OMAP™ SS&P DESIGN SPECIFICATION

OpenMAX™ 1.0 Nucleus® TCS2300 IMG Component

Document Revision: 1.0

Issue Date: 18 November 2005

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

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 Camera Component. The environment for OpenMAX™ 1.0 Camera Component is:

Nucleus® on TCS2300.

OpenMAX™ 1.0 Camera Component.

Revision 2.x or later I-Sample board.

1.1 Purpose

This document details the design specifications for OpenMAX™ 1.0 Nucleus® Camera Component on TCS2300.

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_Locosto_OMX_Camera_DesignSpec.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 Camera Component facilitates the interaction between an Application and the Camera Driver. Communication between OpenMAX™ 1.0 Camera Component and Camera Driver 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 Camera 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 Camera Component's interfaces to send commands to the underlying Camera Driver to perform operations such as initialize, start, and stop of different camera operations. In this usage, OpenMAX™ 1.0 Camera Component is said to wrap the underlying Camera Driver.

To offer services to an application, OpenMAX™ 1.0 Camera Component works in unison with the OpenMAX™ 1.0 core. For details on OpenMAX™ 1.0 core, please refer to OMAPSSP_V1030_OMX_Core_DesignSpec.

2.1 Features

The table1 lists the high-level features that OpenMAX™ 1.0 Camera Component. Details on initialization, configuration and data flow for each of these features are explained later in the document.

Table 2 Feature List

Feature	**Description**
Snapshot Capture	Snapshot image capture upto VGA dimension
ViewFinder mode	
Color format support	
Sharpness control	
Gamma correction	

2.2 System Design

The OpenMAX™ 1.0 Camera Component is divided into two separate components

1) Camera Driver.

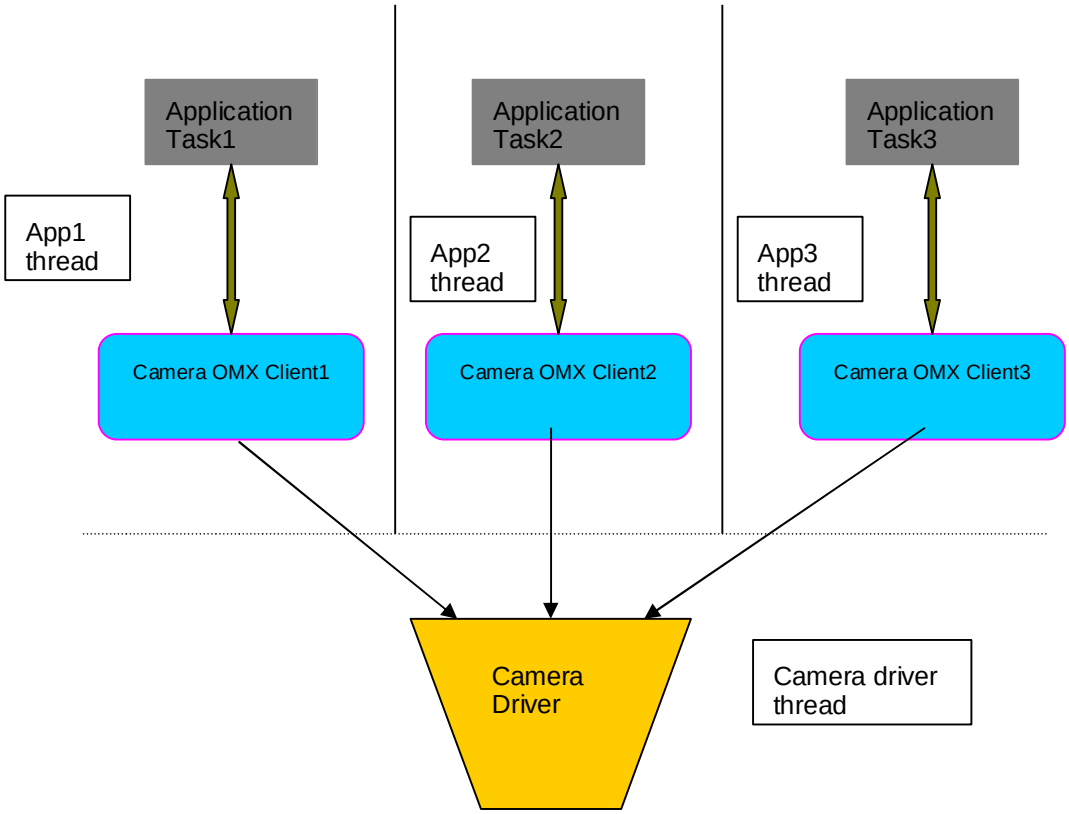
2) Camera OMX client

Camera driver is a separate thread running in the system and facilitates actual data capture and configuration from the hardware. Camera driver is started when the system boots up and it waits for commands from different IMG Camera OMX clients. There is only one instance of this thread running in the system. Camera driver interface is not exposed to application space and only Camera OMX clients interface is exposed.

Camera OMX client is the interface which enables applications to facilitate different camera operation. 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 Camera OMX clients as per the requirement (since there is only Camera in the system, only one client's request will be serviced and all other request will return "resource not available" error). All these clients will interface with Camera Driver to carry out the actual camera operation. But this client to Core communication is hidden from the application and the application just need to invoke the OMX client APIs to carry out the functionality.

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

Figure 1 OpenMAX™ 1.0 Camera Component 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.

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 Camera 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 Camera Component to access common items.
OMX_Index.h	Contains the definitions of parameter and configuration indices for both application and Camera Component.
OMX_Image.h	This file contains the structures needed by Image components to exchange parameters and configuration data between the application and the Camera Component.
OMX_TICamera.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 Camera 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_TICAMERA_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 Camera Component. This string is defined in OMX_TICamera.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_RGB444	
OMX_TICOLOR_RGB565	
OMX_TICOLOR_RGB888	
OMX_TICOLOR_YUV444	
OMX_TICOLOR_YUV422H	
OMX_TICOLOR_YUV422V	
OMX_TICOLOR_YUV422I	
OMX_TICOLOR_YUV420P	

3.3.2 OMX_TICAMERA_PARAMTYPE

This parameter is passed to the OpenMAX™ 1.0 Camera Component using the standard OMX SetParameter function call. The index value for this param type is OMX_ TICAMERA _PARAMINDEX (defined in OMX_TICamera.h). This parameter needs to be set if the OMX IMG client needs to be configured for JPEG encoding. The structure members are detailed in the table below.

Table 5 OMX_ TICAMERA _PARAMTYPE Structure

Data field Name	Description or Evaluation
`tCaptureMode (viewfinder, image)`	It could be viewfinder, video or snapshot. Enum types are defined in OMX_TICamera.h
`iImageWidth`	Width of the image
`iImageHeight`	Height of the image
`iImageFormat`	This OMX_TICOLOR_FORMATTYPE enum type field indicates the format of the image to be captured.
`bHalfToneEnable`	If set, dithering will be enabled for the image.
`iFrameRate`	Frame rate for video and viewfinder mod.
`iSharpness`	Sharpness control value
`iGamma`	Gamma correction value
`iContrast`	Contrast setting value
`bHorizontalMirror`	If set to TRUE, the image will be horizontally flipped
`bVerticalMirror`	If set to TRUE, the image will be vertically flipped
`tRotation`	Rotation type
`tFlashmode`	Flash mode type
`tExposureMode`	Exposure control mode
`iOpticalZoom`	Optical zoom value
`iDigitalZoom`	Digital zoom value

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 Camera Component etc. These macros are defined in OMX_Core.h. Each macro maps to a function implemented by the OpenMAX™ 1.0 Camera Component. Detailed description of each function implemented by the Camera 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_Camera_ComponentInit

```
OMX_ERRORTYPE OMX_Camera_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_Camera_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 Camera Component as OMX_StateLoaded.

3.4.2 OMX_Camera_SetCallbacks

```
OMX_ERRORTYPE OMX_Camera_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_Camera_ComponentInit, the OMX core calls this function to provide application callbacks to the OpenMAX™ 1.0 Camera 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_Camera_GetComponentVersion

```
OMX_ERRORTYPE OMX_Camera_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 Camera Component fills component version, OMX specification version and UUID information in the structures passed by application.

3.4.4 OMX_Camera_SendCommand

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

Description

Sends a command to the component. Currently there is only 1 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_Camera_GetParameter

```
OMX_ERRORTYPE OMX_Camera_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_JPEGENCODE_PARAM_TYPE.

These structures are defined OMX_Image.h and OMX_TIImage.h

3.4.6 OMX_Camera_SetParameter

```
OMX_ERRORTYPE OMX_Camera_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_AUDIO_PARAM_TYPE

These 3 structures are defined OMX_Image.h and OMX_TIImage.h

3.4.7 OMX_ Img_GetConfig

```
OMX_ERRORTYPE OMX_Camera_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 Camera 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 Camera Component not supports any configuration structures.

3.4.8 OMX_Camera_SetConfig

```
OMX_ERRORTYPE OMX_Camera_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 Camera 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 Camera Component not supports any configuration structures.

3.4.9 OMX_Camera_GetState

```
OMX_ERRORTYPE OMX_Camera_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.

3.4.10 OMX_Camera_EmptyThisBuffer

```
OMX_ERRORTYPE OMX_Camera_EmptyThisBuffer (OMX_HANDLETYPE hComp,
                                          OMX_U32 nPortIndex,
                                          OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

The camera component is source component. It does not consume any buffer. So this API should never be called by application. It always returns OMX_ErrorPortsNotCompatible error.

Parameters

Name	Type	Description
HComp	IN	This input argument is the component handle.
NportIndex	IN	This specifies an input port of the component.
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.

3.4.11 OMX_Camera_FillThisBuffer

```
OMX_ERRORTYPE OMX_Camera_FillThisBuffer (OMX_HANDLETYPE hComp,
                                         OMX_U32 nPortIndex,
                                         OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

The application calls this function to send an empty buffer to the output port of the component. In snapshot mode, this API needs to be called once to capture the snapshot image. In viewfinder or video mode, this API needs to be called multiple times.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
nPortIndex	IN	This specifies an output port of the component.
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.

3.4.12 OMX_Camera_ComponentTunnelRequest

```
OMX_ERRORTYPE OMX_Camera_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.

3.4.13 OMX_Camera_ComponentDeInit

```
OMX_ERRORTYPE OMX_Camera_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 Camera 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 Camera 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_Camera_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 Camera Component States

OpenMAX™ 1.0 Camera Component will exist in one of five states at any given time. Figure 2 represents the state diagram for the OpenMAX™ 1.0 Camera Component. The Camera 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 Camera Component

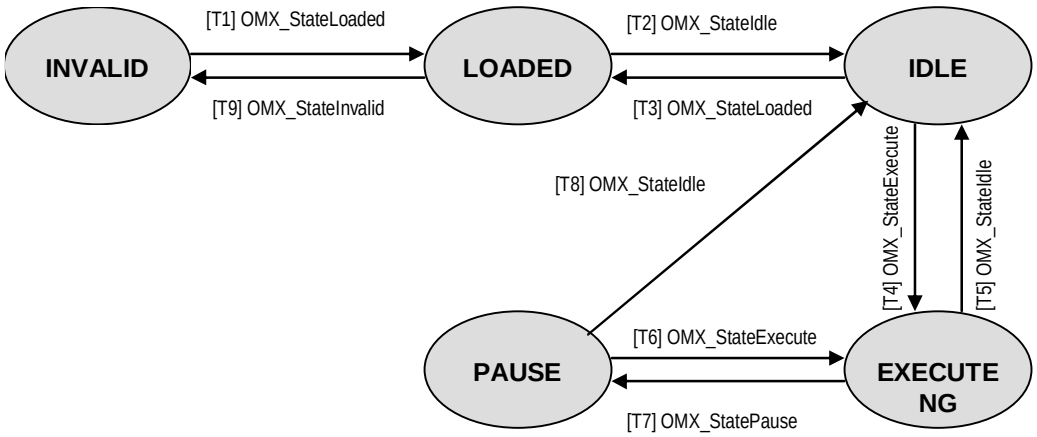


Table 6 shows state transitions for the OpenMAX™ 1.0 Camera Component and the functions/triggers that initiate these transitions.

Table 6 Camera Component State Transition

State	State change	Function/trigger
T1	Invalid->Loaded	OMX_GetHandle() Get the Handle to Camera Component Allocate resources needed for component execution
T2	Loaded->Idle	OMX_Camera_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_Camera_SendCommand (stateLoaded) Deallocate input buffers Deallocate output buffers Deallocate buffer headers for input buffers Deallocate buffer headers for output buffers
T4	Idle->Execute	OMX_Camera_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_Camera_SendCommand (stateIdle) Reclaim all component allocated buffers and return all application allocated buffers.
T6	Pause->Execute	OMX_Camera_SendCommand (stateExecute) 'Execute' State request from Application results in change of current state of Camera Component from pause to Execute.
T7	Executing->Pause	OMX_Camera_SendCommand (statePause) 'Pause' State request from Application results only in change of current state of Camera Component to 'pause'. Data processing will be halted until it transitions to execute state.
T8	Pause->Idle	OMX_Camera_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 Camera Component state are illustrated below in Figure 3.

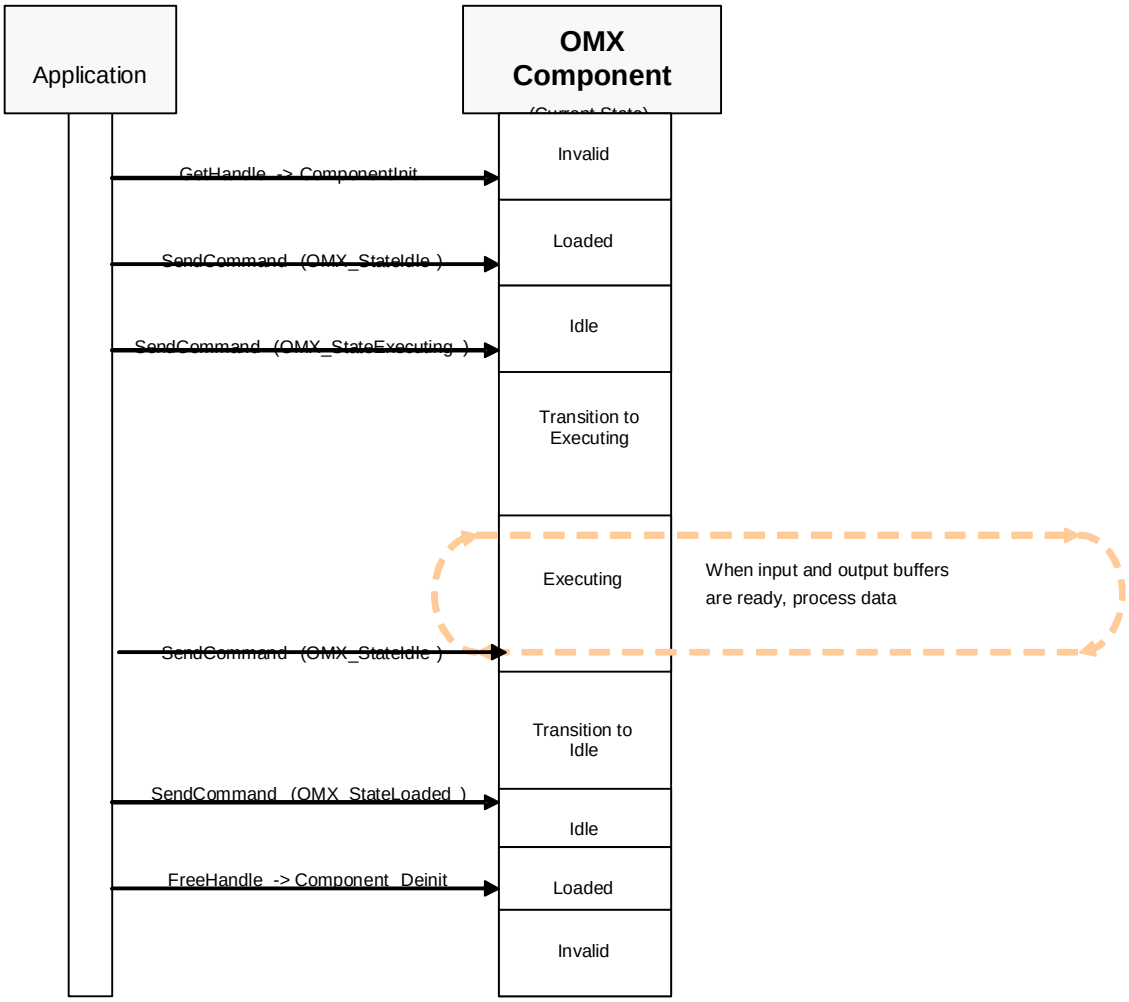


Figure 3 Camera Component State Transitions

4.2 OpenMAX™ 1.0 Camera Component Phases

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

4.2.1 OpenMAX™ 1.0 Camera Component Load

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

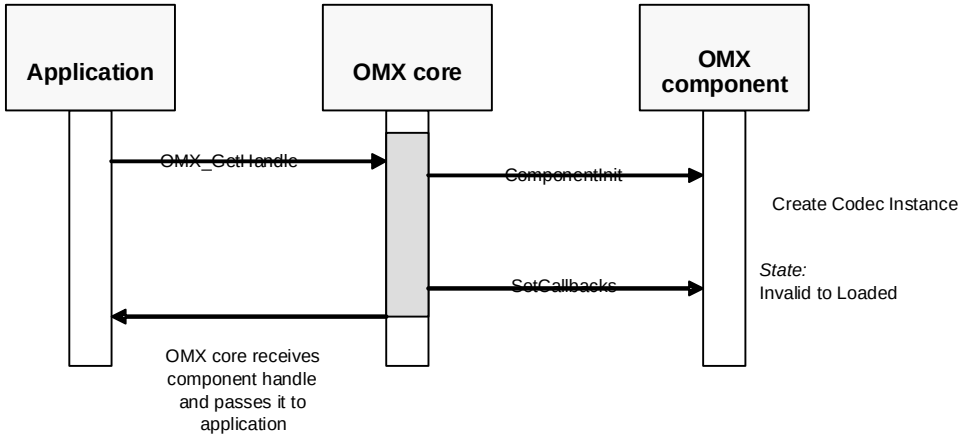


Figure 4 Loading of Camera Component

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

application calls function OMX_GetHandle of OMX core and supplies the name of OpenMAX™ 1.0 Camera Component and callbacks of application as arguments to this function.

The OMX core searches for OpenMAX™ 1.0 Camera 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_Camera_ComponentInit function of the component.

OMX_Camera_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_Camera_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 Camera Component's handle structure with pointers to functions exported by the component.

4.2.2 OpenMAX™ 1.0 Camera Component Unload

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

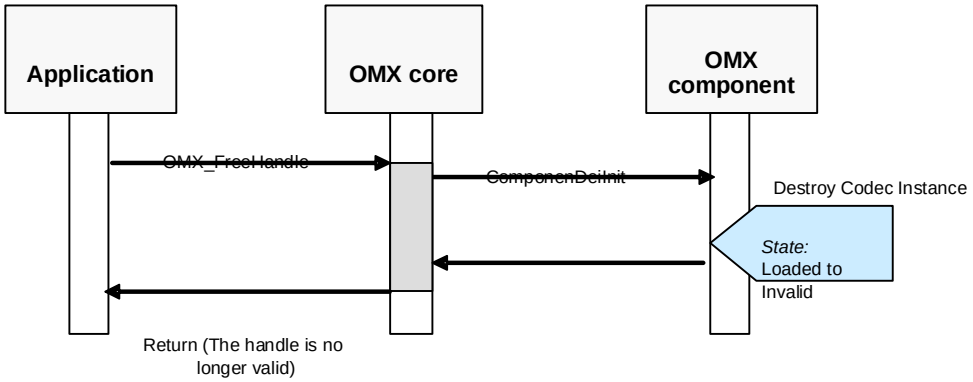


Figure 5 Unloading of Camera Component

The following sequence is followed while unloading OpenMAX™ 1.0 Camera 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 Camera Component.

The Core method OMX_FreeHandle in turn calls the OMX_Camera_ComponentDeInit function of the component. The OMX_Camera_ComponentDeInit deallocates the resources that were allocated by the component.

The OMX core then frees the memory used by OpenMAX™ 1.0 Camera Component handle.

4.2.3 OpenMAX™ 1.0 Camera Component Initialization

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

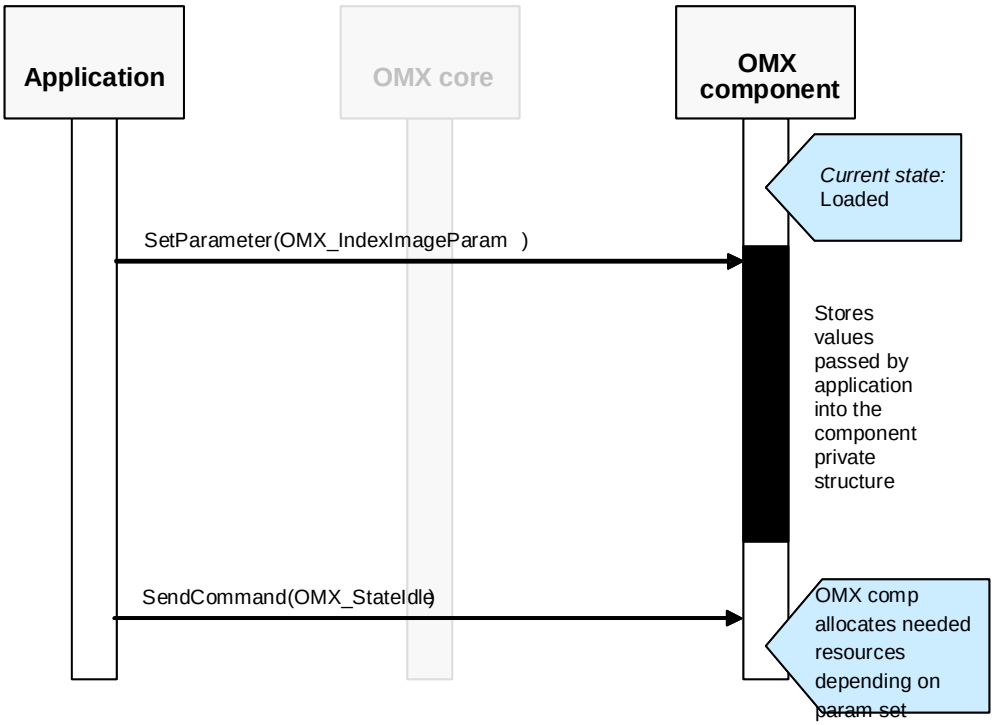


Figure 6 Initialization Phase of Camera Component – Parameter Setting

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 Camera 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 Camera 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 Camera 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 Camera 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 Camera Component Execution

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

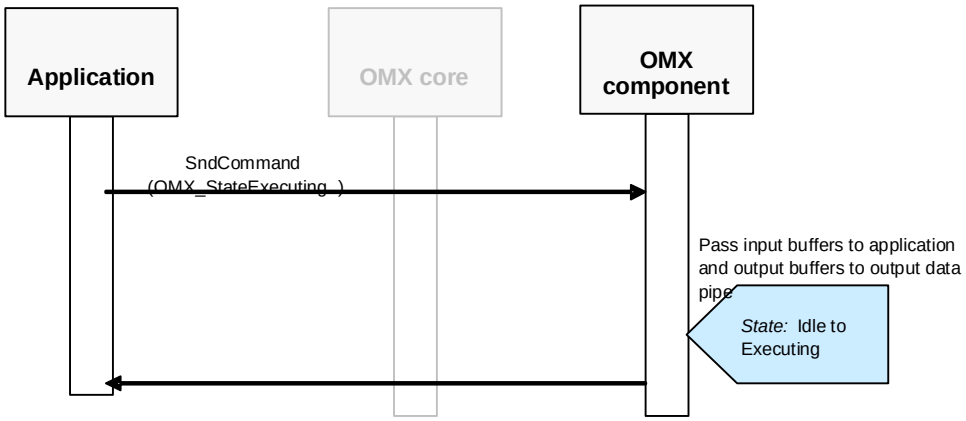


Figure 7 Execution Phase of Camera Component

OpenMAX™ 1.0 Camera 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 Camera Component reads this command, it invokes ProcessFunction function, which does the following:

In case Camera Component allocated input buffers, Call EmptyBufferDone to provide input buffers to application.

In case Camera Component allocated output buffers write output buffer headers into output data pipe.

4.2.5 OpenMAX™ 1.0 Camera Component Pause

The entry criterion for this phase is that the Camera Component should be in 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, Camera Component state will be changed to pause while the component continues to process the buffers.

4.2.6 OpenMAX™ 1.0 Camera Component Resume

The entry criterion for this phase is that the Camera 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 Camera Component state will be changed to Executing.

4.2.7 OpenMAX™ 1.0 Camera Component Stop

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

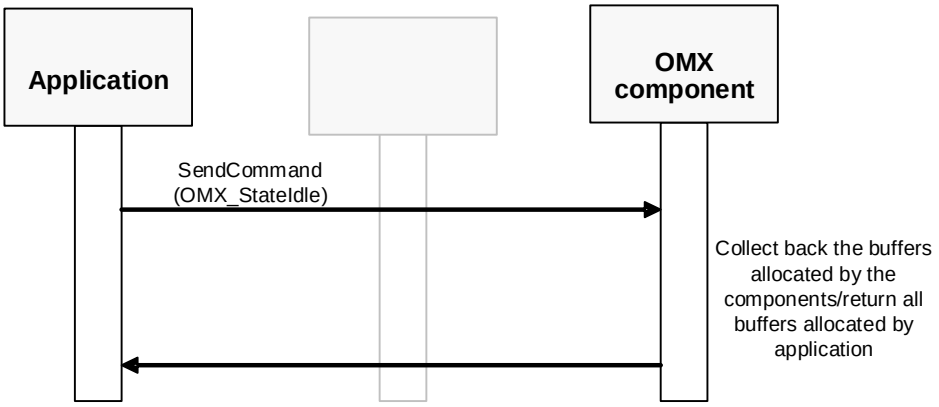


Figure 8 Stop Command from application

OpenMAX™ 1.0 Camera 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 Camera 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 Camera 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 Camera 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, Camera 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 to Section 7.2.1 of OMAPSSP_V1030_OMX_Core_DesignSpec.doc.

4.3 Configuration And Data Flow Scenarios

OpenMAX™ 1.0 Camera Component can be configured for viewfinder mode and snapshot mode. The exact sequence of calls/configuration is given below

Application calls function OMX_GetHandle of OMX core and supplies the name of OpenMAX™ 1.0 Camera Component and callbacks of application as arguments to this function. The name of Camera Component is “OMX_TICAMERA_COMPONENT” and this is defined in OMX_TICamera.h file.

Application calls function OMX_SetParameter of OMX core to set the camera in the required configuration. The configuration structure should be OMX_ TICAMERA _PARAMTYPE.

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 state

Application calls OMX_FillThisBuffer for the camera OMX client to fill the buffer. Once the buffer is filled with captured data, the component will call the callback function fillBufferDone to inform the application that capture is complete. This API is called multiple times in video/viewfinder mode.

Application calls OMX_FreeHandle to destroy the instance of OpenMAX™ 1.0 Camera Component once all image captures are complete.

5 Memory Requirements

The OpenMAX™ 1.0 Camera Component memory requirements are as shown in Table 7.

Table 7 Memory Requirements for Camera Component

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

5.1 Memory Allocation

Table 8 gives details about minimum values for OMX input and output buffers.

Table 8 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 Camera Component are listed in Table 9 below. Each of these features are covered in details in section 2.2.

Table 9 Requirements List

SR Tag/Index	Feature Requirement text
1	
2	
3	
4	
5	
6	
7	
8	
9	