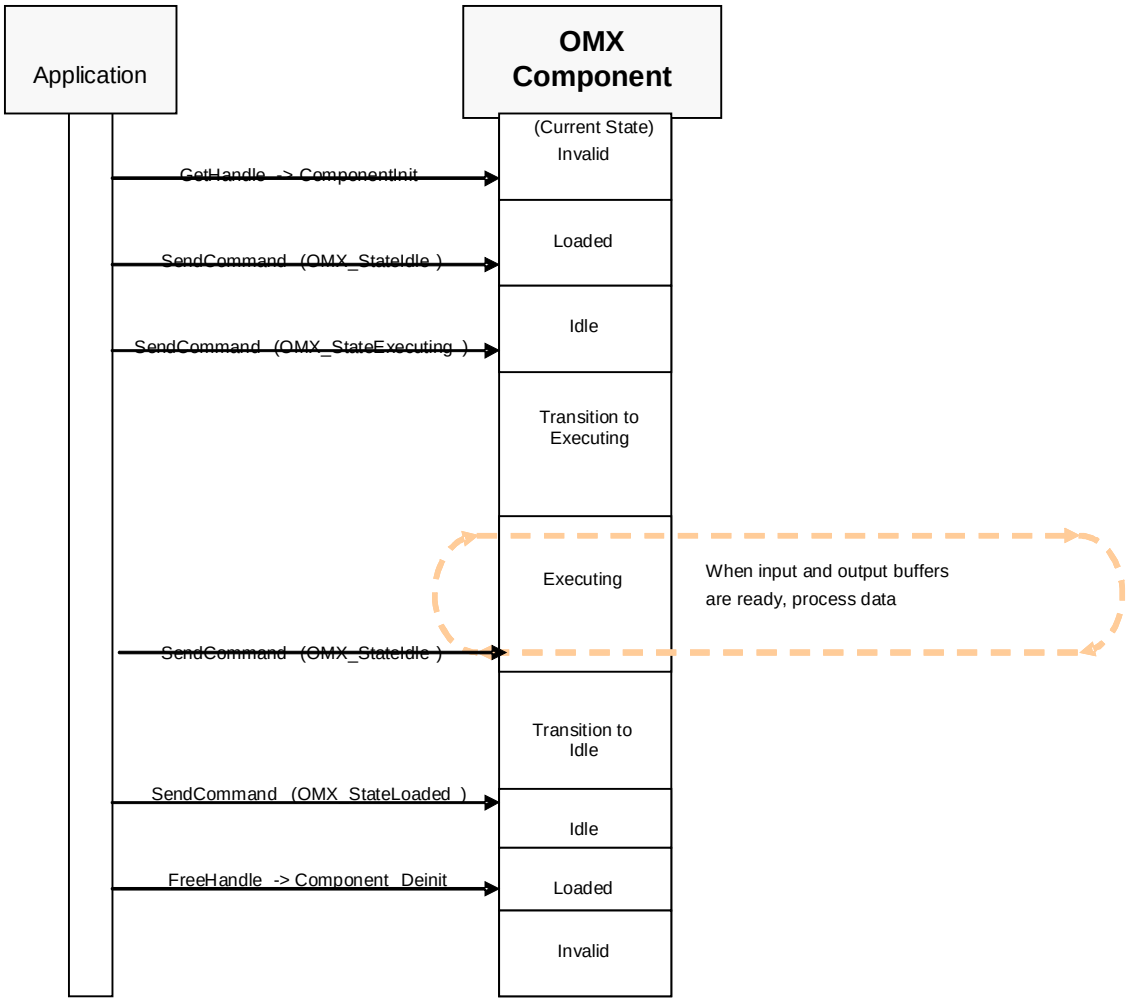


Figure 3 IMG Component State Transitions

4.2 OpenMAX™ 1.0 IMG Component Phases

There are various phases in the life cycle of OpenMAX™ 1.0 IMG Component. This section describes the control and data flow between application and OpenMAX™ 1.0 IMG Component using sequence diagrams. The OMX Component depicted in these sequence diagrams is the OpenMAX™ 1.0 IMG Component.

4.2.1 OpenMAX™ 1.0 IMG Component Load

Figure 4 shows the loading phase of OpenMAX™ 1.0 IMG Component's life cycle.

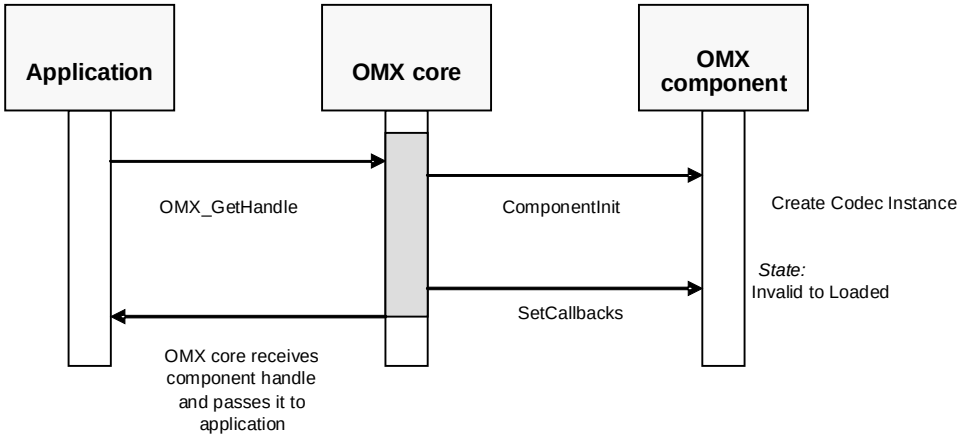


Figure 4 Loading of IMG Component

The following sequence is followed while loading OpenMAX™ 1.0 IMG Component:

- application calls function OMX_GetHandle of OMX core and supplies the name of OpenMAX™ 1.0 IMG Component and callbacks of application as arguments to this function.
- The OMX core searches for OpenMAX™ 1.0 IMG Component name in the component table. If the component entry is found, OMX core allocates memory for the component handle and calls the registered OMX_Img_ComponentInit function of the component.
- OMX_Img_ComponentInit will fill in the component handle with the function pointers of the exported functions and allocates necessary resources. It initializes itself to default values. Failing to do so will result in OMX component returning 'OMX_ErrorInsufficientResources' error to Application.
- OMX Core calls the function OMX_Img_SetCallbacks of the component and supplies application's callback function pointers to the component, which is recorded in the component's private data area.
- Application receives OpenMAX™ 1.0 IMG Component's handle structure with pointers to functions exported by the component.

4.2.2 OpenMAX™ 1.0 IMG Component Unload

Figure 5 shows OpenMAX™ 1.0 IMG Component during the unloading phase of its life cycle.

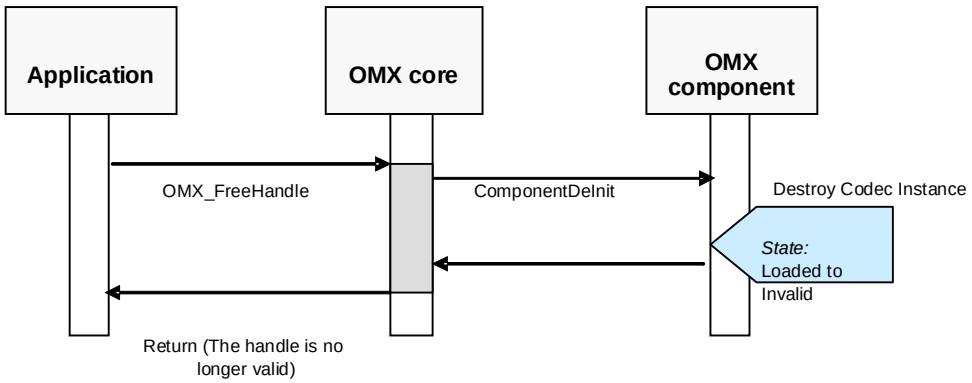


Figure 5 Unloading of IMG Component

The following sequence is followed while unloading OpenMAX™ 1.0 IMG Component:

- Application calls OMX core function OMX_FreeHandle to free the component handle, which was obtained by calling OMX_GetHandle at the time of loading OpenMAX™ 1.0 IMG Component.
- The Core method OMX_FreeHandle in turn calls the OMX_Img_ComponentDeInit function of the component. The OMX_Img_ComponentDeInit deallocates the resources that were allocated by the component.
- The OMX core then frees the memory used by OpenMAX™ 1.0 IMG Component handle.

4.2.3 OpenMAX™ 1.0 IMG Component Initialization

0 Shows OpenMAX™ 1.0 IMG Component during the initialization phase of its life cycle.

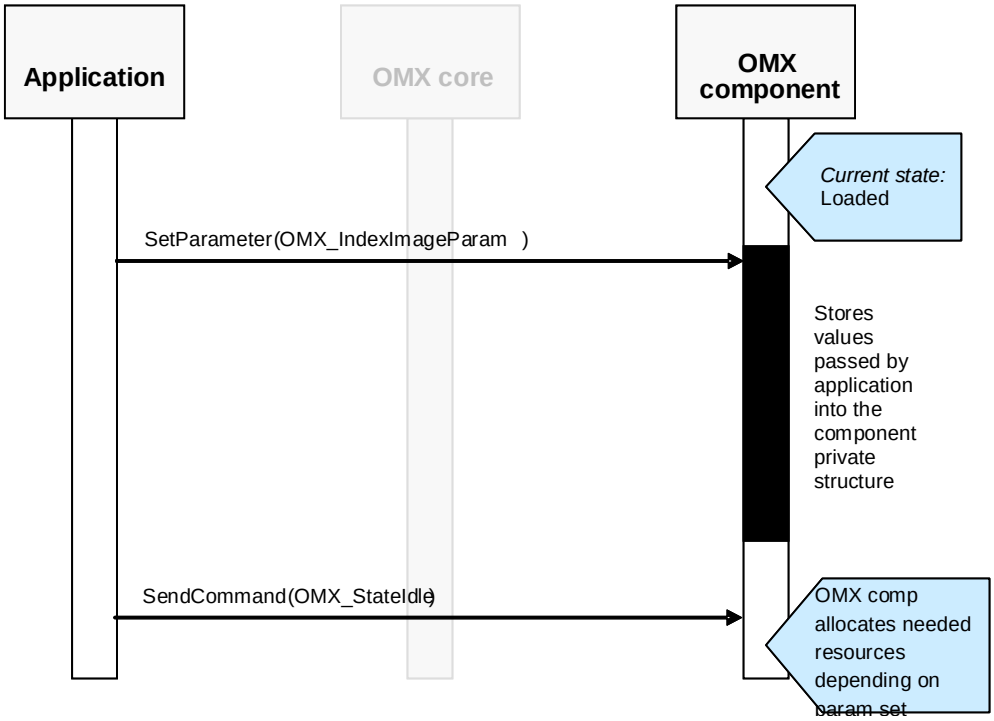


Figure 6 Initialization Phase of IMG Component – Parameter Setting

After the component is brought into loaded state, the component can be setup with different parameters. After receiving any necessary parameters the component can be made ready for execution by moving to IDLE state. When the component is in IDLE state no more parameters can be set. In IDLE state application can only change runtime changeable parameters using OMX_SetConfig. The Following sequences of actions describe how initialization is done:

- Prior to the initialization, the application will have obtained a valid OpenMAX™ 1.0 IMG Component Handle using OMX_GetHandle.
- The application will then use one or more initialization parameter structures depending on how the application wants to use the component instance. The OpenMAX™ 1.0 IMG Component can be initialized for encoding, decoding or doing any of the image processing tasks. The behavior depends on the parameter values passed during the loaded state. Once the OpenMAX™ 1.0 IMG Component is initialized for say JPEG enc and state transitioned to IDLE, then this instance cannot be used for other img processing tasks (like jpeg decode or color conversion). A new instance of the component needs to be created for each of these separate tasks. The details on which parameter index needs to be passed and what configuration structures can be passed are detailed in next section.
- Next the application sends idle command to the component by making a call to function OMX_SendCommand with second argument as OMX_CommandStateSet and third argument as OMX_StateIdle. In response to this command, the IMG Component will allocate the input and output buffers (number and size as specified by the Application) etc.
- The final action for Component Initialization is for the component to issue a callback (EventHandler) to the application to notify that the initialization process has been completed (successfully or with an error).

4.2.4 OpenMAX™ 1.0 IMG Component Execution

Figure 7 shows OpenMAX™ 1.0 IMG Component's transition from idle to execute.

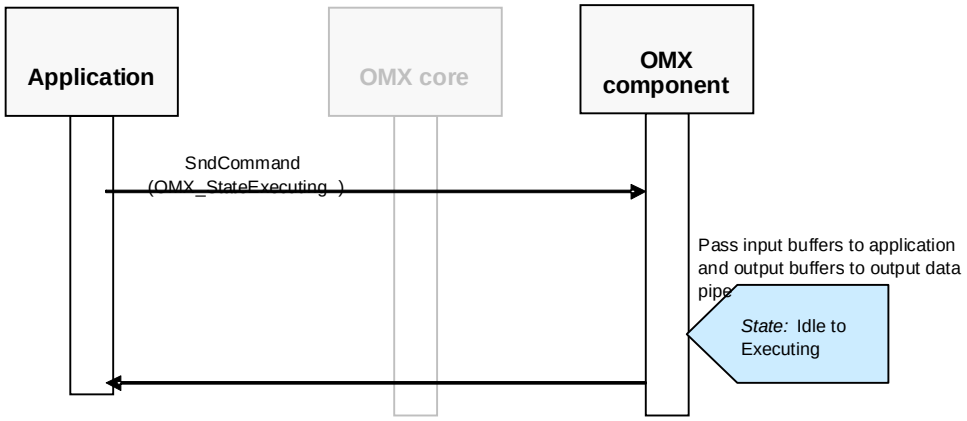


Figure 7 Execution Phase of IMG Component

OpenMAX™ 1.0 IMG Component is put into execution state by making a call to function OMX_SendCommand with second argument as OMX_CommandStateSet and third argument as OMX_StateExecuting. When OpenMAX™ 1.0 IMG Component reads this command, it invokes ProcessFunction function, which does the following:

- In case IMG Component allocated input buffers, Call EmptyBufferDone to provide input buffers to application.
- In case IMG Component allocated output buffers write output buffer headers into output data pipe.

4.2.5 OpenMAX™ 1.0 IMG Component Pause

The entry criterion for this phase is that the IMG Component should be in idle or executing state. The application sends the pause command to the component using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StatePause. On receiving this command, IMG Component state will be changed to pause while the component continues to process the buffers.

4.2.6 OpenMAX™ 1.0 IMG Component Resume

The entry criterion for this phase is that the IMG Component should be in paused state. The application sends the execute command to the component using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateExecute. On receiving this command IMG Component state will be changed to Executing.

4.2.7 OpenMAX™ 1.0 IMG Component Stop

Figure 8 shows OpenMAX™ 1.0 IMG Component's transition from executes to idle.

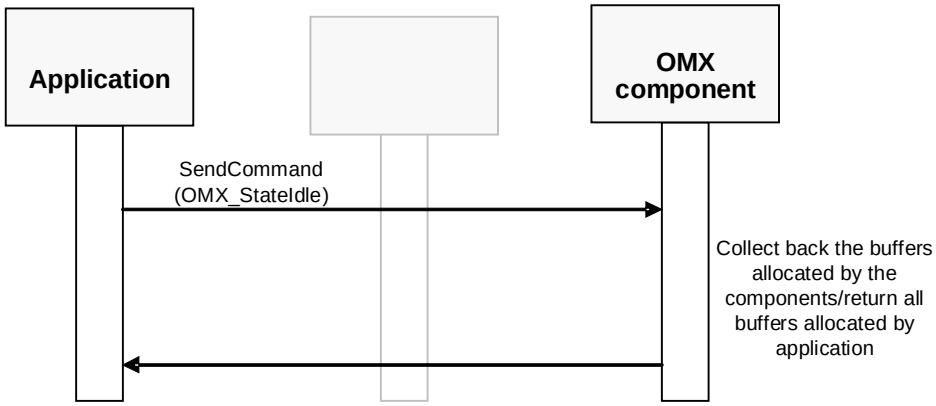


Figure 8 Stop Command from application

OpenMAX™ 1.0 IMG Component is put into idle state by application using core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateIdle. When OpenMAX™ 1.0 IMG Component reads this command, it invokes ProcessFunction function, which does the following:

- Return the buffers to application if application allocated buffers.
- Collects back all the buffers if allocated by the component.
- Move it idle state only when all the buffers are with respective owners.

4.2.8 OpenMAX™ 1.0 IMG Component De-initialization

The entry criterion for this phase is that the component should be in idle state. The application sends the De-initialization command to the IMG Component using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateLoaded. On receiving this command from Application, IMG Component will de-allocate buffers and buffer headers.

4.2.9 Valid State Transitions

For detailed description of states and valid component state transitions, please refer OpenMaxIL1.0 Specification.

4.3 Configuration And Data Flow Scenarios

OpenMAX™ 1.0 IMG Component can be configured for different image processing scenarios like JPEG encode, JPEG decode, image rescale, color conversion etc. Most of the control and data flow are similar. The generic flow is explained first followed by the detail configuration and data flow for each scenarios.

4.3.1 Generic Configuration And Data Flow

The generic sequence is:

- Application calls function OMX_GetHandle of OMX core and supplies the name of OpenMAX™ 1.0 IMG Component and callbacks of application as arguments to this function. The name of IMG Component is “OMX_TIIMAGE_COMPONENT” and this is defined in OMX_TIImage.h file.
- Application calls function OMX_SetParameter of OMX core providing the needed behavior specific configuration structure. This is the function which determines the core behavior of OpenMAX™ 1.0 IMG Component. If the component is created for encoding and image, then JPEG encode specific parameter needs to be set and if the component is created for jpeg decode then JPEG decode specific parameter needs to be set and similarly for other features. The details on which parameter needs to be set is detailed in the specific scenario sections below.
- Depending on the previous behavior parameter setting, additional configuration and parameter can be set by calling the OMX_SetParameter or OMX_SetConfig function calls. The details on which additional parameters and configuration structures can be passed are explained in scenario specific subsections below.
- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application calls OMX_EmptyThisBuffer or OMX_FillThisBuffer calls to send input and output buffers to components. Depending on usage scenario, these APIs needs to be send single time or multiple times. The details can be found in specific subsections below.
- Once the processing is over Application calls OMX_FreeHandle to destroy the instance of OpenMAX™ 1.0 IMG Component.

4.3.2 JPEG Encode Scenario

This feature enables an application to encode a raw YUV image to JPEG baseline format. The input image could be of the following format

a) YUYV interleaved

b) YUV 420planar

c) YUV 422planar

The input image could be of any arbitrary image dimension and the user has an option to specify the quality factor. No progressive image compression is supported.

The sequence of calls for a encoder is:

- Application calls function OMX_GetHandle of OMX core and supplies the name “OMX_TIIMAGE_COMPONENT” and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_ENCODE_PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_ENCODE_PARAMTYPE. Here the user specifies the type of encoding, quality factor for compression, the image width and height, the input image format type, the encoded format type, the stride and the access unit size. In this version only JPEG encoding is supported and also the input image format can only be among OMX_TICOLOR_YUV420P, OMX_TICOLOR_YUV422H, OMX_TICOLOR_YUV422V or OMX_TICOLOR_YUV422I.

- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application sends the image to be encoded with a OMX_EmptyThisBuffer call. The encoder instance is created and initialized.
- Incase the entire image is to be encoded at one shot, the application queries the encoder for the approximate size of the encoded file using which the output buffer size is determined with a OMX_GetConfig call with the OMX_TIIMAGE_ENCODE_OUTBUFFER_INDEX as the second parameter and OMX_TIIMAGE_OUTBUFFERTYPE as the third parameter. If it is a MCU based encoding, the application allocates an output buffer of the same size as the input buffer.
- The application sends the output buffer to the component with a OMX_FillThisBuffer call.
- The encoding operation is performed. On successful completion, the component calls the EmptyBufferDone callback to the application with the buffer supplied by the application and the FillBufferDone callback with the encoded frame.
- In case off multiple pass encoding, when the application has the next available input or output buffer it can call the component with a EmptyThisBuffer or FillThisBuffer call and continue the encoding operation. Both the buffers and the commands are queued up internally and processed.
- At any time during the operation the application can query the component using GetConfig calls and also change the configuration using SetConfig calls.
- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

4.3.3 JPEG Decode Scenario

This feature enables an application to decode an baseline JPEG encoded image to YUV format. The decoder has the ability to support JPEG streams of following source formats

a) YUV 420

b) YUV422H

c) YUV422V

d) YUV444

e) RGB444

f) Monochrome

Note that decoder will not crop the width and height padding for images which are non-multiples of MCUs. The application needs to remove this padding using color conversion feature which supports cropping.

The sequence of calls for a decoder is:

- Application calls function OMX_GetHandle of OMX core and supplies the name “OMX_TIIMAGE_COMPONENT” and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_DECODE_PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_DECODE_PARAMTYPE. Here the user specifies the type of encoding, quality factor for compression, the image width and height, the input image format type, the encoded format type, the stride and the access unit size. In this version only JPEG encoding is supported and also the input image format can only be among OMX_TICOLOR_YUV420P, OMX_TICOLOR_YUV422H, OMX_TICOLOR_YUV422V or OMX_TICOLOR_YUV444, OMX_TICOLOR_RGB444, OMX_TICOLOR_MONOCHROME.
- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.

- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE
- Application sends the image to be decoded with a OMX_EmptyThisBuffer call and the decoder instance is created and initialized.
- Application sends a getConfig call with the second parameter as OMX_TIMIMAGE_OUTBUFFERINDEXTYPE and the third structure as OMX_TIIMAGE_OUTBUFFER_TYPE. This returns the output image details like the image format, width, height information.
- In case of one shot decoding, the application allocates an output buffer of output image size. For decoding with multiple passes, the application allocates a buffer of size double that of the input buffer.
- The application sends the output buffer to the component through a OMX_FillThisBuffer call.
- The decoding operation is performed. On successful completion, the component calls the EmptyBufferDone callback to the application with the input buffer supplied by the application and the FillBufferDone callback with the output buffer.
- When the application has the next available input or output buffer it can call the component with a OMX_EmptyThisBuffer or OMX_FillThisBuffer call and continue the encoding operation.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.
- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

4.3.4 Color Conversion Scenario

This feature enables an application to convert image data from one format to another. The following conversion features are supported.

Table 19 Color conversion Matrix

Source color format	Output color format
RGB565	YUYV
YUV420	YUYV
YUV422H	YUYV
YUV422V	YUYV
YUV444	YUYV
RGB444	YUYV
RGB444	RGB16
RGB444	RGB24
YUV420	RGB16
YUV422H	RGB16
YUV422V	RGB16
YUV420	RGB24
YUV422H	RGB24
YUV422V	RGB24

The sequence of calls for color conversion is:

- Application calls function OMX_GetHandle of OMX core and supplies the name “OMX_TIIMAGE_COMPONENT” and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_ COLORCONVERSION _PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_COLORCONVERSION_PARAMTYPE. Here the user specifies the input image

width, height, color format, output color format, X and Y offset and a flag to indicate in-place conversion.

- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application sends the image to be color converted with a OMX_EmptyThisBuffer call.
- The application also allocates a buffer of the same size as the input buffer and sends it to the output port of the component through a OMX_FillThisBuffer call. If the bInPlace flag is set, no separate output buffer is allocated.
- On successful completion, the component calls the EmptyBufferDone callback to the application with the input buffer supplied by the application and the FillThisBufferDone callback with the output buffer.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.
- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

4.3.5 Rescale/Zoom Scenario

This feature enables an application to rescale or zoom an RGB444 or YUV image. This feature has the ability to crop an input image too.

The sequence of calls for rescale/zoom is:

- Application calls function OMX_GetHandle of OMX core and supplies the name “OMX_TIIMAGE_COMPONENT” and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_ RESCALE _PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_RESCALE_PARAMTYPE. Here the user specifies the input image width, height, color format, X and Y offset, zoom factor, start, stop and increment zoom factors for auto-zoom mode and a flag for resetting of zoom factor once it reaches the final value. If the image needs to be only cropped, the zoom factor is 1 and the x and y offsets determine the cropped region. If the image is to be rescaled in an auto-zoom mode, the start and end zoom factors are provided along with the zoom increment.
- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application sends the image to be rescaled with a OMX_EmptyThisBuffer call.
- The application also allocates a buffer of the same size as the input buffer and sends it to the output port of the component through a OMX_FillThisBuffer call.
- On successful completion, the component calls the EmptyBufferDone callback to the application with the input buffer supplied by the application and the FillThisBufferDone callback with the output buffer.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.
- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

4.3.6 Overlay Scenario

This feature enables overlay or alpha blending of two images.

Overlay : enables an application to overlay a given image on top of another image at a specific location.

Alpha blending : enables an application to blend to RGB565 images together. The blending will depend on the transmittivity value set for the image. The following transmittivity values are supported a) 1/4th b) ½ and c) 3/4$^{th.}$

Both images should be in RGB565 format.

The sequence of calls is:

- Application calls function OMX_GetHandle of OMX core and supplies the name "OMX_TIIMAGE_COMPONENT" and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_ OVERLAY _PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_OVERLAY_PARAMTYPE. Here the user specifies the input and output width, height, color format, X and Y offset, overlay type, colour key and a flag for inplace overlay. The overlay type parameter determines whether the operation is to be a colour key replacement or a alpha blending.
- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application sends the images to be overlaid with a OMX_EmptyThisBuffer calls to the two ports of the component.
- The application also allocates a buffer of the same size as the source buffer and sends it to the output port of the component through a OMX_FillThisBuffer call.
- On successful completion, the component calls the EmptyBufferDone callback to the application with the input buffer supplied by the application and the FillThisBufferDone callback with the output buffer.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.
- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

4.3.7 Rotation Scenario

This feature enables an application to rotate an RGB565 or YUYV image by 90, 180 or 270 degrees.

The sequence of calls for rotation is:

- Application calls function OMX_GetHandle of OMX core and supplies the name "OMX_TIIMAGE_COMPONENT" and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_ ROTATE _PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_ROTATE_PARAMTYPE. Here the user specifies the input image width, height, color format, rotation factor and a flag for in-place rotation.
- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application sends the image to be rotated with a OMX_EmptyThisBuffer call.
- The application also allocates a buffer of the same size as the input buffer and sends it to the output port of the component through a OMX_FillThisBuffer call.
- On successful completion, the component calls the EmptyBufferDone callback to the application with the input buffer supplied by the application and the FillThisBufferDone callback with the output buffer.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.

- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

4.3.8 Effects Scenario

This feature enables an application to convert a color image to gray scale or add sepia tone to a color image. The input format could be YUV422H, YUV422V, YUV444, YUV420 and RGB444. The output format will be same as input format.

The sequence of calls for these effects is:

- Application calls function OMX_GetHandle of OMX core and supplies the name "OMX_TIIMAGE_COMPONENT" and callbacks of application as arguments to this function.
- Application calls OMX_SetParameter with the second parameter as OMX_TIIMAGE_ EFFECT_PARAMINDEX and the third parameter as a structure of type OMX_TIIMAGE_EFFECT_PARAMTYPE. Here the user specifies the input image width, height, color format, effect type which is OMX_TIIMAGE_SEPIAEFFECT / OMX_TIIMAGE_GRAYEFFECT and a flag for in-place conversion.
- Application calls OMX_SendCommand function to change the state of the component from LOADED to IDLE.
- Application calls OMX_SendCommand function to change the state of the component from IDLE to EXECUTE.
- Application sends the input image with a OMX_EmptyThisBuffer call.
- The application also allocates a buffer of the same size as the input buffer and sends it to the output port of the component through a OMX_FillThisBuffer call.
- On successful completion, the component calls the EmptyBufferDone callback to the application with the input buffer supplied by the application and the FillThisBufferDone callback with the output buffer.
- At any time during the operation the application can query the component using getConfig calls and also change the configuration using setConfig calls.
- The application can delete the instance of the component using a OMX_FreeHandle call which internally calls the ComponentDeInit function. All the buffers allocated by the application need to be freed after the component is freed.

5 Memory Requirements

The OpenMAX™ 1.0 IMG Component memory requirements are as shown in Table 20.

Table 20 Memory Requirements for IMG Component

Items	Memory required
RAM	(To be updated)
FLASH (code size)	(To be updated)

5.1 Memory Allocation

Table 21 gives details about minimum values for OMX input and output buffers.

Table 21 Minimum Values for OMX Input and Output Buffers

Buffer Type	No of Buffers	Size of Each Buffer
Input Buffers	Minimum 2	tbd Bytes each
Output Buffers	Minimum 2	tbd Bytes each

6 Software Requirements

The feature requirements for the OpenMAX™ 1.0 IMG Component are listed in Table 22 below. Each of these features are covered in details in section 2.2.

Table 22 Requirements List

SR Tag/Index	Feature Requirement text
1	The component shall support baseline JPEG encoding
2	The component shall support baseline JPEG decoding
3	The component shall support image rotation
4	The component shall support image overlay
5	The component shall support image alpha blending
6	The component shall support sepia effect
7	The component shall support gray scale effect
8	The component shall support color conversion
9	The component shall be compliant with openMax1.0 standard