OpenMAX Development Layer API Specification

Version 1.0

February 10th, 2006

Document version 1.0

Version	Ref.	Version Information
1.0		Originate

Version 1.0 – February 10, 2006

Table of Contents

List of Tables

Table of Figures

1.0 Overview

1

1.1 Introduction

This document defines the Application Programming Interface (API) for the OpenMAX Development Layer (DL). Published as an open standard by the Khronos Group, the DL provides a set of low-level primitives to ensure portability across processors and hardware acceleration units for audio, video, and imaging codecs used within embedded and/or mobile devices. The principal goal of the DL is to enable portability of silicon acceleration for rich media codecs across diverse processor and acceleration hardware architectures by providing an essential set of "hotspot" primitives.

1.1.1 About the Khronos Group

The Khronos Group is a member-funded industry consortium focused on the creation of open standard APIs to enable the authoring and playback of dynamic media on a wide variety of platforms and devices. All Khronos members are able to contribute to the development of Khronos API specifications, are empowered to vote at various stages before public deployment, and are able to accelerate the delivery of their multimedia platforms and applications through early access to specification drafts and conformance tests. The Khronos Group is responsible for open APIs such as OpenGL ES, OpenML, and OpenVG.

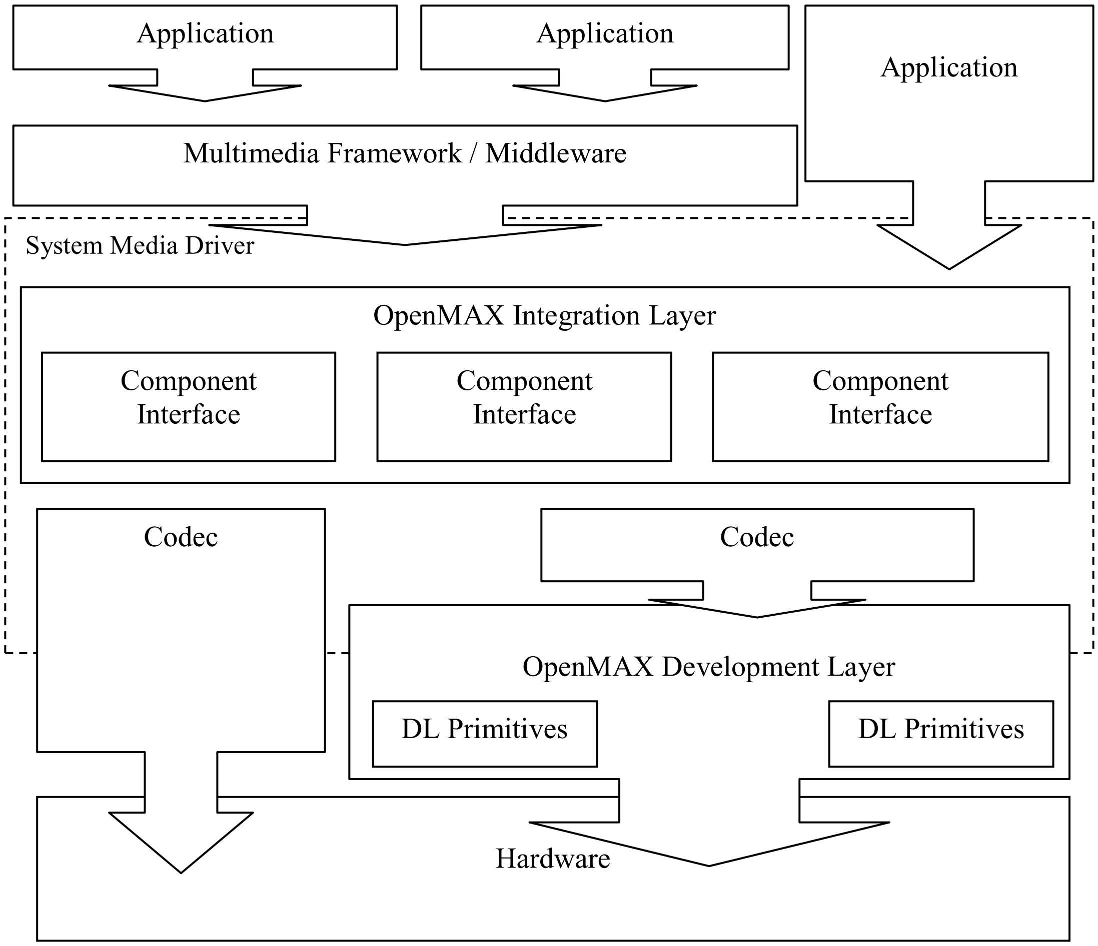
1.1.2 A Brief History of OpenMAX

The OpenMAX set of APIs was originally conceived as a method of enabling portability of codecs and media applications with the mobile device landscape. Brought into the Khronos Group in mid 2004 by a handful of key mobile hardware companies, OpenMAX has gained the contributions of companies and institutions stretching the breadth of the multimedia field. As such, OpenMAX stands to unify the industry in taking steps toward media codec portability. Stepping beyond mobile platforms, the general nature of OpenMAX DL API makes it applicable to all media platforms.

1.2 The OpenMAX Development Layer

1.2.1 Software Landscape

The OpenMAX DL is a part of the larger multimedia software landscape depicted in the figure below.



To remove possible reader confusion, the OpenMAX standard also defines an Integration Layer (IL). The IL and its full relationship to the DL are specified in other OpenMAX specification documents [*OpenMAX Integration Layer API Specification, Version 1.0*, Khronos Group, 2006]

1.2.2 The Interface

The DL API defines a set low-level mulitmedia kernels or media processing building blocks that might be used to accelerate traditional computational hotspots within standardized media codecs and other integrated media processing engines. The functional scope of the DL API spans several domains key to mobile multimedia platforms, including the following: signal and image processing, audio coding, image coding, and video coding. As such, the DL API is organized into a collection of function domains named, respectively, omxSP, omxIP, omxAC, omxIC, and omxVC. Each domain is further decomposed into sub-domains that address the needs of specific standardized media codecs. Within each sub-domain, a set of low-level kernels is defined to provide functional coverage that tends to obey the 80/20 rule.

1.3 Definitions

When this specification discusses requirements and features of the OpenMAX DL API, several specific words are used to convey their nessecity in an implementation. A list of these words can be found in Table 1.

Word	Definition
Shall	“Shall” means that the stated functionality is a requirement for an implementation of the OpenMAX DL API. If a component fails to meet a “shall” statement, it is not considered to be conformant to this specification. Shall is always used as a requirement as in “The component designers shall produce good documentation.”
Will	“Will” means that the stated functionality is not a requirement for an implementation of the OpenMAX DL API. The word “Will” is usually used when referring to a third party as in “the application framework will correctly handle errors.”
Should	“Should” means that the stated functionality is not a requirement for an implementation of the OpenMAX DL API, but is the recommended thing to do or is a good practice. The word “Should” is usually used as follows: “The component should begin processing buffers immediately after it receives a `Start` Command.” While this is good practice, there may be a valid reason to delay processing buffers such as not having input data available.
May	“May” means that the stated functionality is optional requirement for an implementation of the OpenMAX DL API. Optional features are not required by the specification, but may have conformance requirements if they are implemented. This is an optional feature as in “The component may have vendor specific extensions”

Table 1 - Definitions for Commonly Used Words

1.4 Authors

The following individuals contributed to the OpenMAX Development Layer Specification (listed alphabetically by company):

- Martyn Capewell (ARM)
- Hedley Francis (ARM)
- Dominic Symes (ARM)
- Brian Murray (Freescale)
- Yolanda Prieto (Freescale)
- Yong Yan (Freescale)
- Ted Painter (Intel)
- Omry Paiss (Intel)
- Julian Vlaiko (Intel)
- Beryl Xu (Intel)
- Fan Zhang (Intel)
- Kathy Moseler (Motorola)
- Mark Kokes (Nokia)
- Leo Estevez (TI)
- Joseph Meehan (TI)

1.5 Document Organization

The rest of this specification is organized as follows. The remainder of Chapter 1 introduces DL conventions common to all function domains and sub-domains. Chapter 2 defines the signal processing domain (omxSP). Chapter 3 defines the audio coding domain (omxAC), including the MP3 and AAC-LC/LTP sub-domains (omxACMP3, and omxACAAC, respectively). Chapter 4 defines the image processing domain (omxIP). Chapter 5 defines the image coding domain (omxIC), including the JPEG sub-domain (omxICJP). Chapter 6 defines the video coding domain (omxVC), including the MPEG-4 simple profile sub-domain (omxVCM4P2), the H.264 baseline sub-domain (omxVCM4P10), and a set of functions common to both video coding sub-domains (omxVCCOMM). Finally, Chapter 7 defines concurrency mechanisms.

1.6 API Conventions

1.6.1 Function Naming

OpenMAX DL function names are constructed as follows:

omx<domain><sub_domain>_<operation>_<function-specific modifier>_<datatype>_<data modifier>(parameter list)

<domain> **-** two character function domain specifier; the following domains are defined:

Table 1-1: Function Domain definitions

<sub-domain> - two character sub-domain specifier; the following sub-domains are defined:

Domain	Meaning
AC	Audio Coding
SP	Signal Processing
VC	Video Coding
IP	Image Processing
IC	Image Coding

Table 1-2: Function sub-domain definitions

Domain	Sub-domain	Meaning
AC	MP3	MP3
	AAC	AAC
VC	COMM	Common
	M4P2	MPEG4 Part 2
	M4P10	MPEG4 Part 10
IP	PP	Pre- and Post-processing
	CS	Color Space Conversion
	BM	Bitmap Manipulation
IC	JP	JPEG codec functions

<operation> - an abbreviated descriptor that encapsulates function behavior. For example,

“FIR”.

<function-specific modifier> - a short mnemonic string that augments the operation descriptor; used typically when the operation name is imprecise. For example, consider the function `omxSP_FFTFwd_CToC_SC16_Sfs(...)`. The operation “FFTFwd” is a generic operation for which there are multiple instantiations, each with unique characteristics (e.g., real-to-complex, complex-to-complex). The function-specific modifier “CToC” informs the user of the data types processed by this particular function, namely complex-valued input and output vectors.

<data type> - Specifies bit depth and/or data layout using a string of the form:

<U|S>#[c]

Where the "#" symbol is replaced by an integer that indicates the bit depth, either of the symbols "U" or "S" is included to denote, respectively, "unsigned integer" or "signed integer", and the optional symbol "C" denotes complex data. For the functions described in this manual, the "#" symbol is replaced by one of the following bit depth indicators: 8, 16, 32, or 64.

For example, the following function operates exclusively on a single data type:

```
omxSP_Copy_S16(OMX_S16 *pSrc1, OMX_S16 *pSrc2, OMX_S16 *pDst, OMX_INT
len)
```

The data type is specified by the suffix "_S16," which implies that both the input and output operands are represented by 16-bit signed integers (OMX_S16). For functions that operate on more than one data type, the source data type is listed first, followed by destination data type.

`<data modifier>` - The data modifier further describes the data associated with the operation. It may contain implied parameters and/or indicate additional required parameters. The set of OMX data modifiers is given in the list below. Data modifiers are always presented in alphabetical order.

- D2 - two-dimensional signal
- I - in-place operation
- Sfs - Saturated fixed scale operation

1.6.2 Function Arguments

The OpenMAX DL convention for function argument lists can be generally expressed as follows:

```
<input>, <input data length>, <output>, <output data length>,
<parameter list>
```

Whenever an input or output argument is a scalar rather than a vector (non-array), the associated data length argument is eliminated, as in the case of the following example function that computes the standard deviation of a vector:

```
omxSP_StdDev_S16(OMX_S16 * pSrc, OMX_INT len, OMX_S16 * pResult)
```

Whenever the input and output vectors have the same length, the input vector length argument will be eliminated. For example, consider the following function for pointwise vector addition:

```
omxSP_Add_S16(OMX_S16 * pSrc1, OMX_S16 * pSrc2, OMX_S16 * pDst,
OMX_INT len)
```

1.6.3 Data Types

The table below shows OpenMAX data types. Complex-valued sequences are represented using structures that interleave the real and imaginary components.

Table 1-3: OMX Data Types

Data Type	Corresponding Data Type in C
OMX_U8	8-bit unsigned integer, i.e., unsigned char
OMX_S8	8-bit signed integer, i.e., char
OMX_U16	16-bit unsigned integer, i.e., unsigned short, unsigned short int
OMX_S16	16-bit integer, i.e., short, short int, signed short int
OMX_U32	32-bit unsigned integer, i.e., unsigned int, unsigned long, unsigned long int
OMX_S32	32-bit signed integer, i.e., int, long, long int, signed long int
OMX_U64	64-bit unsigned integer
OMX_S64	64-bit signed integer
OMX_SC8	struct {OMX_S8 Re; OMX_S8 Im;}, i.e., real/imaginary interleaved complex
OMX_SC16	struct {OMX_S16 Re; OMX_S16 Im;}, i.e., real/imaginary interleaved complex
OMX_SC32	struct {OMX_S32 Re; OMX_S32 Im;}, i.e., real/imaginary interleaved complex
OMX_SC64	struct {OMX_S64 Re; OMX_S64 Im;}, i.e., real/imaginary interleaved complex
OMXResult	32-bit signed integer, i.e., int

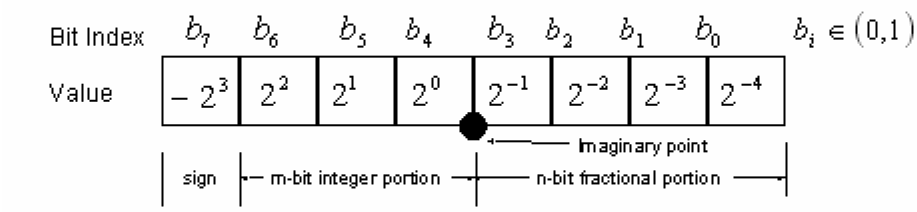
1.6.4 Qm.n Format

Some OpenMAX DL functions require a fractional interpretation of integer input and/or output parameters. The “Qm.n” format provides a standard mechanism for representing fractional values using an integer data type. For example, a Q3.4 word is illustrated in Figure 1-1. The integer binary word has been partitioned using an imaginary fixed point. The n-bits to the right of the imaginary point comprise the fractional portion of the value being represented, and these n-bits act as weights for negative powers of 2. The m-bits to the left of the imaginary point comprise the integer portion of the value being represented, and these m-bits act as weights for positive powers of 2. The overall signed Qm.n representation requires a total of m+n+1 bits, with the additional bit required for the sign. As shown in Figure 1-1, the Q3.4 word requires a total of 3+4+1 = 8 bits. The dynamic range for the Q3.4 word spans the open interval [-8, 8], and the precision (or “quantization error”) is 1/16. The value represented by a particular set of Q3.4 bits is given by the adding up the weighted powers of 2:

$$\texttt{Value} = -b_7 2^3 + b_6 2^2 + \ldots b_0 2^{-4}$$

Where $b_i \in \{0,1\}$ are the binary bits.

Figure 1-1: Q3.4 Format



In general, the m+n+1-bit Qm.n word can be represented as shown in Figure 1-2. In the figure, each bit cell has the value indicated by a power of 2, and the Qm.n word value is determined by adding together the individual bit cell values weighted by the bits, b_i, where $b_i \in \{0, 1\}, 0 \le i \le m + n$, $b_i \in \{0,1\} \le i \le m+r_i$, i.e.:

$$value = -b_{m+n} 2^{-m} + \sum_{i=1}^{m+n} b_{m+n-i} 2^{m-i}$$

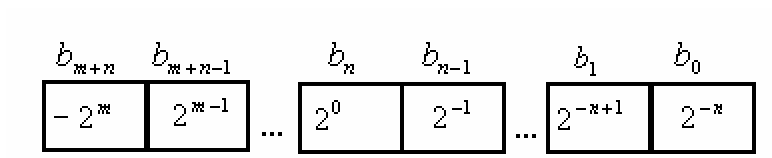
The parameter *m* (number of bits to the left of the point) determines the dynamic range:

$$range = [-2^m, 2^m)$$

and the parameter *n* (number of bits to the right of the point) determines the precision:

$$precision = 2^{-n}$$

Figure 1-2: Qm.n Format



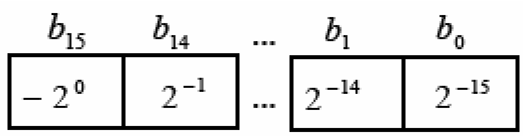
1.6.5 Qm.n Conventions

Two conventions define the use of the Qm.n format for the OpenMAX DL API. First, OpenMAX documents employ the abbreviated notation “Qn” to denote “Qm.n,” where m=L-n-1, and L is the word length, in bits, of the underlying data type. In other words, a particular value for m is implied by the combination of a data type and the particular choice of n specified by “Qn.” The second naming convention is that all functions containing arguments interpreted as Qm.n are prototyped such that the suffix “Qn” is appended to the prototype argument names. For the purposes of illustration, two examples are given next.

Example 1-1: Q0.15, OMX_S16

Consider a Q0.15 parameter with the underlying data type of `OMX_S16`. In this case, L=16, and therefore a Q0.15 interpretation is as shown in Figure 1-3.

Figure 1-3: Q0.15 Example with OMX_S16



The value, dynamic range, and precision can be obtained easily using the expressions given for a general Qm.n. After substituting the values m=0, n=15, it can be seen that the value, range, and precision, respectively, are given by

$$value = -b_{15}2^0 + \sum_{i=1}^{15} b_{15-i}2^{-i}$$

$$range = [-1,1)$$

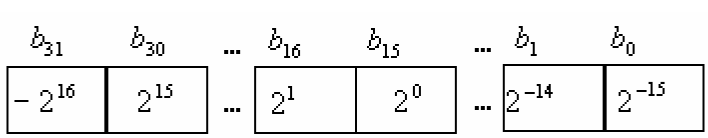
$$precision = 2^{-15}$$

An example function that makes use of Q0.15 in combination with `OMX_S16` is the FIR filter.

Example 1-2: Q16.15, OMX_S32

Next, consider a Q16.15 parameter with the underlying data type of `OMX_S32`. In this case, L=32, and therefore a Q16.15 interpretation is as shown in Figure 1-4.

Figure 1-4: Q16.15 Example with OMX_S32



The value, dynamic range, and precision can be obtained easily using the expressions given for a general Qm.n. After substituting the values m=16 and n=15, it can be seen that the value, range, and precision, respectively, are given by

$$value = -b_{31} 2^{16} + \sum_{i=1}^{31} b_{31-i} 2^{16-i}$$

$$range = [-65536, 65536)$$

$$precision = 2^{-15}$$

1.6.6 Integer Scaling Conventions

The OpenMAX DL API includes a scaling mechanism to achieve the maximum possible precision for fixed-point integer operations. Many functions perform internal computation using a precision higher than the data types that are used for the input and output parameters. This higher precision could be `OMX_INT`, `long`, or `long long`, depending on the implementation and precision requirements. Therefore, it may be necessary to scale the function output arguments to achieve a desired precision in the result. OpenMAX DL provides saturated fixed scaling as a mechanism that allows users to control the precision of output arguments. In functions that use it, saturated fixed scaling (Sfs) is controlled by the input argument, "scalefactor." For Sfs functions, the output will be multiplied by $2^{-scalefactor}$ before returning to the user. In other words, for functions that internally accumulate results having precision higher than the input and output arguments, it is possible for the user to control which subset of the most significant bits is returned.

A typical function with scaled output has the following format:

```
omxSP_Function_Sfs(..., OMX_INT scaleFactor)
```

In all cases, the mnemonic "Sfs" denotes "Saturated fixed scaling."

The Sfs-enabled function performs the required calculation using an internal data type having a higher precision (larger number of bits) than the input and/or output parameters. Once the computation has been completed, the internal high-precision result is shifted by the number of bits indicated in the scale factor (positive scalefactors correspond to right shifts, negative scalefactors to left shifts) and copied into the low-precision output variable.

The user must choose carefully the scalefactor such that significant bits are not truncated from the scaled result. A scaled vector operation using two 16-bit input operands (`OMX_S16`) could potentially produce a result having 32 significant bits. The internal accumulator might actually contain 32 bits. By supplying a scalefactor, the user is able to choose any 16 out of the available 32 result bits. If the top 16 bits of the 32-bit result were needed, then the user would set the scalefactor to 16. On the other hand, if the dynamic range of the input data was constrained such that only 24 significant bits were contained in the internal multiplication result, then the user would select a scalefactor of 8, which would mean that bits 8 through 23 (assuming that bit indices start from 0) were returned in the 16-bit output argument. Clearly, when choosing a scalefactor, the user should consider the dynamic range of the data to avoid the loss of the most significant result bits.

Unless otherwise specified, OpenMAX DL functions implement overflow saturation. Upon overflow, non-scaled OMX integer output arguments will saturate to the maximum possible absolute value. For example, upon overflow, a non-scaled output argument of the type `OMX_S16` will saturate to 0x8000 (-32768) for a negative overflow or 0x7fff (32767) for a positive overflow.

1.6.7 Function Variants

To maximize flexibility and ease of use, the OpenMAX DL API offers up to four variants on each function:

- Basic or default
- In-place (I)
- Saturation fixed scale (Sfs)
- In-place and saturation fixed scale (ISfs)

For in-place function variables, input and output vectors share common memory. As a result, the contents of the input vector are replaced by contents of the output vector upon return from the function call. For non-in-place variables, input and output vectors use distinct memory blocks, and therefore the input vector remains unmodified upon return from the function call. As described previously, saturation fixed scale function variables return outputs that have been scaled by $2^{-scaleFactor}$. i.e., output values have been shifted `scaleFactor` bits to the left or right for negative or positive `scaleFactor` values, respectively. Non-Sfs function variables can be viewed as a special case of the Sfs variables in which `scaleFactor` has been set to 0. The ISfs function variables combine in-place and saturation fixed scale behavior with the underlying default function functionality.

In the interest of clarity and simplicity, the block diagrams, equations, and other detailed behavioral descriptions given throughout the remainder of this specification apply only to the non-in-place and non-scaled (so-called "default") function variants, unless explicitly otherwise noted. Behavior of the scaled and in-place variables can be understood easily by applying the generic in-place and scaled function behavioral rules given above to the default behavioral specification.

1.6.8 Return Codes

Unless otherwise specified, all OpenMAX DL functions return status codes to report errors and warnings to the calling program. The calling function may or may not choose to implement an appropriate exception handling scheme. All DL return codes are of enumerated type `OMXResult`, and all enumerated warnings and errors take negative values, as specified in the omxTypes.h header file. Return codes are classified into two types: mandatory and optional. Mandatory return codes are required for all DL implementations. Optional return codes are recommended but are not required. The complete set of status codes, their associated messages, and their classifications are listed in Table 1-4. The definitions for each function in sections two through six of this document define which mandatory or optional return codes, respectively, shall or may be returned by each particular function.

Table 1-4: Enumerated OpenMAX DL Return Codes

Symbolic Status	Associated Message	Classification
OMX_StsAacGainCtrErr	Unsupported gain control data detected	Mandatory
OMX_StsAacPlsDataErr	Wrong start scalefactor band for pulse data or its position offset	Optional
OMX_StsAnchorErr	The anchor point is outside mask	Optional
OMX_StsBadArgErr	Bad Arguments	Optional

Symbolic Status	Associated Message	Classification
OMX_StsChannelErr	Illegal channel number	Optional
OMX_StsContextMatchErr	The context parameter doesn't match to the operation	Optional
OMX_StsEvenMedianMaskSize	Even size of Median Filter mask was replaced by odd one	Optional
OMX_StsLengthErr	Wrong value of string length	Optional
OMX_StsMaskSizeErr	Invalid mask size	Optional
OMX_StsMemAllocErr	Not enough memory for the operation	Mandatory
OMX_StsMirrorFlipErr	Invalid flip mode	Optional
OMX_StsNoErr	No error	Mandatory
OMX_StsNullPtrErr	Null pointer error	Optional
OMX_StsScaleRangeErr	Scale bounds is out of range	Optional
OMX_StsSizeErr	One or more ROI size fields has negative or zero value	Optional
OMX_StsStepErr	Step value is less or equal zero	Optional
OMX_StsAacPrgNumErr	AAC: Invalid number of elements for one program	Mandatory
OMX_StsAacCoefValErr	AAC: Invalid quantized coefficient value	Mandatory
OMX_StsAacMaxSfbErr	AAC: Invalid coefficient index	Mandatory
OMX_StsAacTnsNumFiltErr	AAC: Invalid number of TNS filters	Optional
OMX_StsAacTnsLenErr	AAC: Invalid TNS region length	Optional
OMX_StsAacTnsOrderErr	AAC: Invalid order of TNS filter	Optional
OMX_StsAacTnsCoefResErr	AAC: Invalid bit-resolution for TNS filter coefficients	Optional
OMX_StsAacTnsCoefErr	AAC: Invalid TNS filter coefficients	Optional
OMX_StsAacTnsDirectErr	AAC: Invalid TNS filter direction	Optional
OMX_StsJPEGMarkerWarn	JPEG marker encountered; Huffman decoding operation terminted early	Optional
OMX_StsRangeErr	Bad values of bounds: the lower bound is greater than the upper bound	Mandatory
OMX_StsErr	Unknown/unspecified error	Mandatory
OMX_StsMaximumEnumeration	Placeholder, forces 32-bit enum	X

When an error occurs, function execution is interrupted and control is returned to the caller. The status codes ending with “Err,” except for the `OMX_StsNoErr` status, indicate an error. When a warning condition occurs, execution is completed, and the warning status code is returned.

1.6.9 Implementation-Dependent Data Structures

The OpenMAX DL API provides a facility (void *) to represent vendor-specific information that may be implementation-dependent. For example, the `OMXFFTSpec_C_SC32` structure might be used to store twiddle factors and bit reversal indices that are needed to compute the fast Fourier transform. The contents of implementation-dependent data structures are not defined in public header files.

1.7 Implementation Methodologies

There are three implementation methodologies associated with OpenMAX DL: synchronous, asynchronous, and integration (each of which maps to conformance profiles as outlined in the DL conformance document). The synchronous methodology is on a DL API basis as specified in this document. The asynchronous and integration methodologies are defined in Chapter 7, "Concurrency Mechanisms." The methodologies are not defined in public header files.

1.8 Accuracy Criteria

The implementation accuracy and conformance criteria set forth in the OpenMAX DL 1.0 Adopter's Package Conformance Test Specification shall supercede numerical accuracy criteria specified directly in this document (OpenMAX DL 1.0 API Specification) or incoproated by reference to another document.

2.0 Signal Processing

2

This section describes the functions and data structures that comprise the OpenMAX DL signal processing domain (omxSP) API. It includes functions for digital filtering, discrete transforms, and vector manipulation.

2.1 Data Structures

The following vendor-specific data structures are defined for the omxSP domain:

- OMXFFTSpec_C_SC16
- OMXFFTSpec_C_SC32
- OMXFFTSpec_R_S16S32
- OMXFFTSpec_R_S32

The contents of vendor-specific data structures may be implementation-dependent.

2.2 Functions

2.2.1 Vector Manipulation

2.2.1.1 Block Copy

2.2.1.1.1 Copy_16s

Prototype

```
OMXResult omxSP_Copy_S16(const OMX_S16 * pSrc, OMX_S16 * pDst, OMX_INT len);
```

Description

Copies the `len` elements of the vector pointed to by `pSrc` into the len elements of the vector pointed to by `pDst`. That is:

`pDst[i] = pSrc[i], i=0, 1, …, len-1`

Input Arguments

- `pSrc` – pointer to the source vector
- `len` – number of elements contained in the source and destination vectors

Output Arguments

- `pDst` – pointer to the destination vector

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.2 Vector Arithmetic

2.2.2.1 Dot Product

2.2.2.1.1 DotProd_S16

2.2.2.1.2 DotProd_S16_Sfs

Prototype

```
OMX_S32 omxSP_DotProd_S16 (const OMX_S16 * pSrc1, const OMX_S16 * pSrc2,
    OMX_INT len);
OMX_S32 omxSP_DotProd_S16_Sfs (const OMX_S16 * pSrc1, const OMX_S16 * pSrc2,
    OMX_INT len, OMX_INT scaleFactor);
```

Description

`omxSP_DotProd_S16`

Calculates the dot product of two input signals. This function does not perform scaling.

`omxSP_DotProd_S16_Sfs`

Calculates the dot product of the two input signals with output scaling, i.e., the result is multiplied by two to the power of the negative (-)scalefactor prior to return. The internal accumulator width must be at least 32 bits, and therefore the result is saturated prior to the scaling operation.

2 ***Note:*** *These functions return the actual result rather than the standard OMXError.*

Input Arguments

- `pSrc1` – Pointer to the first input signal buffer
- `pSrc2` – Pointer to the second input signal buffer
- `len` – Length of the signals in pSrc and pDst
- `scaleFactor` – Integer scaling factor

Returns

- The dot product result

2.2.2.2 Block Exponent

2.2.2.2.1 BlockExp_S16

2.2.2.2.2 BlockExp_S32

Prototype

```
OMX_S32 omxSP_BlockExp_S16 (const OMX_S16 * pSrc, OMX_INT len);
OMX_S32 omxSP_BlockExp_S32 (const OMX_S32 * pSrc, OMX_INT len);
```

Description

Block exponent calculation for 16-bit and 32-bit signals (count leading sign bits). These functions compute the number of extra sign bits of all values in the 16-bit and 32-bit input vector `pSrc` and return the minimum sign bit count. This is also the maximum shift value that could be used in scaling the block of data.

---

2 ***Note:*** *These functions differs from other DL functions by not returning the standard OMXError but the actual result.*

Input Arguments

- `pSrc` – pointer to the input vector
- `len` – number of elements contained in the input and output vectors (0 < len < 65536)

Output Arguments

- none

Return

- Maximum exponent that may be used in scaling

2.2.3 Filtering

This section defines functions for digital filtering. Supported filter types include the following:

- Finite Impulse Response (FIR)
- Infinite Impulse Response (IIR)
- Biquad IIR

For simplicity and consistency, the mathematical expressions in this section that describe the behavior of each filter function represent the case of the non-in-place and non-scaled variable (the default version). The behavior of any scaled and/or in-place variables can be best understood by applying to the default behavioral specification the generic in-place and scaled function behavioral rules that are given in section 1.6, “API Conventions.” Moreover, several of the filters described in this section make use of the Qm.n integer fixed-point representation of floating-point parameters. A detailed description of the Qm.n format are given in section 1.6, “API Conventions.”

2.2.3.1 FIR Filters

This section describes the FIR filtering functions, including block- and single-sample instantiations. An FIR filter is a discrete-time linear system for which the value of the current output sample can be determined by computing a weighted sum of the current and past input samples. In particular, the operation of an FIR filter can be described in terms of the time-domain difference equation:

$$y(n) = \sum_{k=0}^{K} b_k x(n-k)$$

where $x(n)$ is the input sequence, $y(n)$ is the output sequence, b_k are the filter coefficients (called “taps”), K is the filter order, and n is the discrete-time (sample) index. For the omxSP functions that implement FIR filtering, the floating point filter coefficients, b_K, are represented using Q15 parameters, such that

$$\texttt{pTapsQ15(k)} = b_K \cdot 32768,\ 0 \le k < \texttt{tapsLen}$$

Because the underlying type is `OMX_S16`, the filter coefficients must be normalized so that |bk|≤1 prior to the Q0.15 scaling. In addition to Q0.15 coefficient representations, the block- and single-sample FIR functions require external state buffers (filter memories).

2.2.3.1.1 FIR_Direct_S16

Prototype

```
OMXResult omxSP_FIR_Direct_S16(const OMX_S16 * pSrc, OMX_S16 * pDst, OMX_INT
    sampLen, const OMX_S16 * pTapsQ15, OMX_INT tapsLen, OMX_S16 *
    pDelayLine, OMX_INT * pDelayLineIndex);
OMXResult omxSP_FIR_Direct_S16_I(OMX_S16 * pSrcDst, OMX_INT sampLen, const
    OMX_S16 * pTapsQ15, OMX_INT tapsLen, OMX_S16 * pDelayLine, OMX_INT *
    pDelayLineIndex);
OMXResult omxSP_FIR_Direct_S16_Sfs(const OMX_S16 * pSrc, OMX_S16 * pDst,
    OMX_INT sampLen, const OMX_S16 * pTapsQ15, OMX_INT tapsLen, OMX_S16 *
    pDelayLine, OMX_INT * pDelayLineIndex, OMX_INT scaleFactor);
OMXResult omxSP_FIR_Direct_S16_ISfs(OMX_S16 * pSrcDst, OMX_INT sampLen,
    const OMX_S16 * pTapsQ15, OMX_INT tapsLen, OMX_S16 * pDelayLine, OMX_INT
    * pDelayLineIndex, OMX_INT scaleFactor);
```

Description

`omxSP_FIR_Direct_S16` and `omxSP_FIR_Direct_S16_I`

Block FIR filtering for 16-bit data type. This function applies the FIR filter defined by the coefficient vector `pTapsQ15` to a vector of input data.The output will saturate to 0x8000 (-32768) for a negative overflow or 0x7fff (32767) for a positive overflow.

omxSP_FIR_Direct_S16_Sfs and omxSP_FIR_Direct_S16_Isfs

Block FIR filtering for 16-bit data type. This function applies the FIR filter defined by the coefficient vector pTapsQ15 to a vector of input data. The output is multiplied by 2 to the negative power of scalefactor (i.e., 2^-scalefactor) before returning to the user.

Input Arguments

- pSrc, pSrcDst – pointer to the vector of input samples to which the filter is applied
- sampLen – the number of samples contained in the input and output vectors
- pTapsQ15 – pointer to the vector that contains the filter coefficients, represented in Q0.15 format (see "Introduction"). Given that -32768≤pTapsQ15(k)<32768, 0≤k<tapsLen, the range on the actual filter coefficients is $-1 \leq b_K < 1$, and therefore coefficient normalization may be required during the filter design process.
- tapsLen – the number of taps, or, equivalently, the filter order + 1
- pDelayLine – pointer to the 2·tapsLen -element filter memory buffer (state). The user is responsible for allocation, initialization, and de-allocation. The filter memory elements are initialized to zero in most applications.
- pDelayLineIndex – pointer to the filter memory index that is maintained internally by the function. The user should initialize the value of this index to zero.
- scaleFactor – saturation fixed scalefactor (only for the scaled function).

Output Arguments

- pDst, pSrcDst – pointer to the vector of filtered output samples

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments

2.2.3.1.2 FIROne_Direct_S16

Prototype

```
OMXResult omxSP_FIROne_Direct_S16(OMX_S16 val, OMX_S16 * pResult, const
   OMX_S16 * pTapsQ15, OMX_INT tapsLen, OMX_S16 * pDelayLine, OMX_INT *
   pDelayLineIndex);
OMXResult omxSP_FIROne_Direct_S16_I(OMX_S16 * pValResult, const OMX_S16 *
   pTapsQ15, OMX_INT tapsLen, OMX_S16 * pDelayLine, OMX_INT *
   pDelayLineIndex);
OMXResult omxSP_FIROne_Direct_S16_Sfs(OMX_S16 val, OMX_S16 * pResult, const
   OMX_S16 * pTapsQ15, OMX_INT tapsLen, OMX_S16 * pDelayLine, OMX_INT *
   pDelayLineIndex, OMX_INT scaleFactor);
OMXResult omxSP_FIROne_Direct_S16_ISfs(OMX_S16 * pValResult, const OMX_S16 *
   pTapsQ15, OMX_INT tapsLen, OMX_S16 * pDelayLine, OMX_INT *
   pDelayLineIndex, OMX_INT scaleFactor);
```

Description

`omxSP_FIROne_Direct_S16` and `omxSP_FIROne_Direct_S16_I`: Single-sample FIR filtering for 16-bit data type. These functions apply the FIR filter defined by the coefficient vector pTapsQ15 to a single sample of input data. The output saturates to 0x8000 (-32768) for a negative overflow and 0x7fff (32767) for a positive overflow.

`omxSP_FIROne_Direct_S16_Sfs` and `omxSP_FIROne_Direct_S16_ISfs`: Single-sample FIR filtering for 16-bit data type. These functions apply the FIR filter defined by the coefficient vector pTapsQ15 to a single sample of input data. The output is multiplied by 2 to the negative power of scalefactor (i.e., 2^-scalefactor) before returning to the user.

Input Arguments

- `val, pValResult` – the single input sample to which the filter is applied. A pointer is used for the in-place version.
- `pTapsQ15` – pointer to the vector that contains the filter coefficients, represented in Q0.15 format (see section 1.6, "API Conventions"). Given that `-32768≤pTapsQ15(k)<32768, 0≤k<tapsLen`, the range on the actual filter coefficients is $-1 \le b_K < 1$, and therefore coefficient normalization may be required during the filter design process.
- `tapsLen` – the number of taps, or, equivalently, the filter order + 1
- `pDelayLine` – pointer to the 2·`tapsLen` -element filter memory buffer (state). The user is responsible for allocation, initialization, and de-allocation. The filter memory elements are initialized to zero in most applications.
- `pDelayLineIndex` – pointer to the filter memory index that is maintained internally by the function. The user should initialize the value of this index to zero.
- `scaleFactor` – saturation fixed scaleFactor (only for the scaled function)

Output Arguments

- `pResult, pValResult` – pointer to the filtered output sample

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.3.2 IIR Filters

This section describes the IIR filtering functions, including block and single sample variants. An IIR filter is a discrete-time linear system for which the value of the current output sample can be determined by computing a weighted sum of the current input sample, past input samples, and past output samples. In particular, the operation of an IIR filter can be described in terms of the time-domain difference equation, i.e.,

$$y(n) = \sum_{k=0}^{K} b_k x(n-k) - \sum_{m=1}^{M} a_m y(n-m)$$

where $x(n)$ is the input sequence, $y(n)$ is the output sequence, n is the discrete-time (sample) index, and a_m and b_K are the filter coefficients (aka "taps"). For the omxSP IIR filter implementations, the floating-point filter coefficients b_k and a_m are represented in a combined coefficient vector that is referenced by the parameter `pTaps` and is organized as follows:

pTaps					pTaps+*L*+*1*				pTaps+2*L*+1
b_0	b_1	b_2	…	b_L	sf	a_1	a_2	…	a_L

The combined coefficient vector contains $2L+2$ elements, where $L=max\{K,M)$. Therefore, if $K \;\; M$ for a particular filter design, the user must pad with zeros the smaller set of coefficients such that the organization of the combined coefficient vector matches the figure. For example, if $K=2$ and $M=3$, then the combined coefficient vector would be arranged as follows:

pTaps				pTaps+*L*+*1*			pTaps+2*L*+1
b_0	b_1	b_2	0	sf	a_1	a_2	a_3

The specific Q-format used to represent the elements of the IIR coefficient vector is controlled by the scaling coefficient denoted in the above figure by sf. In particular, the actual filter coefficients are related to the elements of the coefficient vector in the following way:

$$b_{\mathrm{k}}=pTaps(k)\ 2^{-sf},\ 0\ \ k\ \ K$$

and

$$a_{\mathrm{m}}=pTaps(m+L+2)\cdot 2^{-sf},\ 0\ \ m\ \ M$$

where

$$sf = pTaps(L+1),\ \text{and}\ sf\ \ 0.$$

2.2.3.2.1 IIR_Direct_S16

Prototype

```
OMXResult omxSP_IIR_Direct_S16(const OMX_S16 * pSrc, OMX_S16 * pDst, OMX_INT
    len, const OMX_S16 * pTaps, OMX_INT order, OMX_S32 * pDelayLine);
OMXResult omxSP_IIR_Direct_S16_I(OMX_S16 * pSrcDst, OMX_INT len, const
    OMX_S16 * pTaps, OMX_INT order, OMX_S32 * pDelayLine);
```

Description

Block IIR filtering for 16-bit data. This function applies the direct form II IIR filter defined by the coefficient vector `pTaps` to a vector of input data. The output will saturate to 0x8000 (-32768) for a negative overflow or 0x7fff (32767) for a positive overflow.

Input Arguments

- `pSrc, pSrcDst` – pointer to the vector of input samples to which the filter is applied
- `len` – the number of samples contained in both the input and output vectors
- `pTaps` – pointer to the *2L+2*-element vector that contains the combined numerator and denominator filter coefficients from the system transfer function, *H(z)*. Coefficient scaling and coefficient vector organization should follow the conventions described above. The value of the coefficient scaleFactor exponent must be non-negative (sf 0).
- `order` – the maximum of the degrees of the numerator and denominator coefficient. polynomials from the system transfer function, *H(z)*, i.e.:*order=max(K,M)-1=L-1*.
- `pDelayLine` – pointer to the *L*-element filter memory buffer (state). The user is responsible for allocation, initialization, and deallocation. The filter memory elements are initialized to zero in most applications.

Output Arguments

- `pDst, pSrcDst` – pointer to the vector of filtered output samples

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.3.2.2 IIROne_Direct_S16

Prototype

```
OMXResult omxSP_IIROne_Direct_S16 (OMX_S16 val, OMX_S16 * pResult, const
    OMX_S16 * pTaps, OMX_INT order, OMX_S32 * pDelayLine);
OMXResult omxSP_IIROne_Direct_S16_I(OMX_S16 * pValResult, const OMX_S16 *
    pTaps, OMX_INT order, OMX_S32 * pDelayLine);
```

Description

Single sample IIR filtering for 16-bit data. This function applies the direct form II IIR filter defined by the coefficient vector `pTaps` to a single sample of input data. The output will saturate to 0x8000 (-32768) for a negative overflow or 0x7fff (32767) for a positive overflow.

Input Arguments

- `val, pValResult` – the single input sample to which the filter is applied. A pointer is used for the in-place version.
- `pTaps` – pointer to the *2L+2* -element vector that contains the combined numerator and denominator filter coefficients from the system transfer function, *H(z)*. Coefficient scaling and coefficient vector organization should follow the conventions described above. The value of the coefficient scaleFactor exponent must be non-negative (*sf 0*).
- `order` – the maximum of the degrees of the numerator and denominator coefficient polynomials from the system transfer function, *H(z)*. i.e.: *order=max(K,M)-1=L-1*.

- `pDelayLine` – pointer to the L-element filter memory buffer (state). The user is responsible for allocation, intialization, and deallocation. The filter memory elements are initialized to zero in most applications.

Output Arguments

- `pResult, pValResult` – pointer to the filtered output sample

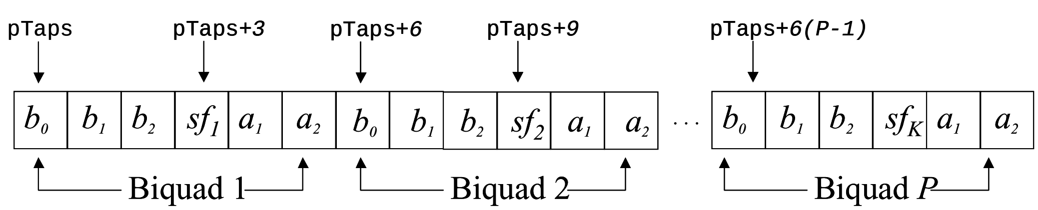
Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.3.3 Biquad IIR Filters

For the block and single-sample variables, the floating point filter coefficients b_k and a_m for all of the biquad stages are represented in a combined coefficient vector that the parameter `pTaps` points to and is organized as shown in Figure 2-1:

Figure 2-1: Combined Coefficient Vector Organization



The combined coefficient vector contains $6P$ elements, where P is the number of biquad stages in the cascade structure. As with the coefficient vector for the standard IIR function, if K M for any constituent filter, then the user must pad with zeros the smaller set of coefficients such that the organization of the combined coefficient vector matches the figure. The specific Q-format used to represent the elements of the P^{th} biquad section is controlled by the scaling coefficient denoted in the above figure by sf_p, 1 p P, where sf_p70. In particular, the actual filter coefficients for the P^{th} biquad section are related to the elements of the coefficient vector in the following way:

$$b_k = pTaps(6(p-1)+k) \cdot 2^{-sfp},\ 0 \cdot\ k\ \ k_p$$

and

$$a_m = pTaps(6(p-1)+m+4)\ ) \cdot 2^{-sfp},\ 0\ \ m\ \ M_p.$$

where $sf_p = pTaps(6(p-1)+3$, K_p is the order of the P^{th} biquad numerator polynomial, and M_p is the order of the P^{th} biquad denominator polynomial.

2.2.3.3.1 IIR_BiQuadDirect_S16

Prototype

```
OMXResult omxSP_IIR_BiQuadDirect_S16(const OMX_S16 * pSrc, OMX_S16 * pDst,
    OMX_INT len, const OMX_S16 * pTaps, OMX_INT numBiquad, OMX_S32 *
    pDelayLine);
OMXResult omxSP_IIR_BiQuadDirect_S16_I(OMX_S16 * pSrcDst, OMX_INT len, const
    OMX_S16 * pTaps, OMX_INT numBiquad, OMX_S32 * pDelayLine);
```

Description

Block biquad IIR filtering for 16-bit data type. This function applies the direct form II biquad IIR cascade defined by the coefficient vector `pTaps` to a vector of input data.The output will saturate to 0x8000 (-32768) for a negative overflow or 0x7fff (32767) for a positive overflow.

Input Arguments

- `pSrc, pSrcDst` – pointer to the vector of input samples to which the filter is applied
- `len` – the number of samples contained in both the input and output vectors
- `pTaps` – pointer to the *6P* -element vector that contains the combined numerator and denominator filter coefficients from the biquad cascade. Coefficient scaling and coefficient vector organization should follow the conventions described above. The value of the coefficient `scaleFactor` exponent must be non-negative. (*sfp 0*).
- `numBiquad` – the number of biquads contained in the IIR filter cascade: (*P*)
- `pDelayLine` – pointer to the *2P* -element filter memory buffer (state). The user is responsible for allocation, intialization, and de-allocation. The filter memory elements are initialized to zero in most applications.

Output Arguments

- `pDst, pSrcDst` – pointer to the vector of filtered output samples

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.3.3.2 IIROne_BiQuad_S16

Prototype

```
OMXResult omxSP_IIROne_BiQuadDirect_S16(OMX_S16 val, OMX_S16 * pResult,
    const OMX_S16 * pTaps, OMX_INT numBiquad, OMX_S32 * pDelayLine);
OMXResult omxSP_IIROne_BiQuadDirect_S16_I(OMX_S16 * pValResult, const
    OMX_S16 * pTaps, OMX_INT numBiquad, OMX_S32 * pDelayLine);
```

Description

Single-sample biquad IIR filtering for 16-bit data type. This function applies the direct form II biquad IIR cascade defined by the coefficient vector `pTaps` to a single sample of input data.The output will saturate to 0x8000 (-32768) for a negative overflow or 0x7fff (32767) for a positive overflow.

Input Arguments

- `val, pValResult` – the single input sample to which the filter is applied. A pointer is used for the in-place version.
- `pTaps` – pointer to the 6P-element vector that contains the combined numerator and denominator filter coefficients from the biquad cascade. Coefficient scaling and coefficient vector organization should follow the conventions described above. The value of the coefficient scalefactor exponent must be non-negative: (sfp70).
- `numBiquad` – the number of biquads contained in the IIR filter cascade: (P)
- `pDelayLine` – pointer to the 2p-element filter memory buffer (state). The user is responsible for allocation, intialization, and deallocation. The filter memory elements are initialized to zero in most applications.

Output Arguments

- `pResult, pValResult` – pointer to the filtered output sample

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.3.4 Median Filters

2.2.3.4.1 FilterMedian_S32

Prototype

```
OMXRESULT omxSP_FilterMedian_S32(const OMX_S32 *pSrc, OMX_S32 *pDst, OMX_INT
    len, OMX_INT maskSize);
OMXResult omxSP_FilterMedian_S32_I(OMX_S32 *pSrcDst, OMX_INT len, OMX_INT
    maskSize);
```

Description

This function computes the median over the region specified by the median mask for the every element of the input array. The median outputs are stored in the corresponding elements of the output vector.

Input Arguments

- `pSrc` – pointer to the input vector
- `pSrcDst` – pointer to the input vector
- `len` – number of elements contained in the input and output vectors (0 < len < 65536)
- `maskSize` – median mask size; if an even value is specified, the function subtracts 1 and uses the odd value of the filter mask for median filtering (0 < maskSize < 256)

Output Arguments

- `pDst` – pointer to the median-filtered output vector
- `pSrcDst` – pointer to the median-filtered output vector

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

2.2.3.5 Filtering Usage Examples

This section provides 'C' language source listings that illustrate the usage of the FIR, IIR, and biquad IIR filtering functions.

2.2.3.5.1 FIR Filter Example

Example 2-1: omxSP_FIR_Direct_S16_Sfs Usage

This example illustrates the usage of the scaled FIR filtering function, `FIR_Direct_S16_Sfs`*. The example code implements a linear-phase, lowpass, 19th order FIR filter having the coefficient:*

$$b_k = \{.08,\ 0.10492407, 0,\ 0.17699537,\ 0.28840385, \ldots,\ 0.08\}$$

Given that

$$\sum_{k=0}^{19} |b_k| = 10.34$$

the dynamic range on the output, y(n)*, is* -10.34 y(n)<10.34*, which means that a Q4.11 output representation is required to avoid saturation in the 16-bit output word. Therefore, the scaled FIR function is used to accommodate the dynamic range on the output with a scalefactor value of 4.*

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "omxSP.h"
#define tapsLen 20
#define N 40
#define scaleFactor 4
```

```
OMX_INT main()
{
 OMX_INT     i;
 OMX_S16  pSrc[N], pDst[N];
 OMX_S32  pDelayLine[tapsLen*2];
 OMX_INT     delayLineIndex;
 OMX_S16  pTaps[tapsLen];
 float b[tapsLen] =
 { 0.080000000, 0.104924069, 0.176995366, 0.288403847, 0.427076676,
 0.577986499, 0.724779895, 0.851549523, 0.944557926, 0.993726200,
 0.993726200, 0.944557926, 0.851549523, 0.724779895, 0.577986499,
 0.427076676, 0.288403847, 0.176995366, 0.104924069, 0.080000000 };

 /* scale the filter taps to Q15 */
 for ( i = 0; i < tapsLen; i ++ ) {
 b[i] *= (1<<14);
 pTaps[i] = (b[i] > 32767)?32767 : b[i];
 }
 /* random input signal */
 srand(200);
 for ( i = 0; i < N; i ++ )

 pSrc[i] = rand()-32768/2;
 delayLineIndex = 0;
 for ( i = 0; i < tapsLen*2; i ++ )
 pDelayLine[i] = 0;
 omxSP_FIR_Direct_S16_Sfs(pSrc, pDst, N, pTaps, tapsLen, pDelayLine,
     &delayLineIndex, scaleFactor);
 /* display out signal vector */
 for ( i = 0; i < N; i ++ ) {
 printf("%8d", pDst[i]);
 if ( (i+1)%5 == 0 ) {
       printf("\n");
 }
 }
 return(0);
}
```

2.2.3.5.2 IIR

The example below illustrates the usage of the `omxSP_IIR_Direct_S16` function.

Example 2-2: omxSP_IIR_Direct_S16 Usage

```
#include <stdio.h>
#include "omxSP.h"
#define tapsLen 4
#define N 40

OMX_INT main()
{
   OMX_INT    i;
 OMX_S16 pSrc[N], pDst[N];

 /* here, the scaleFactor is 15 */
 OMX_S16 pTapsIIR[(tapsLen+1)*2] = {
  7922, 16348, 22394, 16348,  7922,    15,
  6338, 29356,  1841,  4222
 };
 OMX_S32 pDelayLineIIR[tapsLen];

 for ( i = 0; i < tapsLen; i ++ ) {
    pDelayLineIIR[i] = 0;
 }

 printf("\nTesting <omxSP_IIR_Direct_S16>: \n");
 for ( i = 0; i < N; i ++ ) {
```

```
        pSrc[i] = i;
    }

    omxSP_IIR_Direct_S16(pSrc, pDst, N, pTapsIIR,
    tapsLen, pDelayLineIIR);
    for ( i = 0; i < N; i ++ ) {
        printf("%8d", pDst[i]);
        if ( i%5 == 0 && i != 0 ) {
            printf("\n");
        }
    }
    return(0);
}
#undef N
#undef tapsLen
```

2.2.3.5.3 Biquad IIR

The example below illustrates the usage of the `omxSP_IIROne_BiQuadDirect_S16_I` function.

Example 2-3: omxSP_IIROne_BiQuadDirect_S16_I Usage

```
#include <stdio.h>
#include "omxSP.h"
#define numBiQuad 4
#define N 40

OMX_INT main()
{
     OMX_INT    i;

    OMX_S16 pValResult;
 OMX_S16 pTapsIIR[numBiQuad*6] = {
        3178,  4488,  3178, 14,  -922,  1766,
        7569,  2155,  7569, 14,  -225,  7572,
        9458,   513,  9458, 14,    11,  9542,
```

```
        9895,    159,   9895, 14,     55,   9934
    };

    OMX_S32 pDelayLineIIR[numBiQuad*2];

    for ( i = 0; i < numBiQuad*2; i ++ ) {
            pDelayLineIIR[i] = 0;
    }
    printf("\nTesting <omxSP_IIROne_BiQuadDirect_S16_I>:\n");
    for ( i = 0; i < N; i ++ ) {
            pValResult = i;
            omxSP_IIROne_BiQuadDirect_S16_I(&pValResult, pTapsIIR,
                 numBiQuad, pDelayLineIIR);
            printf("%8d", pValResult);



            if ( i != 0 && i%5 == 0 ) {
               printf("\n");
        }
    }

    return(0);
}
#undef numBiQuad
#undef N
```

2.2.4 FFT

This section describes functions for computing variable-length FFTs, including:

- Forward and Inverse FFT for complex-valued sequences ("CToC")
- Forward and Inverse FFT for real-valued input sequences ("RToCCR, CCRToR")

The FFT functions support radix-2 block lengths of 2^N for 0 <= N <= 12. Helper functions are provided to initialize length-dependent specification structures that are required for each FFT. Example programs are provided to illustrate calling conventions.

2.2.4.1 FFT Helper Functions

2.2.4.1.1 FFTInit_C_SC16

2.2.4.1.2 FFTInit_C_SC32

Prototype

```
OMXResult omxSP_FFTInit_C_SC16(OMXFFTSpec_C_SC16* pFFTSpec, OMX_INT order);
OMXResult omxSP_FFTInit_C_SC32(OMXFFTSpec_C_SC32* pFFTSpec, OMX_INT order);
```

Description

These functions initialize the specification structures required for the complex FFT and IFFT functions. Desired block length is specified as an input.

- The function <FFTInit_C_SC16> is used to initialize the specification structures for functions <FFTFwd_CToC_SC16_Sfs> and <FFTInv_CToC_SC16_Sfs>
- The function <FFTInit_C_SC32> is used to initialize the specification structures for the functions <FFTFwd_CToC_SC32_Sfs> and <FFTInv_CToC_SC32_Sfs>

Memory for the specification structure *pFFTSpec must be allocated prior to calling these functions and should be 4-byte aligned for omxSP_FFTInit_C_SC16 and 8-byte aligned for omxSP_FFTInit_C_SC32. The space required for *pFFTSpec, in bytes, can be determined using <FFTGetBufSize_C_SC16> and <FFTGetBufSize_C_SC32>, respectively, for the 16-bit and 32-bit functions.

Input Arguments

- order – base-2 logarithm of the desired block length; valid in the range [0,12]

Output Arguments

- pFFTSpec – pointer to initialized specification structure

Return

- OMX_StsNoErr - no error
- OMX_StsBadArgErr – bad arguments

The function returns OMX_StsBadArgErr if one or more of the following is true:

- The pointer pFFTSpec is NULL
- order <0 or order > 12

2.2.4.1.3 FFTInit_R_S16S32

2.2.4.1.4 FFTInit_R_S32

Prototype

```
OMXResult omxSP_FFTInit_R_S16S32(OMXFFTSpec_R_S16S32* pFFTFwdSpec, OMX_INT
   order);
OMXResult omxSP_FFTInit_R_S32(OMXFFTSpec_R_S32* pFFTFwdSpec, OMX_INT order);
```

Description

These functions initialize specification structures required for the real FFT and IFFT functions.

- The function `<FFTInit_R_S16S32>` is used to initialize the specification structures for functions `<FFTFwd_RToCCS_S16S32_Sfs>` and `<FFTInv_CCSToR_S32S16_Sfs>`.
- The function <FFTInit_R_S32> is used to initialize the specification structures for functions `<FFTFwd_RToCCS_S32_Sfs>` and `<FFTInv_CCSToR_S32_Sfs>`.

Memory for `*pFFTFwdSpec` must be allocated before calling these function and should be 8-byte aligned. The number of bytes required for `*pFFTFwdSpec` can be determined using `<FFTGetBufSize_R_S16S32>` and `<FFTGetBufSize_R_S32>`, respectively, for the 16-bit and 32-bit functions.

Input Arguments

- `order` – base-2 logarithm of the desired block length; valid in the range [0,12]

Output Arguments

- `pFFTFwdSpec` – pointer to the initialized specification structure

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- The pointer `pFFTFwdSpec` is NULL
- `order` <0 or `order` > 12

2.2.4.1.5 FFTGetBufSize_C_SC16

2.2.4.1.6 FFTGetBufSize_C_SC32

Prototype

```
OMXResult omxSP_FFTGetBufSize_C_SC16(OMX_INT order, OMX_INT *pSize);
OMXResult omxSP_FFTGetBufSize_C_SC32(OMX_INT order, OMX_INT *pSize);
```

Description

These functions compute the size of the specification structure required for the length 2^{order} complex FFT and IFFT functions.

- The function <`FFTGetBufSize_C_SC16`> is used in conjunction with the 16-bit functions <`FFTFwd_CToC_SC16_Sfs`> and <`FFTInv_CToC_SC16_Sfs`>.
- The function <`FFTGetBufSize_C_SC32`> is used in conjunction with the 32-bit functions <`FFTFwd_CToC_SC32_Sfs`> and <`FFTInv_CToC_SC32_Sfs`>.

Input Arguments

- `order` – base-2 logarithm of the desired block length; valid in the range [0,12]

Output Arguments

- `pSize` – pointer to the number of bytes required for the specification structure

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `pSize` is NULL
- `order` <0 or `order` > 12

2.2.4.1.7 FFTGetBufSize_R_S16S32

2.2.4.1.8 FFTGetBufSize_R_S32

Prototype

```
OMXResult omxSP_FFTGetBufSize_R_S16S32(OMX_INT order, OMX_INT *pSize);
OMXResult omxSP_FFTGetBufSize_R_S32(OMX_INT order, OMX_INT *pSize);
```

Description

These functions compute the size of the specification structure required for the length 2^{order} real FFT and IFFT functions.

- The function <`FFTGetBufSize_R_S16S32`> is used in conjunction with the 16-bit functions <`FFTFwd_RToCCS_S16S32_Sfs`> and <`FFTInv_CCSToR_S32S16_Sfs`>.
- The function <`FFTGetBufSize_R_S32`> is used in conjunction with the 32-bit functions <`FFTFwd_RToCCS_S32_Sfs`> and <`FFTInv_CCSToR_S32_Sfs`>.

Input Arguments

- `order` – base-2 logarithm of the length; valid in the range [0,12]

Output Arguments

- `pSize` – pointer to the number of bytes required for the specification structure

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `pSize` is NULL
- `order` <0 or `order` > 12

2.2.4.2 FFT for Complex-Valued Signals

2.2.4.2.1 FFTFwd_CToC_SC16_Sfs

2.2.4.2.2 FFTFwd_CToC_SC32_Sfs

Prototype

```
OMXResult omxSP_FFTFwd_CToC_SC16_Sfs(const OMX_SC16 *pSrc, OMX_SC16 *pDst,
    const OMXFFTSpec_C_SC16 *pFFTSpec, OMX_INT scaleFactor);
OMXResult omxSP_FFTFwd_CToC_SC32_Sfs(const OMX_SC32 *pSrc, OMX_SC32 *pDst,
    const OMXFFTSpec_C_SC32 *pFFTSpec, OMX_INT scaleFactor);
```

Description

Compute an FFT for a complex signal of length of 2^{order}, where 0 <= order <= 12. Transform length is determined by the specification structure, which must be initialized prior to calling the FFT function using the appropriate helper, i.e., <FFTInit_C_sc32> or <FFTInit_C_SC16>. The relationship between the input and output sequences can be expressed in terms of the DFT, i.e.,

$$X[k] = 2^{-scaleFactor} \sum_{n=0}^{2^{order}-1} x[n] e^{-j\frac{2\pi}{2^{order}}nk}, k = 0,1,\ldots,2^{order}-1 (N = 2^{order})$$

Input Arguments

- `pSrc` – pointer to the input signal, a complex-valued vector of length 2^{order}
- `pFFTSpec` – pointer to the preallocated and initialized specification structure
- `scaleFactor` – output scale factor; the range for <omxSP_FFTFwd_CToC_SC16_Sfs> is [0,16], and for <omxSP_FFTFwd_CToC_SC32_Sfs> the range is [0,32]

Output Arguments

- `pDst` – pointer to the complex-valued output vector, of length 2^{order}

Return

- `OMX_StsNoErr` – no error

- `OMX_StsBadArgErr` – Bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- If `pSrc`, `pDst`, `pFFTSpec`, or `pBuffer` is NULL for `<omxSP_FFTFwd_CToC_SC16_Sfs>`
- If `pSrc`, `pDst`, or `pFFTSpec` is NULL for `<omxSP_FFTFwd_CToC_SC32_Sfs>`
- `pSrc` or `pDst` is not 8-byte aligned
- `order` <0 or `order` > 12
- `scaleFactor`<0 or `scaleFactor` >16 for `<omxSP_FFTFwd_CToC_SC16_Sfs>`
- `scaleFactor`<0 or `scaleFactor` >32 for `<omxSP_FFTFwd_CToC_SC32_Sfs>`

2.2.4.2.3 FFTInv_CToC_SC16_Sfs

2.2.4.2.4 FFTInv_CToC_SC32_Sfs

Prototype

```
OMXResult omxSP_FFTInv_CToC_SC16_Sfs(const OMX_SC16 *pSrc, OMX_SC16 *pDst,
    const OMXFFTSpec_C_SC16 *pFFTSpec, OMX_INT scaleFactor);
OMXResult omxSP_FFTInv_CToC_SC32_Sfs(const OMX_SC32 *pSrc,OMX_SC32 *pDst,
    const OMXFFTSpec_C_SC32 *pFFTSpec, OMX_INT scaleFactor);
```

Description

These functions compute an inverse FFT for a complex signal of length of 2^{order}, where 0 <= order <= 12. Transform length is determined by the specification structure, which must be initialized prior to calling the FFT function using the appropriate helper, i.e., <FFTInit_C_sc32> or <FFTInit_C_SC16>. The relationship between the input and output sequences can be expressed in terms of the IDFT, i.e.,:

$$x[n] = \frac{2^{-\text{scaleFactor}}}{2^{order}} \cdot \sum_{k=0}^{2^{order}-1} X[k] \cdot e^{j\frac{2x}{2^{order}}nk}, n = 1,2,\ldots 2^{order}-1.(N = 2^{order})$$

Input Arguments

- `pSrc` – pointer to the complex-valued input signal, of length 2^{order}
- `pFFTSpec` – pointer to the preallocated and initialized specification structure
- `scaleFactor` – scale factor of the output. Valid range for `<omxSP_FFTInv_CToC_SC16_Sfs>` is [0,16] and for `<omxSP_FFTInv_CToC_SC32_Sfs>` is [0,32].

Output Arguments

- `pDst` – pointer to the complex-valued output signal, of length 2^{order}

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- One of the pointers `pSrc`, `pDst`, `pFFTSpec` is NULL for <omxSP_FFTInv_CToC_SC16_Sfs>
- One of the pointers `pSrc`, `pDst`, `pFFTSpec` is NULL for <omxSP_FFTInv_CToC_SC32_Sfs>
- `pSrc` or `pDst` is not aligned on an 8-byte boundary
- `order` <0 or `order` > 12
- `scaleFactor`<0 or `scaleFactor` >16 for <omxSP_FFTInv_CToC_SC16_Sfs>
- `scaleFactor`<0 or `scaleFactor` >32 for <omxSP_FFTInv_CToC_SC32_Sfs>

2.2.4.3 Example, FFT for Complex-Valued Signals

Example 2-4: Complex-Valued FFT Usage

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "OMXSP.h"

OMXResult FFT_ctoc_example();

OMX_INT main()
{
   FFT_ctoc_example();
}

OMXResult FFT_ctoc_example()
{
    OMX_SC16 x[8], y[8], z[8];
    OMX_INT n, bufSize;
    OMXResult status;
    OMXFFTSpec_C_SC16* pFwdSpec = NULL;
    OMXFFTSpec_C_SC16* pInvSpec = NULL;

    srand( (unsigned)time( NULL ) );

    for(n=0; n<8; n++)  {
        x[n].re = (OMX_S16)((rand()%1024)-512);
        x[n].im = (OMX_S16)((rand()%1024)-512);
```

```
    }

    status = omxSP_FFTGetBufSize_C_SC16(3, &bufSize);
    pFwdSpec = (OMX_U8*)malloc(bufSize);
    pInvSpec = (OMX_U8*)malloc(bufSize);
    status = omxSP_FFTInit_C_SC16( pFwdSpec, 3 );
    status = omxSP_FFTInit_C_SC16( pInvSpec, 3 );

    status = omxSP_FFTFwd_CToC_SC16_Sfs ( x, y, pFwdSpec, 0 );



    status = omxSP_FFTInv_CToC_SC16_Sfs ( y, z, pInvSpec, 0 );

printf("n\tx.r\tx.i\ty.r\ty.i\tz.r\tz.i\n”);
    for(n=0; n<8; n++)  {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n, ", n, x[n].re,
x[n].im, y[n].re, y[n].im, z[n].re, z[n].im );
    }

free( pFwdSpec );
free( pInvSpec );

    return status;

}
```

2.2.4.4 FFT for Real-Valued Signals

2.2.4.4.1 FFTFwd_RToCCS_S16S32_Sfs

2.2.4.4.2 FFTFwd_RToCCS_S32_Sfs

Prototype

```
OMXResult omxSP_FFTFwd_RToCCS_S16S32_Sfs(const OMX_S16 *pSrc, OMX_S32 *pDst,
    const OMXFFTSpec_R_S16S32 *pFFTSpec, OMX_INT scaleFactor);
OMXResult omxSP_FFTFwd_RToCCS_S32_Sfs (const OMX_S32 *pSrc, OMX_S32 *pDst,
    const OMXFFTSpec_R_S32 *pFFTSpec, OMX_INT scaleFactor);
```

Description

These functions compute an FFT for a real-valued signal of length of 2^{order}, where 0 <= order <= 12. Transform length is determined by the specification structure, which must be initialized prior to calling the FFT function using the appropriate helper, i.e., <`FFTInit_R_S16S32`> or <`FFTInit_R_S32`>. The relationship between the input and output sequences can be expressed in terms of the DFT, i.e.,:

$$x[n] = \frac{2^{-scaleFactor}}{2^{order}} \sum_{k=0}^{2^{order}-1} X[k] e^{j\frac{2\pi}{2^{order}}nk}, n = 0,1,\ldots,2^{order} - 1.(N = 2^{order})$$

The conjugate-symmetric output sequence is represented using a packed RCCS vector, which is of length N+2, and is organized as follows:

Index:	0	1	2	3	4	5	. . .	N-2	N-1	N	N+1
Component	R0	0	R1	I1	R2	I2	. . .	$R_{N/2-1}$	$I_{N/2-1}$	$R_{N/2}$	0

where Rn and In, respectively, denote the real and imaginary components for FFT bin n. Bins are numbered from 0 to N/2, where N is the FFT length. Bin index 0 corresponds to the DC component, and bin index N/2 corresponds to the foldover frequency.

Input Arguments

- `pSrc` – pointer to the real-valued input sequence, of length 2^{order}
- `pFFTSpec` – pointer to the preallocated and initialized specification structure
- `scaleFactor` – output scale factor; valid range is [0, 32]

Output Arguments

- `pDst` – pointer to output sequence, represented using RCS format, of length 2^{order}+2

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- One of the pointers `pSrc`, `pDst`, `pFFTSpec` is NULL
- `pSrc`, or `pDst` is not aligned on an 8-byte boundary
- order <0 or order > 12
- `scaleFactor`<0 or `scaleFactor` >32

2.2.4.4.3 FFTInv_CCSToR_S32S16_Sfs

2.2.4.4.4 FFTInv_CCSToR_S32_Sfs

Prototype

```
OMXResult omxSP_FFTInv_CCSToR_S32S16_Sfs(const OMX_S32 *pSrc, OMX_S16 *pDst,
    const OMXFFTSpec_R_S16S32 *pFFTSpec, OMX_INT scaleFactor);
```

```
OMXResult omxSP_FFTInv_CCSToR_S32_Sfs (const OMX_S32 *pSrc, OMX_S32 *pDst,
    const OMXFFTSpec_R_S32 *pFFTSpec, OMX_INT scaleFactor);
```

Description

These functions compute the inverse FFT for a conjugate-symmetric input sequence. Transform length is determined by the specification structure, which must be initialized prior to calling the FFT function using either <FFTInit_C_sc32> or <FFTInit_C_SC16>. For a transform of length M, the input sequence is represented using a packed RCCS vector of length M+2, and is organized as follows:

Index:	0	1	2	3	4	5	. . .	M-2	M-1	M	M+1
Component	R0	0	R1	I1	R2	I2	. . .	$R_{M/2-1}$	$I_{M/2-1}$	$R_{M/2}$	0

where Rn and In, respectively, denote the real and imaginary components for FFT bin n. Bins are numbered from 0 to M/2, where M is the FFT length. Bin index 0 corresponds to the DC component, and bin index M/2 corresponds to the foldover frequency.

Input Arguments

- `pSrc` – pointer to the complex-valued input sequence represented using RCCcs format, of length $2^{order} + 2$
- `pFFTSpec` – pointer to the preallocated and initialized specification structure
- `scaleFactor` – output scalefactor; range is [0,32] for the function <omxSP_FFTInv_CCSToR_S32_Sfs>, and [0,16] for the function <omxSP_FFTInv_CCSToR_S32S16_Sfs>

Output Arguments

- `pDst` – pointer to the real-valued output sequence, of length 2^{order}

Return

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `pSrc`, `pDst`, `pFFTSpec`, or `pBuffer` is NULL
- `pSrc` or `pDst` or `pBuffer` is not aligned at 8-byte boundary
- `order` <0 or `order` > 12
- `scaleFactor`<0 or `scaleFactor` >16 for <`omxSP_FFTInv_CCSToR_S32S16_Sfs`>
- `scaleFactor`<0 or `scaleFactor` >32 for <`omxSP_FFTInv_CCSToR_S32_Sfs`>

2.2.4.5 Example, FFT for Real-Valued Signals

Example 2-5: Real-Valued FFT Usage

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "OMXSP.h"

OMXResult FFT_rtoccs_example();

OMX_INT main()
{
    FFT_rtoccs_example();
}

OMXResult FFT_rtoccs_example()
{
    OMX_S16 x[8];
    OMX_S32 y[10];
    OMX_S16 z[8];
    OMX_INT n, bufSize;
    OMXResult status;
    OMXFFTSpec_R_S16S32* pFwdSpec = NULL;
    OMXFFTSpec_R_S16S32* pInvSpec = NULL;

    srand( (unsigned)time( NULL ) );
```

```
    for(n=0; n<8; n++)  {
        x[n] = (OMX_S32)((rand()%1024)-512);
    }
    status = omxSP_FFTGetBufSize_R_S16S32( 3, &bufSize);
    pFwdSpec = (OMXFFTSpec_R_S16S32 *)malloc(bufSize);
    pInvSpec = (OMXFFTSpec_R_S16S32 *)malloc(bufSize);
    status = omxSP_FFTInit_R_S16S32( pFwdSpec, 3 );
    status = omxSP_FFTInit_R_S16S32( pInvSpec, 3 );
    status = omxSP_FFTFwd_RToCCS_S16S32_Sfs ( x, y, pFwdSpec, 0 );
    status = omxSP_FFTInv_CCSToR_S32S16_Sfs ( y, z, pInvSpec, 0 );

    printf("FFT Input\n");
    printf("n\tx[n]\n");
    for(n=0; n<7; n++)  {
        printf("%d\t%d",n,x[n]);
    }

    printf("FFT Output / IFFT Input\n");
    printf("n\ty.re[n]\ty.im[n]\n");
    for(n=0; n<=4; n++)  {
        printf("%d\t%d\t%d\n",n,y[2*n],y[2*n+1]);
    }
    printf("IFFT Output\n");
    printf("n\tz[n]\n");
    for(n=0; n<7; n++)  {
        printf("%d\t%d",n,z[n]);
    }

    free( pFwdSpec );
    free( pInvSpec );
    return status;
}
```

Audio Coding

3

This section describes the functions and data types that comprise the OpenMAX DL audio coding domain (omxAC) API, including functions that can be used to construct the MPEG-1/MPEG-2 layer 3 decoders (“MP3”), the “MPEG-2.5” decoder (omxACMP3 sub-domain), as well as functions that can be used to construct MPEG-2/MPEG-4 AAC-LC/LTP decoders (omxACAAC sub-domain).

3.1 MP3 Decoder Sub-Domain (omxACMP3)

3.1.1 Constants

Table 3-1: MP3 Macro and Constant Definitions

Global Macro's Name	Definition	Notes
OMX_MP3_GRANULE_LEN	576	The number of samples in one granule
OMX_MP3_SF_BUF_LEN	40	Scalefactor buffer length (8-bit words)
OMX_MP3_V_BUF_LEN	512	V data buffer length (32-bit words)
OMX_MP3_SFB_TABLE_LONG_LEN	138	
OMX_MP3_SFB_TABLE_SHORT_LEN	84	

3.1.2 Data Structures

3.1.2.1 Frame Header

```
typedef struct {
    OMX_INT idEx;
    OMX_INT id;     /**  IDex/ID 1/1: MPEG-1, IDex/ID: 1/0 MPEG-2, IDex/ID:
   0/0    MPEG-2.5 */
    OMX_INT layer;              /** layer index 0x3: Layer I
                                //          0x2: Layer II
                                //          0x1: Layer III */
    OMX_INT protectionBit;      /** CRC flag 0: CRC on, 1: CRC off */
    OMX_INT bitRate;            /** bit rate index */
    OMX_INT samplingFreq;       /** sampling frequency index */
    OMX_INT paddingBit;         /** padding flag 0: no padding, 1 padding
   */
    OMX_INT privateBit;         /** private_bit, no use  */
    OMX_INT mode;               /** mono/stereo select information */
```

```
    OMX_INT modeExt;            /** extension to mode */
    OMX_INT copyright;          /** copyright or not, 0: no, 1: yes  */
    OMX_INT originalCopy;       /** original bitstream or copy, 0: copy, 1:
   original */
    OMX_INT emphasis;           /** flag indicates the type of de-emphasis
   that shall be used */
    OMX_INT CRCWord;            /** CRC-check word */

} OMXMP3FrameHeader;
```

3.1.2.2 Side Information

```
typedef struct {
OMX_INT  part23Len;       /* number of main_data bits */
OMX_INT  bigVals;         /* half the number of Huffman code words whose
   maximum
                               amplitudes may be greater than 1 */
OMX_INT  globGain;        /* quantizes step size information */
OMX_INT  sfCompress;      /* number of bits used for scale factors */
OMX_INT  winSwitch;       /* window switch flag */
OMX_INT  blockType;       /* block type flag */
OMX_INT  mixedBlock;      /* flag 0: non mixed block, 1: mixed block */
OMX_INT  pTableSelect[3];  /* Huffman table index for the 3 rectangle in
                            <big_values> field */
OMX_INT  pSubBlkGain[3]; /* gain offset from the global gain for one
                            subblock */
OMX_INT  reg0Cnt;         /* the number of scale factor bands in
                            the first region of <big_values> less one */
OMX_INT  reg1Cnt;         /* the number of scale factor bands in
                            the second region of <big_values> less one
                          */
OMX_INT  preFlag;         /* flag indicating high frequency boost */
OMX_INT  sfScale;  /* scalefactor scaling */
OMX_INT  cntlTabSel;      /* Huffman table index for the <count1> quadruples
   */
} OMXMP3SideInfo;
```

3.1.2.3 ScaleFactorBand Table

```
#define OMX_MP3_SFB_TABLE_LONG_LEN138
```

```
typedef const OMX_S16
   OMXMP3ScaleFactorBandTableLong[OMX_MP3_SFB_TABLE_LONG_LEN]; /* 138
   elements */
#define OMX_MP3_SFB_TABLE_SHORT_LEN      84
typedef const OMX_S16
   OMXMP3ScaleFactorBandTableShort[OMX_MP3_SFB_TABLE_SHORT_LEN]; /* 84
   elements */
```

3.1.3 Functions

3.1.3.1 Bitstream Unpacking

3.1.3.1.1 UnpackFrameHeader

Prototype

```
OMXResult omxACMP3_UnpackFrameHeader (const OMX_U8 **ppBitStream,
   OMXMP3FrameHeader *pFrameHeader);
```

Description

Unpacks the audio frame header. If CRC is enabled, this function also unpacks the CRC word. Before calling `omxACMP3_UnpackFrameHeader`, the decoder application should locate the bit stream sync word and ensure that `*ppBitStream` points to the first byte of the 32-bit frame header. If CRC is enabled, it is assumed that the 16-bit CRC word is adjacent to the 32-bit frame header, as defined in the MP3 standard. Before returning to the caller, the function updates the pointer `*ppBitStream,` such that it references the next byte after the frame header or the CRC word. The first byte of the 16-bit CRC word is stored in `pFrameHeader->CRCWord[15:8]`, and the second byte is stored in `pFrameHeader->CRCWord[7:0]`. The function does not detect corrupted frame headers.

Reference

ISO/IEC 13818-3:1998, 2.4.2.3

Input Arguments

- `ppBitStream` – double pointer to the first byte of the MP3 frame header

Output Arguments

- `pFrameHeader` – pointer to the MP3 frame header structure (defined in section "Data Structures")
- `ppBitStream` – double pointer to the byte immediately following the frame header

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – invalid arguments – either `ppBitStream`, `pFrameHeader`, or `*ppBitStream` is Null

3.1.3.1.2 UnpackSideInfo

Prototype

```
OMXResult omxACMP3_UnpackSideInfo (const OMX_U8 **ppBitStream,
   OMXMP3SideInfo *pDstSideInfo, OMX_INT *pDstMainDataBegin, OMX_INT
   *pDstPrivateBits, OMX_INT *pDstScfsi, OMXMP3FrameHeader *pFrameHeader);
```

Description

Unpacks the side information from the input bit stream. Before `omxACMP3_UnpackSideInfo` is called, the pointer `*ppBitStream` must point to the first byte of the bit stream that contains the side information associated with the current frame. Before returning to the caller, the function updates the pointer `*ppBitStream` such that it references the next byte after the side information.

Reference

ISO/IEC 13818-3:1998, 2.4.1.7

Input Arguments

- `ppBitStream` – double pointer to the first byte of the side information associated with the current frame in the bit stream buffer
- `pFrameHeader` – pointer to the structure that contains the unpacked MP3 frame header. The header structure provides format information about the input bit stream. Both single- and dual-channel MPEG-1 and MPEG-2 modes are supported.

Output Arguments

- `pDstSideInfo` – pointer to the MP3 side information structure(s). The structure(s) contain(s) side information that applies to all granules and all channels for the current frame. One or more of the structures are placed contiguously in the buffer pointed by `pDstSideInfo` in the following order: { granule 0 (channel 0, channel 1), granule 1 (channel 0, channel 1) }.
- `pDstMainDataBegin` – pointer to the `main_data_begin` field
- `pDstPrivateBits` – pointer to the `private bits` field
- `pDstScfsi` – pointer to the scalefactor selection information associated with the current frame, organized contiguously in the buffer pointed to by `pDstScfsi` in the following order: {channel 0 (scfsi_band 0, scfsi_band 1, ..., scfsi_band 3), channel 1 (scfsi_band 0, scfsi_band 1, ..., scfsi_band 3) }.
- `ppBitStream` – double pointer to the bit stream buffer byte immediately following the side information for the current frame

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad argument; at least one of the following pointers is NULL:
  - `ppBitStream`
  - `pDstSideInfo`
  - `pDstMainDataBegin`

— `pDstPrivateBits`

— `DstScfsi`

— `pFrameHeader`

— `*ppBitStream`

- `OMX_StsErr` – one or more elements of the MP3 frame header structure is invalid, i.e., one or more of the following conditions is true:

  — `pFrameHeader->id` exceeds `[0,1]`;

  — `pFrameHeader->layer!=1`

  — `pFrameHeader->mode` exceeds `[0,3]`

  — `block_type` is normal and `window_switching_flag` is set.

3.1.3.1.3 UnpackScaleFactors_S8

Prototype

```
OMXResult omxACMP3_UnpackScaleFactors_S8 (const OMX_U8 **ppBitStream,
   OMX_INT *pOffset, OMX_S8 *pDstScaleFactor, OMXMP3SideInfo *pSideInfo,
   OMX_INT *pScfsi, OMXMP3FrameHeader *pFrameHeader, OMX_INT granule,
   OMX_INT channel);
```

Description

Unpacks short and/or long block scalefactors for one granule of one channel and places the results in the vector `pDstScaleFactor`. Before returning to the caller, the function updates `*ppBitStream` and `*pOffset` such that they point to the next available bit in the input bit stream.

---

2 ***Note:*** *If the intensity position is equal to the maximum value of intensity position (an illegal position), the illegal position is set to negative. Thus, in the requantization module, negative positions indicate illegal positions. Those scalefactors that are not treated as intensity positions must be made positive before using them.*

Reference

ISO/IEC 13818-3 2.4.1.7

Input Arguments

- `ppBitStream` – double pointer to the first bit stream buffer byte that is associated with the scalefactors for the current frame, granule, and channel
- `pOffset` – pointer to the next bit in the byte referenced by `*ppBitStream`. Valid within the range of 0 to 7, where 0 corresponds to the most significant bit and 7 corresponds to the least significant bit.
- `pSideInfo` – pointer to the MP3 side information structure associated with the current granule and channel

- `pScfsi` – pointer to scalefactor selection information for the current channel
- `channel` – channel index; can take on the values of either 0 or 1
- `granule` – granule index; can take on the values of either 0 or 1
- `pFrameHeader` – pointer to MP3 frame header structure for the current frame

Output Arguments

- `pDstScaleFactor` – pointer to the scalefactor vector for long and/or short blocks
- `ppBitStream` – updated double pointer to the next bit stream byte
- `pOffset` – updated pointer to the next bit in the bit stream (indexes the bits of the byte pointed to by `*ppBitStream`). Valid within the range of 0 to 7, where 0 corresponds to the most significant bit and 7 corresponds to the least significant bit.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments; one or more of the following pointers is NULL:

  — `*ppBitStream`

  — `pOffset`

  — `pDstScaleFactor`

  — `pSideInfo`, `pScfsi`

  — `*ppBitStream`

  — `pFrameHeader`

Bad arguments are also flagged when the value of `*pOffset` exceeds [0,7] or the granule or channel indices have values other than 0 or 1.

- `OMX_StsErr` – input data errors detected; one or more of the following are true:

  — `pFrameHeader ->id` exceeds [0,1],

  — `pSideInfo->blockType` exceeds [0,3],

  — `pSideInfo->mixedBlock` exceeds [0,1].

  — `pScfsi` [0..3] exceeds [0,1].

  — If `pFrameHeader->id` indicates that the bit stream is MPEG-1

  — `pSideInfo->sfCompress` exceeds [0,15]

  — If `pFrameHeader->id` indicates the bit stream is MPEG-2

  — `pSideInfo->sfCompress` exceeds [0,511]

  — `pFrameHeader->modeExt` exceeds [0, 3]

3.1.3.2 Huffman Decoding

3.1.3.2.1 HuffmanDecode_S32

3.1.3.2.2 HuffmanDecodeSfb_S32

3.1.3.2.3 HuffmanDecodeSfbMbp_S32

Prototype

```
OMXResult omxACMP3_HuffmanDecode_S32 (const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMX_S32 *pDstIs, OMX_INT *pDstNonZeroBound, OMXMP3SideInfo
    *pSideInfo, OMXMP3FrameHeader *pFrameHeader, OMX_INT hufSize);
OMXResult omxACMP3_HuffmanDecodeSfb_S32(const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMX_S32 *pDstIs, OMX_INT *pDstNonZeroBound, OMXMP3SideInfo
    *pSideInfo, OMXMP3FrameHeader *pFrameHeader, OMX_INT hufSize,
    OMXMP3ScaleFactorBandTableLong pSfbTableLong);
OMXResult omxACMP3_HuffmanDecodeSfbMbp_S32(const OMX_U8 **ppBitStream,
    OMX_INT *pOffset, OMX_S32 *pDstIs, OMX_INT *pDstNonZeroBound,
    OMXMP3SideInfo *pSideInfo, OMXMP3FrameHeader *pFrameHeader, OMX_INT
    hufSize, OMXMP3ScaleFactorBandTableLong pSfbTableLong,
    OMXMP3ScaleFactorBandTableShort pSfbTableShort,
    OMXMP3MixedBlockPartitionTable pMbpTable);
```

Description

Decodes Huffman symbols for the 576 spectral coefficients associated with one granule of one channel.

Reference

ISO/IEC 13818-3 2.5.2.8.

Input Arguments

- `ppBitStream` – double pointer to the first bit stream byte that contains the Huffman code words associated with the current granule and channel
- `pOffset`– pointer to the starting bit position in the bit stream byte pointed by `*ppBitStream`; valid within the range of 0 to 7, where 0 corresponds to the most significant bit, and 7 corresponds to the least significant bit
- `pSideInfo` – pointer to MP3 structure that contains the side information associated with the current granule and channel
- `pFrameHeader` – pointer to MP3 structure that contains the header associated with the current frame
- `hufSize` – the number of Huffman code bits associated with the current granule and channel
- `pSfbTableLong` – pointer to Scalefactor band table for long block. User can use the default table from MPEG-1 or MPEG-2 standards. User can also use his own table for special purpose. For table format see references [ISO93] Table B.8 and [ISO98] Table B.2.

- `pSfbTableShort` – pointer to Scalefactor band table for short block. User can use the default table from MPEG-1, MPEG-2 standards. User can also use his own table for special purpose. For table format see references [ISO93] Table B.8 and [ISO98] Table B.2.
- `pMbpTable` – pointer to Scalefactor band table for mixed block. User can use the default table from MPEG-1, MPEG-2 standards. User can also use his own table for special purpose.

Output Arguments

- `pDstIs` – pointer to the vector of decoded Huffman symbols used to compute the quantized values of the 576 spectral coefficients that are associated with the current granule and channel
- `pDstNonZeroBound` – pointer to the spectral region above which all coefficients are set equal to zero
- `ppBitStream` – updated double pointer to the particular byte in the bit stream that contains the first new bit following the decoded block of Huffman codes
- `pOffset` – updated pointer to the next bit position in the byte pointed by `*ppBitStream`; valid within the range of 0 to 7, where 0 corresponds to the most significant bit, and 7 corresponds to the least significant bit

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments detected; at least one of the following pointers is NULL:

  — `ppBitStream`

  — `pOffset`

  — `pDstIs`

  — `pDstNonZeroBound`

  — `pSideInfo`

  — `pFrameHeader`

  — `pSfbTableLong`

  — `*ppBitStream`.
- The flag is also asserted when either of the following is true: `*pOffset` < 0, or `*pOffset` > 7.
- `OMX_StsErr` – indicates that the number of remaining Huffman code bits for <count1> partition is less than zero after decoding the <big_values> partition; alternatively, as shown in Table 3-2, the code could also indicate either that one or more elements of the MP3 side information are invalid or that one or more elements of the MP3 frame header are invalid.

Table 3-2:OMX_StsErr List

Input Data	Invalid Value	Condition
pSideInfo-> bigVals * 2	>OMX_MP3_GRANULE_LEN	None
pSideInfo->bigVals * 2	< 0	None
pSideInfo->winSwitch	Exceeds [0,1]	None

Input Data	Invalid Value	Condition
pSideInfo-> blockType	Exceeds [0,3]	None
pSideInfo->blockType	==0	1 == pSideInfo ->winSwitch
pSideInfo->cnt1TabSel	Exceeds [0,1]	None
pSideInfo-> reg0Cnt	< 0	0 == pSideInfo -> blockType
pSideInfo-> reg1Cnt	< 0	0 == pSideInfo ->blockType
pSideInfo-> reg0Cnt + pSideInfo -> reg1Cnt + 2	> 22	0 == pSideInfo -> blockType
pSideInfo-> pTableSelect [0]	Exceeds [0,31]	None
pSideInfo-> pTableSelect[1]	Exceeds [0,31]	None
pSideInfo-> pTableSelect [2]	Exceeds[0,31]	0 == pSideInfo ->blockType
pFrameHeader-> id	Exceeds [0,1]	None
pFrameHeader-> layer	!=1	None
pFrameHeader-> samplingFreq	Exceeds [0,2]	None
hufSize	Exceeds [0, pSideInfo-> part23Len]	None

3.1.3.3 Inverse Quantization

3.1.3.3.1 ReQuantize_S32_I

3.1.3.3.2 ReQuantizeSfb_S32_I

Prototype

```
OMXResult omxACMP3_ReQuantize_S32_I(OMX_S32 *pSrcDstIsXr, OMX_INT
    *pNonZeroBound, OMX_S8 *pScaleFactor, OMXMP3SideInfo *pSideInfo,
    OMXMP3FrameHeader *pFrameHeader, OMX_S32 *pBuffer);
OMXResult omxACMP3_ReQuantizeSfb_S32_I(OMX_S32 *pSrcDstIsXr, OMX_INT
    *pNonZeroBound, OMX_S8 *pScaleFactor, OMXMP3SideInfo *pSideInfo,
    OMXMP3FrameHeader *pFrameHeader, OMX_S32 *pBuffer,
    OMXMP3ScaleFactorBandTableLong pSfbTableLong,
    OMXMP3ScaleFactorBandTableShort pSfbTableShort);
```

Description

Requantizes the decoded Huffman symbols. Spectral samples for the synthesis filter bank are derived from the decoded symbols using the requantization equations given in the ISO standard. Stereophonic mid/side (M/S) and/or intensity decoding is applied if necessary. Requantized spectral samples are returned in the vector `pSrcDstIsXr`. The reordering operation is applied for short blocks. Users must preallocate a workspace buffer pointed to by pBuffer prior to calling the requantization function. The value pointed by pNonZeroBound will be recalculated according to the output data sequence.

Reference

ISO/IEC 13818-3 2.5.3.2.2

Input Arguments

- pSrcDstIsXr – pointer to the vector of decoded Huffman symbols; for stereo and dual-channel modes, right channel data begins at the address &(pSrcDstIsXr[576])
- pNonZeroBound – (Inout/output argument) pointer to the spectral bound above which all coefficients are set to zero; for stereo and dual-channel modes, the left channel bound is pNonZeroBound [0], and the right channel bound is pNonZeroBound [1]
- pScaleFactor – pointer to the scalefactor buffer; for stereo and dual-channel modes, the right channel scalefactors begin at & (pScaleFactor [OMX_MP3_SF_BUF_LEN])
- pSideInfo – pointer to the side information for the current granule
- pFrameHeader – pointer to the frame header for the current frame
- pBuffer – pointer to a workspace buffer. The buffer length must be 576 samples
- pSfbTableLong – pointer to Scalefactor band table for long block. User can use the default table from MPEG-1 or MPEG-2 standards. User can also use his own table for special purpose. For table format see references [ISO93] Table B.8, and [ISO98] Table B.2.
- pSfbTableShort – pointer to Scalefactor band table for short block. User can use the default table from MPEG-1 or MPEG-2 standards. User can also use his own table for special purpose. For table format see references [ISO93] Table B.8 and [ISO98] Table B.2.

Output Arguments

- pSrcDstIsXr – pointer to the vector of requantized spectral samples for the synthesis filter bank, in Q5.26 format (Qm.n defined in "Introduction"). Only the first (pNonZeroBound[ch]+17)/18 18-point blocks data are effective. The others are meaningless at all.
- pNonZeroBound – (Input/output argument) pointer to the spectral bound above which all coefficients are set to zero; for stereo and dual-channel modes, the left channel bound is pNonZeroBound [0], and the right channel bound is pNonZeroBound [1].

Returns

- OMX_StsNoErr – No errors
- OMX_StsBadArgErr – bad arguments detected; one or more of the following pointers are NULL:
    - — pSrcDstIsXr
    - — pNonZeroBound
    - — pScaleFactor
    - — pSideInfo
    - — pFrameHeader
    - — pBuffer
- OMX_StsErr – one or more of the input error conditions listed in Table 3-3 is detected:

Table 3-3: OMX_StsErr List

Input Data	Invalid Value	Condition
pNonZeroBound [ch]	Exceeds [0,576]	None
pFrameHeader->id	Exceeds [0,1]	None
pFrameHeader -> samplingFreq	Exceeds [0,2]	None
pFrameHeader->mode	Exceeds [0,3]	None
pSideInfo [ch]. blockType	Exceeds [0,3]	None
pFrameHeader-modeExt	Exceeds [0,3]	None
pSideInfo [ch]. mixedBlock	Exceeds [0,1]	None
pSideInfo [ch] .globGain	Exceeds [0,255]	None
pSideInfo [ch]. sfScale	Exceeds [0,1]	None
pSideInfo [ch]. preFlag	Exceeds [0,1]	None
pSideInfo [ch]. pSubBlkGain [w]	Exceeds [0,7]	None
|pSrcDstIsXr [i]|	>8206	None
pScaleFactor [sfb]	> 7	If pScaleFactor [sfb] is the intensity position for MPEG-1.
pSideInfo [ch]. blockType	pSideInfo [0]. blockType!= pSideInfo [1]. blockType	If the bit stream is joint stereo mode
pSideInfo[ch].mixedBlock	pSideInfo[0].mixedblock ! = pSideInf[1].mixedBlock	If the bit stream is joint stereo mode

2 ***Note:*** *In Table 3-3, the range on ch is from 0 to chNum-1, and the range on w is from 0 to 2, where chNum is the number of channels decoded by the pFrameHeader ->mode. If pFrameHeader ->mode == 3 then chNum = 1, otherwise chNum = 2.*

3.1.3.4 Synthesis Filterbank

3.1.3.4.1 MDCTInv_S32

Prototype

```
OMXResult omxACMP3_MDCTInv_S32 ( OMX_S32 *pSrcXr, OMX_S32 *pDstY, OMX_S32
    *pSrcDstOverlapAdd, OMX_INT nonZeroBound, OMX_INT *pPrevNumOfImdct,
    OMX_INT blockType, OMX_INT mixedBlock);
```

Description

Stage 1 of the hybrid synthesis filter bank. This performs the following operations:

a) Alias reduction

b) Inverse MDCT according to block size specifiers and mixed block modes

c) Overlap add of IMDCT outputs, and

d) Frequency inversion prior to PQMF bank

Because the IMDCT is a lapped transform, the user must preallocate a buffer referenced by `pSrcDstOverlapAdd` to maintain the IMDCT overlap-add state. The buffer must contain 576 elements. Prior to the first call to the synthesis filter bank, all elements of the overlap-add buffer should be set equal to zero. In between all subsequent calls, the contents of the overlap-add buffer should be preserved. Upon entry to `omxACMP3_MDCTInv_S32`, the overlap-add buffer should contain the IMDCT output generated by operating on the previous granule; upon exit from `omxACMP3_MDCTInv_S32`, the overlap-add buffer will contain the overlapped portion of the output generated by operating on the current granule. Upon return from the function, the IMDCT sub-band output samples are organized as follows: `pDstY[j*32+subband]`, for j=0 to 17; sub-band=0 to 31.

---

2 ***Note:*** *The pointers pSrcXr (input argument) and pDstY (output argument) must reference different buffers.*

Reference

ISO/IEC 13818-3 2.5.3.3.2

Input Arguments

- `pSrcXr` – pointer to the vector of requantized spectral samples for the current channel and granule, represented in Q5.26 format.

---

2 ***Note:*** *The vector buffer is used as a workspace buffer when the input data has been processed. So the data in the buffer is meaningless when exiting the function*

- `pSrcDstOverlapAdd` – pointer to the overlap-add buffer; contains the overlapped portion of the previous granule's IMDCT output
- `nonZeroBound` – the bound above which all spectral coefficients are zero for the current granule and channel
- `pPrevNumOfImdct` – pointer to the number of IMDCTs computed for the current channel of the previous granule
- `blockType` – block type indicator
- `mixedBlock` – mixed block indicator

Output Arguments

- `pDstY` – pointer to the vector of IMDCT outputs in Q7.24 format, for input to PQMF bank
- `pSrcDstOverlapAdd` – pointer to the updated overlap-add buffer; contains overlapped portion of the current granule's IMDCT output
- `pPrevNumOfImdct` – pointer to the number of IMDCTs, for current granule, current channel

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments detected; one or more of the pointers `pSrcXr`, `pDstY`, `pSrcDstOverlapAdd`, and/or `pPrevNumOfImdct` is NULL
- `OMX_StsErr` – one or more of the following input data errors detected: either `blockType` exceeds [0,3], `mixedBlock` exceeds [0,1], `nonZeroBound` exceeds [0,576], or `*pPrevNumOfImdct` exceeds [0,32]

3.1.3.4.2 SynthPQMF_S32_S16

Prototype

```
OMXResult omxACMP3_SynthPQMF_S32_S16(OMX_S32 *pSrcY, OMX_S16 *pDstAudioOut,
    OMX_S32 *pVBuffer, OMX_INT *pVPosition, OMX_INT mode);
```

Description

Stage 2 of the hybrid synthesis filter bank; a critically-sampled 32-channel PQMF synthesis bank that generates 32 time-domain output samples for each 32-sample input block of IMDCT outputs. For each input block, the PQMF generates an output sequence of 16-bit signed little-endian PCM samples in the vector pointed to by pDstAudioOut. If `mode` equals 2, the left and right channel output samples are interleaved (i.e., LRLRLR), such that the left channel data is organized as follows: `pDstAudioOut [2*i], i=0 to 31`. If `mode` equals 1, then the left and right channel outputs are not interleaved. A workspace buffer of size 512 x Number of Channels must be preallocated (pVBuffer). This buffer is referenced by the pointer `pVBuffer`, and its elements should be initialized to zero prior to the first call. During subsequent calls, the `pVBuffer` input for the current call should contain the `pVbuffer` output generated by the previous call. The state variable `pVPosition` should be initialized to zero and preserved between calls. The values contained in `pVBuffer` or `pVPosition` should be modified only during decoder reset, and the reset values should always be zero.

Reference

ISO/IEC 13818-3 2.5.3.3.2

Input Arguments

- `pSrcY` – pointer to the block of 32 IMDCT sub-band input samples, in Q7.24 format
- `pVBuffer` – pointer to the input workspace buffer containing Q7.24 data. The elements of this buffer should be initialized to zero during decoder reset. During decoder operation, the values contained in this buffer should be modified only by the PQMF function.

- `pVPosition` – pointer to the internal workspace index; should be initialized to zero during decoder reset. During decoder operation, the value of this index should be preserved between PQMF calls and should be modified only by the function.
- `mode` – flag that indicates whether or not the PCM audio output channels should be interleaved

  1 – not interleaved

  2 – interleaved

Output Arguments

- `pDstAudioOut` – pointer to a block of 32 reconstructed PCM output samples in 16-bit signed format (little-endian); left and right channels are interleaved according to the mode flag. This should be aligned on a 4-byte boundary
- `pVBuffer` – pointer to the updated internal workspace buffer containing Q7.24 data; see usage notes under input argument discussion
- `pVPosition` – pointer to the updated internal workspace index; see usage notes under input argument discussion

Returns

- `OMX_StsNoErr` – No errors
- `OMX_StsBadArgErr` – bad arguments detected; either `mode < 1`, or `mode > 2`, or at least one of the following pointers is NULL: `pSrcY`, `pDstAudioOut`, `pVBuffer`, and/or `pVPosition`.
- `OMX_StsErr` – the value of `*pVPosition` exceeds [0, 15]

3.2 AAC-LC/LTP Decoder Sub-Domain (omxACAAC)

3.2.1 Constants

Table 3-4: AAC-LC/LTP Constants

Global Macro Name	Definition	Notes
OMX_AAC_FRAME_LEN	1024	The number of data in one frame
OMX_AAC_SF_LEN	120	scalefactor buffer length
OMX_AAC_GROUP_NUM_MAX	8	maximum group number for one frame
OMX_AAC_TNS_COEF_LEN	60	TNS coefficients buffer length
OMX_AAC_TNS_FILT_MAX	8	maximum TNS filters for one frame
OMX_AAC_PRED_SFB_MAX	41	maximum prediction scalefactor bands for one frame
OMX_AAC_ELT_NUM	16	maximum number of elements for one program.
OMX_AAC_LFE_ELT_NUM	4	maximum Low Frequency Enhance elements number for one program
OMX_AAC_DATA_ELT_NUM	8	maximum data elements number for one program
OMX_AAC_COMMENTS_LEN	256	maximum length of the comment field, in bytes.
OMX_AAC_SF_MAX	60	maximum number of scalefactor bands in one window
OMX_AAC_WIN_MAX	8	
OMX_AAC_MAX_LTP_SFB	40	

3.2.2 Data Structures

3.2.2.1 ADIF Header

```
typedef struct {
```

```
    OMX_U32 ADIFId;               /** 32-bit, "ADIF" ASCII code */
    OMX_INT     copyIdPres;       /** copy id flag: 0: off, 1: on */
    OMX_INT     originalCopy;     /** original bitstream or copy, 0: copy,
   1: original */
    OMX_INT    home;
    OMX_INT     bitstreamType;    /** bitstream flag: 0: constant rate
   bitstream, 1: varible rate bitstream */
    OMX_INT     bitRate;          /** bit rate. if 0, unkown bit rate */
    OMX_INT     numPrgCfgElt;     /** number of program config elements */
    OMX_INT     pADIFBufFullness[OMX_AAC_ELT_NUM];  /** buffer fullness */
    OMX_U8      pCopyId[9];       /** 72-bit copy id */
} OMXAACADIFHeader;
```

3.2.2.2 ADTS Frame Header

```
typedef struct {
   /** ADTS fixed header */
    OMX_INT id;                       /** ID 1*/
    OMX_INT layer;                    /** layer index 0x3: Layer I
                              //             0x2: Layer II
                              //             0x1: Layer III */
    OMX_INT protectionBit;    /** CRC flag 0: CRC on, 1: CRC off */
    OMX_INT profile;          /** profile: 0:MP, 1:LP, 2:SSR */
    OMX_INT samplingRateIndex;/** sampling frequency index */
    OMX_INT privateBit;       /** private_bit, no use  */
    OMX_INT chConfig;         /** channel configuration */
    OMX_INT originalCopy;     /** original bitstream or copy, 0: copy, 1:
   original */
    OMX_INT home;
    OMX_INT emphasis;         /** not used by ISO/IEC 14496-3, but used by
   14490-3 */

   /** ADTS variable header */
    OMX_INT cpRightIdBit;    /** copyright id bit */
    OMX_INT cpRightIdStart;  /** copyright id start */
    OMX_INT frameLen;        /** frame length in bytes */
    OMX_INT ADTSBufFullness; /** buffer fullness */
    OMX_INT numRawBlock;     /** number of raw data blocks in the frame */
```

```
    /** ADTS CRC error check, 16bits */
     OMX_INT CRCWord;         /** CRC-check word */
} OMXAACADTSFrameHeader;
```

3.2.2.3 Individual Channel Side Information

```
typedef struct {
    /** unpacked from the bitstream */
    OMX_INT     icsReservedBit;
     OMX_INT     winSequence;                         /** window sequence
    flag */
     OMX_INT     winShape;                            /** window shape flag,
    0: sine window, 1: KBD window */
     OMX_INT     maxSfb;                              /** maximum effective
    scalefactor bands */
     OMX_INT     sfGrouping;                          /** scalefactor
    grouping flag */
     OMX_INT     predDataPres;                        /** prediction data
    present flag for one fraem, 0: prediction off, 1: prediction on */
     OMX_INT     predReset;                           /** prediction reset
    flag, 0: reset off, 1: reset on */
     OMX_INT     predResetGroupNum;                   /** prediction reset
    group number */
     OMX_U8  pPredUsed[OMX_AAC_PRED_SFB_MAX+3];      /** prediction flag
    buffer for each scalefactor band: 0: off, 1: on
                                                         // buffer length 44
    bytes, 4-byte align   */
     /** decoded from the above info */
     OMX_INT     numWinGrp;                           /** number of
    window_groups */
     OMX_INT     pWinGrpLen[OMX_AAC_GROUP_NUM_MAX];  /** buffer for number of
    windows in each group */
} OMXAACIcsInfo;
```

3.2.2.4 Program Configuration Element

```
typedef struct {
    OMX_INT     eltInsTag;
    OMX_INT     profile;
    OMX_INT     samplingRateIndex;
    OMX_INT     numFrontElt;
    OMX_INT     numSideElt;
```

```
    OMX_INT     numBackElt;
    OMX_INT     numLfeElt;
    OMX_INT     numDataElt;
    OMX_INT     numValidCcElt;
    OMX_INT     monoMixdownPres;
    OMX_INT     monoMixdownEltNum;
    OMX_INT     stereoMixdownPres;
    OMX_INT     stereoMixdownEltNum;
    OMX_INT     matrixMixdownIdxPres;
    OMX_INT     matrixMixdownIdx;
    OMX_INT     pseudoSurroundEnable;
    OMX_INT     pFrontIsCpe[OMX_AAC_ELT_NUM];
    OMX_INT     pFrontTagSel[OMX_AAC_ELT_NUM];
    OMX_INT     pSideIsCpe[OMX_AAC_ELT_NUM];
    OMX_INT     pSideTagSel[OMX_AAC_ELT_NUM];
    OMX_INT     pBackIsCpe[OMX_AAC_ELT_NUM];
    OMX_INT     pBackTagSel[OMX_AAC_ELT_NUM];
    OMX_INT     pLfeTagSel[OMX_AAC_LFE_ELT_NUM];
    OMX_INT     pDataTagSel[OMX_AAC_DATA_ELT_NUM];
    OMX_INT     pCceIsIndSw[OMX_AAC_ELT_NUM];
    OMX_INT     pCceTagSel[OMX_AAC_ELT_NUM];
    OMX_INT     numComBytes;
    OMX_S8      pComFieldData[OMX_AAC_COMMENTS_LEN];
} OMXAACPrgCfgElt;
```

3.2.2.5 LTP Information

```
typedef struct{
    OMX_INT ltpDataPresent;  /** if ltp is used */
    OMX_INT ltpLag;          /** the optimal delay from 0 to 2047 */
    OMX_S16 ltpCoef;         /** indicate the  LTP coefficient */
    OMX_INT pLtpLongUsed[OMX_AAC_MAX_LTP_SFB+1]; /** if long block use ltp*/
    OMX_INT pLtpShortUsed[OMX_AAC_WIN_MAX]; /** if short block use ltp */
    OMX_INT pLtpShortLagPresent[OMX_AAC_WIN_MAX]; /** if short lag is
   transmitted */
    OMX_INT pLtpShortLag[OMX_AAC_WIN_MAX]; /** relative delay for short
   window */
}OMXAACLtpInfo, *OMXAACLtpInfoPtr;
```

3.2.2.6 Channel Pair Element

```
typedef struct {
    OMX_INT     commonWin;   /** common window flag, 0: off, 1: on */
    OMX_INT     msMaskPres;  /** MS stereo mask present flag */
    OMX_U8      ppMsMask[OMX_GROUP_NUM_MAX][OMX_AAC_SF_MAX];/** MS stereo
   flag buffer for each scalefactor band */
} OMXAACChanPairElt;
```

3.2.2.7 Channel Information

```
typedef struct {
   OMX_INTtag;
   OMX_INTid;      /* element id */
   OMX_INTsamplingRateIndex;/* sampling rate index */
   OMX_INTpredSfbMax;     /* maximum prediction scalefactor bands */
   OMX_INTpreWinShape;    /* previous block window shape */

   OMX_INTwinLen;         /* 128: if short window, 1024: others */
   OMX_INTnumWin;         /* 1 for long block, 8 for short block */
   OMX_INTnumSwb;         /* decided by sampling frequency and block type */


   /* unpacking from the bitstream */
   OMX_INTglobGain;       /* global gain */
   OMX_INTpulseDataPres;  /* pulse data present flag, 0: off, 1: on */
   OMX_INTtnsDataPres;    /* TNS data present flag, 0: off, 1: on */
   OMX_INTgainContrDataPres;/* gain control data present flag, 0: off, 1:
   on */

   /* icsInfo pointer */
   OMXAACIcsInfo *pIcsInfo;            /** pointer to OMXAACIcsInfo
   structure */

   //** channel pair element pointer */
    OMXAACChanPairElt *pChanPairElt; /** pointer to OMXAACChanPairElt
   structure */

   /** section data */
    OMX_U8 pSectCb[OMX_AAC_SF_LEN];         /** section code book buffer */
```

```
    OMX_U8 pSectEnd[OMX_AAC_SF_LEN];        /** at which scalefactor band
   each     section ends */
    OMX_INT pMaxSect[OMX_AAC_GROUP_NUM_MAX];/** maximum section number for
   each group */

   /** TNS data */
    OMX_INT pTnsNumFilt[OMX_AAC_GROUP_NUM_MAX];     /** TNS number filter
   buffer */
    OMX_INT pTnsFiltCoefRes[OMX_AAC_GROUP_NUM_MAX]; /** TNS coefficients
   resolution flag */
    OMX_INT pTnsRegionLen[OMX_AAC_TNS_FILT_MAX];    /** TNS filter length */
    OMX_INT pTnsFiltOrder[OMX_AAC_TNS_FILT_MAX];    /** TNS filter order */
    OMX_INT pTnsDirection[OMX_AAC_TNS_FILT_MAX];    /** TNS filter direction
   flag */
}OMXAACChanInfo;
```

---

2 ***Note:*** *In the following sections, maxSfb means number of scalefactor bands transmitted per group. This is unpacked from the bit stream. numSwb means number of scalefactor window bands for short block or number of scalefactor window bands for long block. This is calculated according to the sampling rate and the block type.*

See clause 8.3.1 of *ISO/IEC 14496-3:1997*.

3.2.3 Functions

3.2.3.1 Bitstream Unpacking

3.2.3.1.1 UnpackADIFHeader

Prototype

```
OMXResult omxACAAC_UnpackADIFHeader(const OMX_U8 **ppBitStream,
   OMXAACADIFHeader *pADIFHeader, OMXAACPrgCfgElt *pPrgCfgElt, OMX_INT
   prgCfgEltMax);
```

Description

Gets the AAC ADIF format header, including program configuration elements from the input bit stream.

Reference

ISO/IEC 14496-3(1999E), Table 1.A.2.

Input Arguments

- `ppBitStream` – double pointer to the current byte before the ADIF header
- `prgCfgEltMax` – the maximum program configure element number

Output Arguments

- `ppBitStream` – double pointer to the current byte after the ADIF header
- `pADIFHeader` – pointer to the `OMXACCADIFHeader` structure
- `pPrgCfgElt` – pointer to the `OMXAACPrgCfgElt` structure. There must be `prgCfgEltMax` elements in the buffer.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the following pointers: `ppBitStream`, `pADIFHeader`, `pPrgCfgElt` `*ppBitStream` is Note: NULL
  - `prgCfgEltMax` exceeds [1, 16]
- `OMX_StsAacPrgNumErr` – the decoded `pADIFHeader->numPrgCfgElt > prgCfgEltMax.`

---

2 ***Note:*** *pADIFHeader->numPrgCfgElt is the number directly unpacked from bit stream plus 1. prgCfgEltMax is the number of the program configuration elements that the user wants to support. The valid range is [1, 16]*

3.2.3.1.2 UnpackADTSFrameHeader

Prototype

```
OMXResult omxACAAC_UnpackADTSFrameHeader (const OMX_U8 **ppBitStream,
   OMXAACADTSFrameHeader *pADTSFrameHeader);
```

Description

Gets ADTS frame header from the input bit stream. If the CRC word is applied, the first byte of the 16-bit CRC word is stored in `pADTSFrameHeader->CRCWord[15:8]` and the second byte is stored in `pADTSFrameHeader->CRCWord[7:0]`. It does not check whether the header is corrupt.

Reference

ISO/IEC 14496-3(1999E) Table 1.A.6.

Input Arguments

- `ppBitStream` – double pointer to the current byte

Output Arguments

- `ppBitStream` – double pointer to the current byte after unpacking the ADTS frame header.
- `pADTSFrameHeader` – pointer to the `OMXAACADTSFrameHeader` structure

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. At least one of the following pointers: `ppBitStream`, `*ppBitStream`, or `pADTSFrameHeader` is NULL

3.2.3.1.3 DecodePrgCfgElt

Prototype

```
OMXResult omxACAAC_DecodePrgCfgElt(const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMXAACPrgCfgElt *pPrgCfgElt);
```

Description

Gets program configuration element from the input bit stream.

Reference

ISO/IEC 14496-3(1999E), Table 4.4.2.

Input Arguments

- `ppBitStream` – double pointer to the current byte
- `pOffset` – pointer to the bit position in the byte pointed by `*ppBitStream`.. Valid within 0 to 7. 0: MSB of the byte, 7: LSB of the byte

Output Arguments

- `ppBitStream` – double pointer to the current byte after decoding the program configuration element
- `pOffset` – pointer to the bit position in the byte pointed by `*ppBitStream`.
  Valid within 0 to 7. 0: MSB of the byte, 7: LSB of the byte.
- `pPrgCfgElt` – pointer to `OMXAACPrgCfgElt` structure

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments
  - — At least one of the following pointers: `ppBitStream`, `pOffset`, `pPrgCfgElt`, `*ppBitStream` is NULL
  - — `*pOffset` exceeds [0, 7]

3.2.3.1.4 DecodeChanPairElt

Prototype

```
OMXResult omxACAAC_DecodeChanPairElt(const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMXAACIcsInfo *pIcsInfo, OMXAACChanPairElt *pChanPairElt,
    OMX_INT predSfbMax, OMX_INT audioObjectType, OMXAACLtpInfoPtr
    *pLtpInfo);
```

Description

Decodes the `channel_pair_element` from the input bitstream. `Individual_channel_stream` is not included. If `common_window` flag == 0, then only the structure field `pChanPairElt`->`commonWin` is updated, and all other `pIcsInfo` and `pChanPairElt` fields remain unchanged.

Reference

ISO/IEC 14496-3 Table 4.4.5

Input Arguments

- `ppBitStream` – double pointer to the current byte in the input bitstream
- `pOffset` – pointer to the next available bit of the input bitstream byte referenced by `*ppBitStream`. Valid in the range 0 to 7, where 0 signifies the most significant bit and 7 signifies the least significant bit
- `predSfbMax` – maximum prediction scalefactor band. Set `predSfbMax=0` for LC audio object types.
- `audioObjectType` – index of the audio object type: 2=LC, 4=LTP

Output Arguments

- `ppBitStream` – double pointer to the current byte in the input bitstream, updated after decoding the channel pair element.
- `pOffset` – pointer to the next available bit of the input bitstream byte referenced by `*ppBitStream`. Valid in the range 0 to 7, where 0 signifies the most significant bit and 7 signifies the least significant bit.
- `pIcsInfo` – pointer to the updated `OMXAACIcsInfo` structure. If `pIcsInfo`->`predDataPres` = 0, set `pIcsInfo`->`predReset` = 0. Only the first `pIcsInfo`->`numWinGrp` elements in `pIcsInfo`-> `pWinGrpLen` are meaningful. Some members of the structure must not be changed, as shown in Table 3-5.
- `pChanPairElt` – pointer to `OMXAACChanPairElt` structure. Some members of the structure must not be changed, as shown in Table 3-6.
- `pLtpInfo` – pointer to array containing pointers to updated LTP information structures for each channel containing the LTP information decoded from the input stream

Table 3-5: Unchanged Members of pIcsInfo

Members	Conditions
SfGrouping	pIcsInfo->winSequence != 2
predResetGroupNum	pIcsInfo->predDataPres == 0 || pIcsInfo->predReset == 0
pPredUsed[sfb]	pIcsInfo->predDataPres == 0

Table 3-6: Unchanged Members of pChanPairElt

Members	Conditions
pMsUsed[sfb]	pChanPairElt->msMaskPres != 1

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers: `ppBitStream`, `pOffset`, `*ppBitStream`, `pIcsInfo` or `pChanPairElt` is NULL.
  - — `*pOffset` exceeds [0, 7]
  - — `predSfbMax` < 0
  - — `audioObjectType!=2 && audioObjectType!=4`
  - — `predSfbMax` > 41 (maximum value for all sampling frequency in main profile)
- `OMX_StsAacMaxsfbErr` – `pIcsInfo->maxSfb` decoded from bit stream greater than 51 (maximum scalefactor band for all sampling frequency)

3.2.3.1.5 DecodeDatStrElt

Prototype

```
OMXResult omxACAAC_DecodeDatStrElt(const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMX_INT *pDataTag, OMX_INT *pDataCnt, OMX_U8 *pDstDataElt);
```

Description

Gets `data_stream_element` from the input bit stream.

Reference

ISO/IEC 14496-3 table 4.4.10

Input Arguments

- `ppBitStream` – double pointer to the current byte

- `pOffset` – pointer to the bit position in the byte pointed by `*ppBitStream`. Valid within 0 to 7. 0: MSB of the byte, 7: LSB of the byte.

Output Arguments

- `ppBitStream` – double pointer to the current byte after the decode data stream element
- `pOffset` – pointer to the bit position in the byte pointed by `*ppBitStream`. Valid within 0 to 7. 0: MSB of the byte, 7: LSB of the byte.
- `pDataTag` – pointer to `element_instance_tag`.
- `pDataCnt` – pointer to the value of length of total data in bytes
- `pDstDataElt` – pointer to the data stream buffer that contains the data stream extracted from the input bit stream. There are 512 elements in the buffer pointed by `pDstDataElt`.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers: `ppBitStream`, `pOffset`, `*ppBitStream`, `pDataTag`, `pDataCnt` or `pDstDataElt` is NULL.
  - — `*pOffset` exceeds [0, 7]

3.2.3.1.6 DecodeFillElt

Prototype

```
OMXResult omxACAAC_DecodeFillElt(const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMX_INT *pFillCnt, OMX_U8 *pDstFillElt);
```

Description

Gets the fill element from the input bit stream.

Reference

ISO/IEC 14496-3 table 4.4.11.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte
- `pOffset` – pointer to the bit position in the byte pointed by `*ppBitStream`. Valid within 0 to 7. 0: MSB of the byte, 7: LSB of the byte

Output Arguments

- `ppBitStream` – pointer to the pointer to the current byte after the decode fill element
- `pOffset` – pointer to the bit position in the byte pointed by *ppBitStream. Valid within 0 to 7. 0: MSB of the byte, 7: LSB of the byte.
- `pFillCnt` – pointer to the value of the length of total fill data in bytes
- `pDstFillElt`– pointer to the fill data buffer of length 270

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers: `ppBitStream`, `pOffset`, `*ppBitStream`, `pFillCnt` or `pDstFillElt` is NULL
  - — `*pOffset` exceeds [0, 7]

3.2.3.2 Inverse Quantization

3.2.3.2.1 QuantInv_S32_I

Prototype

```
OMXResult omxACAAC_QuantInv_S32_I(OMX_S32 *pSrcDstSpectralCoef, const
   OMX_S16 *pScalefactor, OMX_INT numWinGrp, const OMX_INT *pWinGrpLen,
   OMX_INT maxSfb, const OMX_U8 *pSfbCb, OMX_INT samplingRateIndex, OMX_INT
   winLen);
```

Description

Inverse quantize the Huffman symbols for current channel. The equation is shown below.

$$pSrcDst[i] = sign(pSrcDst[i]) * (pSrcDst[i])^{\frac{4}{3}} * 2^{\left(\frac{1}{4}(pScalefactor[sfb]-100)\right)}$$

Reference

ISO/IEC 14496-3 Sect 4.6.1.

Input Arguments

- `pSrcDstSpectralCoef` – pointer to the input quantized coefficients. For short block the coefficients are interleaved by `scalefactor` window bands in each group. Buffer length must be 1024.
- `pScalefactor` – pointer to the scalefactor buffer, of length 120
- `numWinGrp` – group number
- `pWinGrpLen` – pointer to the number of windows in each group, of length 8
- `maxSfb` – max scalefactor bands number for the current block
- `pSfbCb` – pointer to the scalefactor band codebook, of length 120. Only `maxSfb` elements for each group are meaningful. There are no spaces between the sequence groups.
- `samplingRateIndex` – sampling rate index. Valid in [0, 11]
- `winLen` – the data number in one window

Output Arguments

- `pSrcDstSpectralCoef` – pointer to the destination inverse quantized coefficient in Q13.18 format. For short block, the coefficients are interleaved by scalefactor window bands in each group, of length 1024. The maximum error of output `pSrcDstSpectralCoef` [i] is listed in Table 3-7.

Table 3-7: Computation Error List for pSrcDstSpectralCoef

Output	Conditions	
max(error(pSrcDstSpectralCoef [i]))	Input abs(pSrcDstSpectralCoef [i])	Output abs(pSrcDstSpectralCoef[i])
3	<= 128	< 2 ^ 29
3	129~8191	<= 2 ^ 25
7	129~8191	< 2 ^ 29

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers: `pSrcDstSpectralCoef`, `pScalefactor`, `pWinGrpLen` or `pSfbCb` is NULL
  - — If short block `numWinGrp` exceeds [1, 8]
  - — If long block, `numWinGrp` != 1
  - — `maxSfb` exceed [0, 51]
  - — `samplingRateIndex` exceeds [0, 11]
  - — `winLen` is neither 1024 nor 128
- `OMX_StsAacCoefValErr` – input coefficients value pointed by `pSrcDstSpectralCoef` exceeds[-8191, 8191]
- `OMX_StsAacMaxsfbErr` – the calculated scalefactor band index exceeds the `numSwb` in each window

3.2.3.3 Joint Stereo Decoding

3.2.3.3.1 DecodeMsStereo_S32_I

Prototype

```
OMXResult omxACAAC_DecodeMsStereo_S32_I(OMX_S32 *pSrcDstL, OMX_S32
    *pSrcDstR, OMXAACChanPairElt *pChanPairElt, OMX_U8 *pSfbCb, OMX_INT
    numWinGrp, const OMX_INT *pWinGrpLen, OMX_INT maxSfb, OMX_INT
    samplingRateIndex, OMX_INT winLen);
```

Description

Performs M-S stereo decoding; converts the MS stereo jointly-coded scalefactor bands of a channel pair from the M-S representation to the L-R representation; also performs the `invert_intensity(group, sfb)` function and stores the values in the `pSfbCb` buffer. If `invert_intensity(group, sfb)` = -1, and if `*pSfbCb` = `INTERITY_HCB`, let `*pSfbCb` = `INTERITY_HCB2`; else if `*pSfbCb` = `INTERITY_HCB2`, let `*pSfbCb` = `INTERITY_HCB`. For scalefactor bands in which the MS stereo flag is asserted, the individual left and right channel spectral samples `pSrcDstL[i]` and `pSrcDstR[i]` are computed as follows:

$$pSrcDstL'[i] = pSrcDstL[i] + pSrcDstR[i],$$
$$pSrcDstR'[i] = pSrcDstL[i] - pSrcDstR[i].$$

Reference

ISO/IEC 14496-3 Sect 4.6.7.1.

Input Arguments

- `pSrcDstL` – pointer to left channel data in Q13.18 format. For short blocks, the coefficients are interleaved by scalefactor window bands in each group, of length 1024. `pSrcDstL` must be 8-byte aligned.
- `pSrcDstR` – pointer to right channel data in Q13.18 format. For short block, the coefficients are interleaved by scalefactor window bands in each group, of length 1024. `pSrcDstR` must be 8-byte aligned.
- `pChanPairElt` – pointer to a Channel Pair Element structure that has been previously populated. At minimum, the contents of msMaskPres and pMsUsed fields are used to control MS decoding process and must be valid. These provide, respectively, the MS stereo mask for a scalefactor band (0: MS Off, 1: MS On, 2: all bands on), and the MS stereo flag buffer, of length 120.
- `pSfbCb -` pointer to the scalefactor band codebook, of length 120. Stores `maxSfb` elements for each group. There is no space between the sequence groups
- `numWinGrp` – group number
- `pWinGrpLen` – pointer to the number of windows in each group, of length 8
- `maxSfb` – max scalefactor bands number for the current block
- `samplingRateIndex` – sampling rate index; valid in the range [0, 11]
- `winLen` – the data number in one window

Output Arguments

- `pSrcDstL` – pointer to left channel data in Q13.18 format. For short blocks, the coefficients are interleaved by scalefactor window bands in each group, of length 1024. `pSrcDstL` must be 8-byte aligned.
- `pSrcDstR` – pointer to right channel data in Q13.18 format. For short blocks, the coefficients are interleaved by scalefactor window bands in each group, of length 1024. `pSrcDstR` must be 8-byte aligned.

- pSfbCb– pointer to the scalefactor band codebook. If invert_intensity group, sfb) = -1, and if *pSfbCb = INTERITY_HCB, let *pSfbCb = INTERITY_HCB2; else if *pSfbCb = INTERITY_HCB2, let *pSfbCb = INTERITY_HCB. Buffer length is 120. Store maxSfb elements for each group. There is no space between the sequence groups.

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments
  - — At least one of the following pointers: pSrcDstL, pSrcDstR, pMsUsed, pWinGrpLen, pSfbCb is NULL.
  - — pSrcDstL or pSrcDstR is not 8-byte aligned
  - — For short blocks, numWinGrp exceeds [1, 8]
  - — For long blocks, numWinGrp != 1
  - — maxSfb exceeds [0, 51]
  - — msMaskPres exceeds [1, 2]
  - — samplingRateIndex exceeds [0, 11]
  - — winLen is neither 1024 nor 128
- OMX_StsAacMaxsfbErr – the calculated scalefactor band index exceeds the numSwb in each window

3.2.3.3.2 DecodeIsStereo_S32

Prototype

```
OMXResult omxACAAC_DecodeIsStereo_S32(const OMX_S32 *pSrcL, OMX_S32 *pDstR,
    const OMX_S16 *pScalefactor, const OMX_U8 *pSfbCb, OMX_INT numWinGrp,
    const OMX_INT *pWinGrpLen, OMX_INT maxSfb, OMX_INT samplingRateIndex,
    OMX_INT winLen);
```

Description

Decodes jointly-coded scalefactor bands into discrete L/R stereo pairs for scalefactor bands in which the intensity stereo indicator flag stored in pSfbCb[sfb] is asserted. As described in ISO/IEC 14496-3, the discrete L/R signals pSrcL[i], pDstR[i] are recovered from the intensity-coded representation (single channel spectral coefficients + scalefactor) using the scaling operation expressed below. The parameter invert_intensity(g, sfb) is *not* used in the formula, since it decoded and stored in pSfbCb[sfb] by the MS stereo decoder.

$$pDstR[i] = pSrcL[i] * is\_intensity(g, sfb) * 2^{\left(-\frac{1}{4} pScalefactor[sfb]\right)}$$

Reference

ISO/IEC 14496-3 Sect 4.6.7.2

Input Arguments

- `pSrcL` – pointer to left channel data in Q13.18 format. For short block, the coefficients are interleaved by scalefactor window bands in each group. Buffer length is 1024. `pSrcL` must be 8-byte aligned.
- `pScalefactor` – pointer to the scalefactor buffer, of length 120
- `pSfbCb` – pointer to the scalefactor band codebook, of length 120. Store `maxSfb` elements for each group. There are no spaces between the sequence groups.
- `numWinGrp` – group number
- `pWinGrpLen` – pointer to the number of windows in each group, of length 8
- `maxSfb -` Max scalefactor bands number for the current block
- `samplingRateIndex` – sampling rate index. Valid in [0, 11]
- `winLen` – the data number in one window

Output Arguments

- `pDstR` – pointer to right channel data in Q13.18 format. For short block, the coefficients are interleaved by scalefactor window bands in each group. Buffer length is 1024. `pDstR` must be 8-byte aligned.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the following pointers: `pSrcL`, `pDstR`, `pWinGrpLen`, `pScalefactor`, `pSfbCb` is NULL.
  - If pSrcL, pDstR is not 8-byte aligned.
  - If short block, `numWinGrp` exceeds [1, 8]
  - If long block, `numWinGrp` != 1
  - `maxSfb` exceeds [0, 51]
  - `samplingRateIndex` exceeds [0, 11]
  - `winLen` is neither 1024 nor 128
- `OMX_StsAacMaxsfbErr` – the calculated scalefactor band index exceeds the `numSwb` in each window.

3.2.3.4 Temporal Noise Shaping

3.2.3.4.1 DecodeTNS_S32_I

Prototype

```
OMXResult omxACAAC_DecodeTNS_S32_I(OMX_S32 *pSrcDstSpectralCoefs, const
    OMX_INT *pTnsNumFilt, const OMX_INT *pTnsRegionLen, const OMX_INT
    *pTnsFiltOrder, const OMX_INT *pTnsFiltCoefRes, const OMX_S8
    *pTnsFiltCoef, const OMX_INT *pTnsDirection, OMX_INT maxSfb, OMX_INT
    profile, OMX_INT samplingRateIndex, OMX_INT winLen);
```

Description

This function applies all-pole Temporal Noise Shaping (TNS) decoding filters to selected spectral coefficient regions. The output sequence is ready for the IMDCT synthesis bank.

Reference

ISO/IEC 14496-3 Sect 4.6.8.

Input Arguments

- `pSrcDstSpectralCoefs` – spectral coefficient input vector, of length 1024, represented using Q13.18 format
- `pTnsNumFilt` – pointer to a table containing the number of TNS filters that are applied on each window of the current frame. The table elements are indexed as follows: `pTnsNumFilt[w]`, w=0 to numWin-1; depending upon the current window sequence, this vector may contain up to 8 elements.
- `pTnsRegionLen` – pointer to a table containing TNS region lengths (in scalefactor band units) for all regions and windows on the current frame; the table entry `pTnsRegionLen[i]` specifies the region length for k-th filter on the w-th window. The table index, i, is computed as follows:

$$i = \sum_{j=0}^{w-1} pTnsNumFilt[j] + k$$

where 0 <= w <= `numWin`-1, and 0 <= k <= `pTnsNumFilt[w]`-1.

- `pTnsFiltOrder` – pointer to a table containing TNS filter orders for all regions and windows on the current frame; the table entry `pTnsFiltOrder[i]` specifies the TNS filter order for the k-th filter on the w-th window. The table index, i, is computed as follows

$$i = \sum_{j=0}^{w-1} pTnsNumFilt[j] + k$$

where 0 <= w <= `numWin`-1, and 0 <= k <= `pTnsNumFilt`[w]-1.

- `pTnsFiltCoefRes` – pointer to a table of TNS filter coefficient resolution indicators for each window on the current frame. Resolutions for filters on the w-th window are specified in table entry `pTnsFiltCoefRes[w]`, and `w=0` to `numWin`-1.

- `pTnsFiltCoef` – pointer to a table containing the complete set of TNS filter coefficients for all windows and regions on the current frame. Filter coefficients are stored contiguously in filter-major order, i.e., the table is organized such that the filter coefficients for the k-th filter of the w-th window are indexed using pTnsFiltCoef[w][k][i], where 0 <= i <= pTnsFiltOrder[j]-1, 0 <= k <= pTnsNumFilt[w]-1, 0 <= w <= numWin-1, and the filter order index j is computed as shown above.
- `pTnsDirection` – pointer to a table of tokens that indicate the direction of TNS filtering for all regions and windows on the current frame, with 0 indicating upward and 1 indicating downward; in particular the table entry pTnsDirection[i] specifies direction for k-th filter on the w-th window, and the table index, i, is computed as follows:

$$i = \sum_{j=0}^{w-1} pTnsNumFilt[j] + k$$

where 0 <= w <= `numWin`-1, and 0 <= k <= `pTnsNumFilt`[w]-1.

- `maxSfb` – number of scalefactor bands transmitted per window group on the current frame
- `profile` – the profile index from Table 7.1 in *ISO/IEC 14496-3:1997*
- `samplingRateIndex` – sample rate index for the current frame
- `winLen` – window length

Output Arguments

`pSrcDstSpectralCoefs` – pointer to the output spectral coefficients after TNS filtering represented using Q13.18 format.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the following pointers: `pSrcDstSpectralCoefs`, `pTnsNumFilt`, `pTnsRegionLen`, `pTnsFiltOrder`, `pTnsFiltCoefRes`, `pTnsFiltCoef`, or `pTnsDirection` is NULL.
  - `maxSfb` < 0 or `maxSfb` > `numSwb`
  - `profile`!=1
  - `samplingRateIndex` exceeds [0, 11]
  - `winLen`!=128 and `winLen`!=1024
- `OMX_StsAacTnsNumFiltErr` – for a short window sequence, `pTnsNumFilt[w]` exceeds [0, 1]; For long window sequence, `pTnsNumFilt[w]` exceeds [0, 3], `w`=0 to `numWin`-1.
- `OMX_StsAacTnsLenErr` – `*pTnsRegionLen` exceeds [0, `numSwb`]
- `OMX_StsAacTnsOrderErr` – for short window sequence, `*pTnsFiltOrder` exceeds [0, 7]; For long window sequence, `*pTnsFiltOrder` exceeds [0, 12]
- `OMX_StsAacTnsCoefResErr` – `pTnsFiltCoefRes[w]` exceeds [3, 4], `w`=0 to `numWin`-1
- `OMX_StsAacTnsCoefErr` – `*pTnsFiltCoef` exceeds [-8, 7]
- `OMX_StsAacTnsDirectErr` – `*pTnsDirection` exceeds [0, 1]

2 ***Note:*** *numWin is the number of windows in a window sequence of the current frame. numWin is 8 if window sequence is EIGHT_SHORT_SEQUENCE, or it is 1 for other window sequences.*

numSwb is the total number of scalefactor window bands for the actual window type (long or short window) of the current frame.

3.2.3.5 Synthesis Filterbank

3.2.3.5.1 DeinterleaveSpectrum_S32

Prototype

```
OMXResult omxACAAC_DeinterleaveSpectrum_S32 (const OMX_S32 *pSrc, OMX_S32
    *pDst, OMX_INT numWinGrp, const OMX_INT *pWinGrpLen, OMX_INT maxSfb,
    OMX_INT samplingRateIndex, OMX_INT winLen);
```

Description

Deinterleaves the coefficients for short block.

Reference

ISO/IEC 14496-3 Sect 6.7.2.

Input Arguments

- `pSrc` – pointer to source coefficients buffer. The coefficients are interleaved by scalefactor window bands in each group. Buffer length is 1024. `pSrc` must be 8-byte aligned.
- `numWinGrp` – group number
- `pWinGrpLen` – pointer to the number of windows in each group. Buffer length is 8
- `maxSfb -` Max scalefactor bands number for the current block
- `samplingRateIndex` – sampling rate index. Valid in [0, 11]
- `winLen` – the data number in one window

Output Arguments

- `pDst` – pointer to the output of coefficients. Data sequence is ordered in pDst[w*128+sfb*sfbWidth[sfb]+i]. Where w is window index, `sfb` is scalefactor band index, `sfbWidth` is the scalefactor band width table, `i` is the index within scalefactor band. Buffer length is 1024. The `pDst` pointer must be aligned on an 8-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

— At least one of the following pointers: `pSrc`, `pDst`,`pWinGrpLen` is NULL.
— Either `pSrc` or `pDst` are not 8-byte aligned
— `numWinGrp` exceeds [1, 8]
— `maxSfb` exceeds [0, 51]
— `samplingRateIndex` exceeds [0, 11]
— `winLen` is not 128

- `OMX_StsAacMaxsfbErr` – the calculated scalefactor band index exceeds the `numswb` in each window

3.2.3.5.2 MDCTInv_S32_S16

Prototype

```
OMXResult omxACAAC_MDCTInv_S32_S16(OMX_S32 *pSrcSpectralCoefs, OMX_S16
   *pDstPcmAudioOut, OMX_S32 *pSrcDstOverlapAddBuf, OMX_INT winSequence,
   OMX_INT winShape, OMX_INT prevWinShape, OMX_INT pcmMode);
```

Description

This function computes an inverse MDCT to generate 1024 reconstructed 16-bit signed little-endian PCM samples as output for each channel. In order to adapt the time/frequency resolution of the filterbank to the characteristics of the input signal, a block switching tool is also adopted. For each channel, 1024 time-frequency domain samples are transformed into the time domain via the IMDCT. After applying the windowing operation, the first half of the windowed sequence is added to the second half of the previous block windowed sequence to reconstruct 1024 output samples for each channel. Output can be interleaved according to `pcmMode`.

If `pcmMode` equals 2, output is in the sequence `pDstPcmAudioOut[2*i], i`=0 to 1023, i.e., 1024 output samples are stored in the sequence: `pDstPcmAudioOut[0], pDstPcmAudioOut[2], pDstPcmAudioOut[4]`,..., `pDstPcmAudioOut[2046]`. If `pcmMode` equals 1, output is in the sequence `pDstPcmAudioOut[i], i`=0 to 1023. User must also preallocate an input-output buffer pointed by `pSrcDstOverlapAddBuf` for overlap-add operation. Reset this buffer to zero before first call, then use the output of the current call as the input of the next call for the same channel.

Reference

ISO/IEC 14496-3 Sect 4.6.10

Input Arguments

- `pSrcSpectralCoefs` – pointer to the input time-frequency domain samples in Q13.18 format. There are 1024 elements in the buffer pointed by `pSrcSpectralCoefs`.
- `pSrcDstOverlapAddBuf` – pointer to the overlap-add buffer which contains the second half of the previous block windowed sequence in Q13.18. There are 1024 elements in this buffer.
- `winSequence` – flag that indicates which window sequence is used for current block (0=long, 1=start, 2=short, 3=stop)
- `winShape` – flag that indicates which window function is selected for current block

- `prevWinShape` – flag that indicates which window function is selected for previous block
- `pcmMode` – flag that indicates whether the PCM audio output is interleaved (LRLRLR…) or not:
  1 = not interleaved
  2 = interleaved

Output Arguments

- `pDstPcmAudioOut` – Pointer to the output 1024 reconstructed 16-bit signed little-endian PCM samples in Q15, interleaved if needed.
- `pSrcDstOverlapAddBuf` – pointer to the overlap-add buffer which contains the second half of the current block windowed sequence in Q13.18.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the pointers: `pSrcSpectralCoefs, pSrcDstOverlapAddBuf and pDstPcmAudioOut` is NULL;
  - `winSequence` < 0, or `winSequence` > 3
  - `winShape` < 0, or `winShape` > 1
  - `prevWinShape` < 0, or `prevWinShape` >
  - `pcmMode` < 1, or `pcmMode` > 2

3.2.3.6 Perceptual Noise Substitution

3.2.3.6.1 DecodeMsPNS_S32_I

Prototype

```
OMXResult omxACAAC_DecodeMsPNS_S32_I (OMX_S32 *pSrcDstSpec, OMX_INT
    *pSrcDstLtpFlag, OMX_U8 *pSfbCb, OMX_S16 *pScaleFactor, OMX_INT maxSfb,
    OMX_INT numWinGrp, OMX_INT *pWinGrpLen, OMX_INT samplingFreqIndex,
    OMX_INT winLen, OMX_INT *pRandomSeed, OMX_INT channel, OMX_U8 *pMsUsed,
    OMX_INT *pNoiseState);
```

Description

Performs perceptual noise substitution for one channel across all window groups and scalefactor bands. PNS is activated for SFBs labeled in the pSfbCb vector to be of type NOISE_HCB. For PNS scalefactor bands, spectral coefficients are derived from random vectors rather than from decoded Huffman symbols.

Reference

ISO/IEC 14496-3 Sect 4.6.12

Input Arguments

- `pSrcDstSpec` – pointer to the spectral coefficient vector to which PNS should be applied

- `pSrcDstLtpFlag` – pointer to LTP used flag
- `pSfbCb` – pointer to scalefactor codebook; PNS is applied to SFBs tagged with NOISE_HCB
- `pScaleFactor` – pointer to the scalefactor value
- `maxSfb` – number of scale factor bands used
- `numWinGrp` – number of window group
- `pWinGrpLen` – pointer to the length of every window group
- `samplingFreqIndex` – sampling frequency index
- `winLen` – window length, 1024 for long, 128 for short
- `pRandomSeed` – random seed for PNS
- `channel` - index of current channel, 0:left, 1:right
- `pMsUsed` - pointer to MS used buffer from the CPE structure
- `pNoiseState` – pointer to random noise generator seed history buffer, of dimension [OMX_GROUP_NUM_MAX][OMX_AAC_SF_MAX]. If channel==0, this buffer is used only as an output and the contents upon input are ignored. If channel==1 the entries in this buffer are used to seed the PNS random number generator for each scalefactor band in which pMsUsed==1 in order to guarantee L-R correlation in those particular SFBs. Correlation is guaranteed as long as the seed entries were previously stored into this buffer during a prior call to the function with the input parameter channel==0.

Output Arguments

- `pSrcDstSpec` – pointer to updated spectral coefficient vector after completion of PNS
- `pSrcDstLtpFlag` – pointer to the LTP used flag
- `pRandomSeed` – updated PNS random seed
- `pNoiseState` – random seed buffer, of dimension [OMX_GROUP_NUM_MAX][OMX_AAC_SF_MAX]. Two possible return conditions are possible: If channel==0, this buffer returns the complete set of left channel random seeds used at the start of PNS synthesis for every scalefactor band in every group for which pSfbCb == NOISE_HCB. If channel==1 the buffer is used as an input only and the contents are unchanged from input to output.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the pointers: `pSrcDstSpec`, `pSfbCb`, `pScaleFactor`, `pWinGrpLen` or `pSrcDstLtpFlag` is NULL
  - `maxSfb` exceeds [0,51]
  - `numWinGrp` exceeds [1, 8]
  - `samplingFreqIndex` exceeds [0,12]
  - `winLen` is neither 128 nor 1024

3.2.3.7 Long-Term Prediction

3.2.3.7.1 LongTermReconstruct_S32_I

Prototype

```
OMXResult omxACAAC_LongTermReconstruct_S32_I(OMX_S32 *pSrcDstSpec, OMX_S32
    *pSrcEstSpec, OMX_INT *pLtpFlag, OMX_INT samplingFreqIndex);
```

Description

Reconstruction portion of the LTP loop; adds the vector of decoded spectral coefficients and the corresponding frequency domain vector to obtain a vector of reconstructed spectral samples.

Reference

ISO/IEC 14496-3 Sect 4.6.6

Input Arguments

- `pSrcDstSpec` – pointer to spectral coefficients to do long term prediction
- `pSrcEstSpec` – pointer to the frequency domain vector
- `samplingFreqIndex` – sampling frequency index
- `pLtpFlag` – pointer to the LTP used flag

Output Arguments

`pSrcDstSpec` – pointer to spectral coefficients have been done long term prediction

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the pointers: `pSrcDstSpec`, `pSrcEstSpec` and `pLtpFlag` is NULL
  - `winSequence` exceeds [0,3]
  - `samplingFreqIndex` exceeds [0,12]

3.2.3.7.2 MDCTFwd_S32

Prototype

```
OMXResult omxACAAC_MDCTFwd_S32(OMX_S32 *pSrc, OMX_S32 *pDst, OMX_INT
    winSequence, OMX_INT winShape, OMX_INT preWinShape, OMX_S32
    *pWindowedBuf);
```

Description

Forward MDCT portion of the LTP loop; used only for audio objects of type LTP.

Reference

ISO/IEC 14496-3 Sect 4.6.6.

Input Arguments

- `pSrc` – pointer to temporal signals to do MDCT
- `winSequence` – window sequence; (0=long, 1=start, 2=short, 3=stop)
- `winShape` – window shape shows current window's shape
- `preWinShape` – window shape shows previous window's shape
- `pWindowedBuf` – work buffer for MDCT, length of `pWindowedBuf` is at least 2048 words

Output Arguments

- `pSrcDstOverlapAdd` – reserved; not used in AAC decode
- `pDst` – output of MDCT, the spectral coefficients of PCM samples

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the pointers: `pSrc`, `pDst`, `pWindowedBuf` or `pSrcDstOverlapAdd` is NULL
  - — `winSequence` exceeds [0,3]
  - — `winShape` exceeds [0,1]
  - — `preWinShape` exceeds [0,1]

3.2.3.7.3 EncodeTNS_S32_I

Prototype

```
OMXResult omxACAAC_EncodeTNS_S32_I(OMX_S32 *pSrcDstSpectralCoefs, const
   OMX_INT *pTnsNumFilt, const OMX_INT *pTnsRegionLen, const OMX_INT
   *pTnsFiltOrder, const OMX_INT *pTnsFiltCoefRes, const OMX_S8
   *pTnsFiltCoef, const OMX_INT *pTnsDirection, OMX_INT maxSfb, OMX_INT
   profile, OMX_INT samplingRateIndex, OMX_INT winLen);
```

Description

This function applies a TNS analysis (encoding) filter to spectral coefficients in the LTP feedback loop.

Reference

ISO/IEC 14496-3 Sect 4.6.6.

Input Arguments

- `pSrcDstSpectralCoefs` – pointer to the unprocessed spectral coefficient vector; samples are represented using Q18.13

- `pTnsNumFilt` – pointer to a table containing the number of TNS filters that are applied on each window of the current frame. The table elements are indexed as follows:

  — `pTnsNumFilt[w]`

  w=0 to numWin-1; depending upon the current window sequence, this vector may contain up to 8 elements
- `pTnsRegionLen` – pointer to a table containing TNS region lengths (in scalefactor band units) for all regions and windows on the current frame; the table entry `pTnsRegionLen[i]` specifies the region length for k-th filter on the w-th window. The table index, i, is computed as follows:

$$i = \sum_{j=0}^{w-1} \mathtt{pTnsNumFilt}[j] + k$$

  where 0 <= w <= `numWin`-1, and 0 <= k <= `pTnsNumFilt`[w]-1.
- `pTnsFiltOrder` – pointer to a table containing TNS filter orders for all regions and windows on the current frame; the table entry `pTnsFiltOrder[i]` specifies the TNS filter order for the k-th filter on the w-th window. The table index, i, is computed as follows:

$$i = \sum_{j=0}^{w-1} \mathtt{pTnsNumFilt}[j] + k$$

  where 0 <= w <= `numWin`-1, and 0 <= k <= `pTnsNumFilt`[w]-1.
- `pTnsFiltCoefRes` – pointer to a table of TNS filter coefficient resolution indicators for each window on the current frame. Resolutions for filters on the w-th window are specified in table entry `pTnsFiltCoefRes[w]`, and `w=0` to `numWin-1`.
- `pTnsFiltCoef` – pointer to a table containing the complete set of TNS filter coefficients for all windows and regions on the current frame. Filter coefficients are stored contiguously in filter-major order, i.e., the table is organized such that the filter coefficients for the k-th filter of the w-th window are indexed using `pTnsFiltCoef[w][k][i]`, where `0` <= `i` <= `pTnsFiltOrder[j]-1, 0` <= k <= `pTnsNumFilt[w]-1`, 0 <= `w` <= `numWin-1`, and the filter order index j is computed as shown above.
- `pTnsDirection` – pointer to a table of tokens that indicate the direction of TNS filtering for all regions and windows on the current frame, with 0 indicating upward and 1 indicating downward; in particular the table entry `pTnsDirection[i]` specifies direction for k-th filter on the w-th window, and the table index, i, is computed as follows:

$$i = \sum_{j=0}^{w-1} \mathtt{pTnsNumFilt}[j] + k$$

  where 0 <= w <= numWin-1, and 0 <= k <= pTnsNumFilt[w]-1.
- `maxSfb` – number of scalefactor bands
- `profile` – audio profile
- `samplingRateIndex` – sampling rate index
- `winLen` – window length

Output Arguments

- `pSrcDst` – pointer to the TNS-encoded spectral coefficient vector; samples are represented using Q18.13

Returns

- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the pointers: `pSrcDst`, `pTnsNumFilt`, `pTnsRegionLen`, `pTnsFiltOrder`, `pTnsFiltCoefRes`, `pTnsFiltCoef` or `pTnsDirection` is NULL
  - — `maxSfb` exceeds [0,51]; `winLen` is neither 128 nor 1024; `samplingRateIndex` exceeds [0,12]

3.2.3.7.4 LongTermPredict_S32

Prototype

```
OMXResult omxACAAC_LongTermPredict_S32(OMX_S32 *pSrcTimeSignal, OMX_S32
    *pDstEstTimeSignal, OMXAACLtpInfo *pAACLtpInfo, OMX_INT winSequence);
```

Description

LTP analysis portion of the LTP loop.

Reference

ISO/IEC 14496-3 Sect 4.6.6.

Input Arguments

- `pSrcTimeSignal` – pointer to the temporal signals to be predicted in temporary domain
- `pDstEstTimeSignal` – pointer to the output of samples after LTP
- `pAACLtpInfo` – pointer to the LTP information
- `winSequence` – window type; (0=long, 1=start, 2=short, 3=stop)

Output Arguments

- `pDstEstTimeSignal` – pointer to the output of prediction in time domain

Returns

- `OMX_StsBadArgErr` – bad arguments
  - — The pointer `pSrcDstTime` is NULL
  - — `winSequence` exceeds [0,3]

3.2.3.8 Huffman Decoding

3.2.3.8.1 NoiselessDecode

Prototype

```
OMXResult omxACAAC_NoiselessDecode(const OMX_U8 **ppBitStream, OMX_INT
    *pOffset, OMX_S16 *pDstScalefactor, OMX_S32 *pDstQuantizedSpectralCoef,
    OMX_U8 *pDstSfbCb, OMX_S8 *pDstTnsFiltCoef, OMXAACChanInfo *pChanInfo,
    OMX_INT commonWin, OMX_INT audioObjectType, OMXAACLtpInfoPtr *pLtpInfo);
```

Description

Noiseless (Huffman) decoder for MPEG-2 and MPEG-4 AAC LC and LTP audio objects.

Reference

ISO/IEC 14496-3 Sect 4.6.3.

Input Arguments

- `ppBitStream` – double pointer to current byte in the input bitstream
- `pOffset` – pointer to the offset indicating the next available bit in the current byte of the input bitstream
- `pChanInfo` – pointer to channel information structure
- `commonWin` – commonWin==1 indicates that the channel pair uses the same ics information; commonWin==0 indicates that ics information is not shared
- `audioObjectType` – audio object type indicator: 2=LC, 4=LTP
- `pLtpInfo` – pointer to array containing pointers to LTP information structures for each channel

Output Arguments

- `ppBitStream` – double pointer to the updated stream pointer; references the current byte in the input bitstream after Huffman decoding has been completed
- `pOffset` – pointer to the updated bit index indicating the next available bit in the input stream following after Huffman decoding has been completed
- `pChanInfo` – pointer to the updated channel information structure
- `pDstScalefactor` – pointer to the updated scale factor table
- `pDstQuantizedSpectralCoef` – pointer to the decoded, quantized spectral coefficients
- `pDstSfbCb` – pointer to the updated scalefactor codebook index
- `pDstTnsFiltCoef` – pointer to the updated array of TNS filter coefficients
- `pLtpInfo` – pointer to array containing pointers to updated LTP information structures for each channel containing the LTP information decoded from the input stream

Returns

- `OMX_StsNoErr` – no error

- `OMX_StsBadArgErr` – bad arguments
  - At least one of the pointers: `ppBitStream`, `pOffset`, `*ppBitStream`, `pDstScaleFactor`, `pDstTnsFiltCoef`, `pDstQuantizedSpectralCoef`, `pChanInfo` or `pDstSfbCb` is NULL
  - `*pOffset` exceeds [0,7]
  - `winSequence` exceeds [0,3]; maxSfb exceeds [0,51]
  - `commonWin` exceeds [0,1]
  - `scaleFlag` exceeds [0,1]
  - `audioObjectType` exceeds [0,16]
- `OMX_StsAacStsDataErr -` The `pChanInfo`->`pIcsInfo`->winSequence indicates short sequence and pChanInfo->pulsePres indicates pulse data present. The start scalefactor band for pulse data >= pChanInfo->numSwb. Or Pulse data position offset >= winLen.
- `OMX_StsAacGainCtrErr - pChanInfo`->`gainControlPres` is decoded as 1 that means gain control data present.

Image Processing

4

This section describes the functions and data types that comprise the OpenMAX DL image processing domain (omxIP) API, including functions that support bitmap manipulation (omxIPBM), image pre- and post-processing (omxIPPP), and color space conversion (omxIPCS).

4.1 Common Definitions

This section defines constants, data structures, buffer organizations, and image processing conventions that shall be followed for all image processing sub-domains.

4.1.1 Image Representation

The image processing domain supports absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-orientation (also called interleaved format) or plane-oriented (also called planar format). For images in pixel-oriented, all channel values for each pixel are clustered and stored consecutively. For example, BGRBGRBGR...... for an RGB image. The number of channels in a pixel-oriented image can be one, two, or three (the fourth channel alpha channel is not currently supported) and is identified by the function name descriptor C1, C2, or C3. As a special case, the C2 designator is used with pixel-oriented YCbCr422 data even though the Cb and Cr channels are strictly speaking distinct and there are in fact three channels. This convention is adopted because the pixels are organized in memory as shown in Table 4-1, i.e., Y-Cb-Y-Cr. Also as shown in Table 4-1, the name RGB indicates that the data are stored in BGR order.

For example, in function

```
omxIPCS_RGBToYUV_U8_C3R(const OMX_U8* pSrc, OMX_INT srcStep, OMX_U8*
pDst, OMX_INT dstStep, OMXSize roiSize),
```

both input and output are in C3 format. The input pointer `pSrc` will point to data formatted as BGRBGRBGR......, and similarly output pointer `pDst` will point to data formatted as YUVYUVYUV....

For planar images, all image data for each channel is stored contiguously. Functions that operate on planar images are identified by the presence of P3 descriptor. In this case, three pointers (one for each plane) are specified.

For example, in function

```
omxIPCS_RGBToYUV_U8_C3P3R( const OMX_U8* pSrc, OMX_INT srcStep,
                           OMX_U8* pDst[3], OMX_INT dstStep,
                            OMXSize roiSize )
```

the descriptor "`C3P3`" means that the input is in pixel-oriented format (3 channels) and output is in planar format (3 channels). Therefore input pointer `pSrc` will point to data block BGRBGRBGR....... The output pointer `pDst[0]` will point to data block YYY......, the output pointer `pDst[1]` will point to data block UUU....., and the output pointer `pDst[2]` will point to data block VVV.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers. The data type may be signed (s) or unsigned (u). All channels in an image must have the same data type.

For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in pixel mode. This data type is identified in function names as `U8_C3` descriptor, where `U8` represents 8-bit unsigned data for each channel and C3 represents three channels.

The tables below define how buffers in memory are organized for the interleaved and planar representations of the various color spaces supported in the omxIP, omxIC, and omxVC domains. These memory organizations shall be followed unless otherwise specified in the description for a particular function or function set.

Table 4-1: Memory Organization for Interleaved (Pixel-Oriented) Color Space Data

Color Space	Byte Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RGB565	Base+0x0	G	G	G	B	B	B	B	B
	Base+0x1	R	R	R	R	R	G	G	G
RGB444	Base+0x0	G	G	G	G	B	B	B	B
	Base+0x1	0	0	0	0	R	R	R	R
RGB555	Base+0x0	G	G	G	B	B	B	B	B
	Base+0x1	0	R	R	R	R	R	G	G
RGB888	Base+0x0	B	B	B	B	B	B	B	B
	Base+0x1	G	G	G	G	G	G	G	G
	Base+0x2	R	R	R	R	R	R	R	R
YCbCr444	Base+0x0	Y	Y	Y	Y	Y	Y	Y	Y
	Base+0x1	Cb	Cb	Cb	Cb	Cb	Cb	Cb	Cb
	Base+0x2	Cr	Cr	Cr	Cr	Cr	Cr	Cr	Cr
YCbCr422	Base+0x0	Y_0	Y_0	Y_0	Y_0	Y_0	Y_0	Y_0	Y_0
	Base+0x1	Cb_{01}	Cb_{01}	Cb_{01}	Cb_{01}	Cb_{01}	Cb_{01}	Cb_{01}	Cb_{01}
	Base+0x2	Y_1	Y_1	Y_1	Y_1	Y_1	Y_1	Y_1	Y_1
	Base+0x3	Cr_{01}	Cr_{01}	Cr_{01}	Cr_{01}	Cr_{01}	Cr_{01}	Cr_{01}	Cr_{01}
	Base+0x4	Y_2	Y_2	Y_2	Y_2	Y_2	Y_2	Y_2	Y_2
	Base+0x5	Cb_{23}	Cb_{23}	Cb_{23}	Cb_{23}	Cb_{23}	Cb_{23}	Cb_{23}	Cb_2
	Base+0x6	Y_3	Y_3	Y_3	Y_3	Y_3	Y_3	Y_3	Y_3
	Base+0x7	Cr_{23}	Cr_{23}	Cr_{23}	Cr_{23}	Cr_{23}	Cr_{23}	Cr_{23}	Cr_{23}

Note on YCbCr422 organization: The entries in the YCbCr422 table correspond to pixels as they appear in raster-scan order, i.e., the Y_0 pixel occupies the left-most position, followed by Y_1, Y_2, Y_3, … scanning from left to right, and the subscripts are intended to convey the associations between luminance and sub-sampled chrominance components. The luminance pixels Y_0 and Y_1 are paired with the sub-sampled chrominance pixels Cb_{01} and Cr_{01}, and the luminance pixels Y_2 and Y_3 are paired with the sub-sampled chrominance pixels Cb_{23} and Cr_{23}.

Table 2: Memory Organization for Planar Color Space Data

YCbCrxxx	pSrc[0]/pDst[0]	Y block
	pSrc[1]/pDst[1]	Cb block
	pSrc[2]/pDst[2]	Cr block

4.1.2 Image Processing Models

Most omxIP functions perform identical and independent operations on all channels of the processed image. The same operation is applied to each channel, and the computed results do not depend upon values of other channels. The only exceptions are the color conversion functions, which process three channels together.

4.1.3 Neighborhood Operations

The result of a neighborhood operation is based on values of a certain group of pixels, located near a given input pixel. The set of neighboring pixels is typically defined by the size of rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel.

The omxIP functions that process a neighborhood operate on the assumption that all referred points of the image are available. To support this mode, the application must check that ROI parameters passed to the function have such values that all processed neighborhood pixels actually exist in the image.

The following are examples of functions that perform neighborhood operations.

- `omxIPPP_FilterMedian`

4.1.4 Rectangle or Region of Interest

Some omxIP functions can operate not only on entire images but also on a part of the image. The Region of Interest or Rectangle of Interest (ROI) are rectangular areas which may be either some part of the image or the whole image.

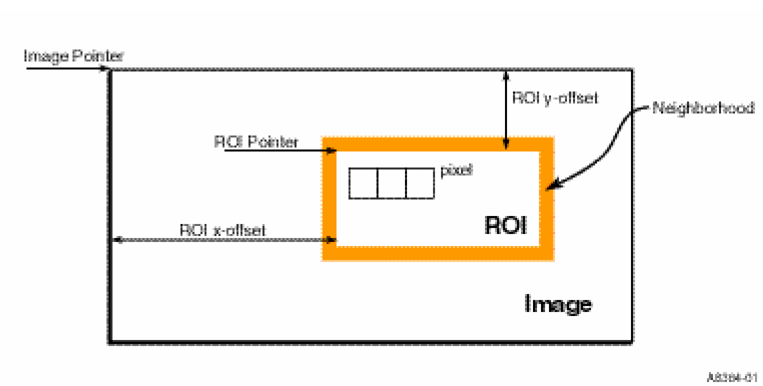
The omxIP functions with ROI support are distinguished by the presence of an R descriptor in their names. ROI of an image is defined by the size and offset from the image's origin as shown in Figure 4-1. The origin of an image is implied to be in the top left corner, with x values increasing from left to right and y values increasing downwards.

Both the source and destination images can have a rectangle of interest. In such cases, the sizes of ROIs are assumed to be the same while offsets may differ. The image processing is then performed on data of the source ROI, and the results are written to the destination ROI. In function call sequences, an ROI is specified by:

- `roiSize` argument of the `OMXSize` type
- `pSrc` and `pDst` pointers to the starts of source and destination ROI buffers
- `srcStep` and `dstStep` arguments which are equal to distances in bytes between the starts of consecutive lines in source and destination images, respectively.

Thus, the arguments `srcStep`, `dstStep` set steps in bytes through image buffers to start processing a new line in the ROI of an image

Figure 4-1: Image, ROI, and Offsets



4.1.5 Data Types and Enumerators

4.1.5.1 Rotations

`OMXIPRotation`, a data type that enumerates image rotations, is defined as follows:

```
typedef enum {
   OMX_IP_DISABLE = 0,
   OMX_IP_ROTATE90L = 1,            /* counter-clockwise */
   OMX_IP_ROTATE90R = 2,            /* clockwise */
   OMX_IP_ROTATE180 = 3,
   OMX_IP_FLIP_HORIZONTAL = 4,      /* from R to L, about V axis */
   OMX_IP_FLIP_VERTICAL = 5,        /* from top to bottom, about H axis */
} OMXIPRotation;
```

Counter-clockwise rotation is denoted by the “L” postfix, and clockwise rotation is denoted by the “R” postfix. A horizontal flip creates a “mirror” image with respect to the vertical image axis, i.e.,

 horizontal flip ,

and a vertical flip creates a “mirror” image with respect to the horizontal image axis, i.e.,

 vertical flip 

4.1.5.2 Rectangle

The structure OMXRect, used for storing the geometric position and size of a rectangle, is defined as follows:

```
typedef struct {
   OMX_INT x;
   OMX_INT y;
   OMX_INT width;
   OMX_INT height;
} OMXRect;
```

where the points x and y specify the coordinates of the top left corner of the rectangle, and the parameters width and height specify dimensions in the x- and y- directions, respectively.

4.1.5.3 Point

The structure OMXPoint is used to represent the geometric position of a point, is defined as follows:

```
typedef struct {
   OMX_INT x;
```

```
    OMX_INT y;
} OMXPoint;
```

where x, y define the coordinates of the point.

4.1.5.4 Size

The structure OMXSize, used for storing the size of a rectangular region, is defined as follows:

```
typedef struct {
    OMX_INT width;
    OMX_INT height;
} OMXSize;
```

where width and height denote the dimensions of the rectangle in the x- and y directions, respectively.

4.2 Bitmap Manipulation Sub-Domain (omxIPBM)

This section defines functions that perform image data set and initialization operations, including functions that support bitmap copy, add, and multiply operations.

Bitmap arithmetic operators are defined in the table below.

Table 4-3: Arithmetic Operators

Operation	Description
AddC	Add a constant with scaling and saturation, i.e., $Z_{n,m} = SAT_{U8}((x_{n,m} + c) * 2^{-S})$ where the parameter x is the input image, the parameters n and m are the pixel indices, the parameter c is the constant value to be added to each pixel, the parameter S is the scalefactor, SAT denotes saturation to an unsigned 8-bit result, and the parameter Z is the output image.
MulC	Multiply by a constant with scaling and saturation, i.e., $Z_{n,m} = SAT_{U8}((x_{n,m} * c) * 2^{-S})$ where the parameter x is the input image, the parameters n and m are the pixel indices, the parameter c is the constant value by which each pixel is multiplied, the parameter S is the scalefactor, SAT denotes saturation to an unsigned 8-bit result, and the parameter Z is the output image.

4.2.1 Functions

4.2.1.1 Block Copy

4.2.1.1.1 Copy_U8_C1R

4.2.1.1.2 Copy_U8_C3R

Prototype

```
OMXResult omxIPBM_Copy_U8_C1R(const OMX_U8* pSrc, OMX_INT srcStep, OMX_U8*
    pDst, OMX_INT dstStep, OMXSize roiSize);
OMXResult omxIPBM_Copy_U8_C3R(const OMX_U8* pSrc, OMX_INT srcStep, OMX_U8*
    pDst, OMX_INT dstStep, OMXSize roiSize);
```

Description

Copy pixel values from the ROI of the source image pointed `pSrc` to the ROI of the destination image `pDst`.

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

The following example describes a simple initialization routine. The following example describes how to copy one image to another.

Example 4-1: Copying One Image to Another

```
#include "extools.h" /* tool for print/display */
main()
{
  // One-channel source and destination images
  const OMX_INT SIDE = 16;
  OMX_U8 src[SIDE*SIDE];
  OMX_U8 dst[SIDE*SIDE];
  OMXSize imgSize = {SIDE,SIDE};

  // Init source image here

  // Specify ROI location and size
  OMXPoint srcROIlocation = {2,2};
  OMXPoint dstROIlocation = {1,1};
  OMXSize dstROIsize = {7,7};

  // Copy source ROI to the destination
  omxIPBM_Copy_U8_C1R(AddressOf(src,SIDE,srcROIlocation), SIDE,
  AddressOf(dst,SIDE,dstROIlocation), SIDE, dstROIsize);

  // Print result
  OMXSize showSize = {9,9};
```

```
    PrintROI_C1(_T("Example: OMXCopy"), dst, SIDE, showSize);
    return 0;
  }
  //
  // Returns address of ROI
  //
  OMX_U8* AddressOf(OMX_U8* pImg, OMX_INT sLine, OMXPoint aLocation)
  {
    return pImg +sLine*aLocation.y +aLocation.x;
  }
```

4.2.1.2 Arithmetic

4.2.1.2.1 AddC_U8_C1R_Sfs

4.2.1.2.2 MulC_U8_C1R_Sfs

Prototype

```
OMXResult omxIPBM_AddC_U8_C1R_Sfs(const OMX_U8* pSrc, OMX_INT srcStep,
    OMX_U8 value, OMX_U8* pDst, OMX_INT dstStep, OMXSize roiSize, OMX_INT
    scaleFactor);
OMXResult omxIPBM_MulC_U8_C1R_Sfs(const OMX_U8* pSrc, OMX_INT srcStep,
    OMX_U8 value, OMX_U8* pDst, OMX_INT dstStep, OMXSize roiSize, OMX_INT
    scaleFactor);
```

Description

Computes corresponding arithmetic operation with a constant and each element of image and places the scaled result in the same image.

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `value` – constant for operation
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels
- `scaleFactor` – scale factor value

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – No error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

4.2.1.3 Mirror

4.2.1.3.1 Mirror_U8_C1R

Prototype

```
OMXResult omxIPBM_Mirror_U8_C1R(const OMX_U8 *pSrc, OMX_INT srcStep, OMX_U8
   *pDst, OMX_INT dstStep, OMXSize roiSize, OMXIPRotation axis);
```

Description

This function mirrors the source image `pSrc` about a horizontal or vertical axis or both, depending on the flip value, and writes it to the destination image `pDst`.

Input Arguments

- `pSrc` – pointer to the source buffer
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels.
- `flip` – specifies the axis about which to mirror the image. Must use one of the following `OMXIPRotation` values:
  - `OMX_IP_FLIP_HORIZONTAL` to mirror about the vertical axis
  - `OMX_IP_FLIP_VERTICAL` to mirror about the horizontal axis
  - `OMX_IP_ROTATE180` to mirror about both horizontal and vertical axes

Output Arguments

- `pDst` – pointer to the destination buffer

Returns

- `OMX_StsNoErr` – No error. Any other value indicates an error or a warning
- `OMX_StsNullPtrErr` – indicates an error condition if `pSrc` or `pDst` pointer is NULL
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value
- `OMX_StsStepErr` – indicates an error condition if `srcStep` or `dstStep` has a zero or negative value
- `OMX_StsMirrorFlipErr` – indicates an error condition if flip has an illegal value

4.3 Pre- and Post-Processing Sub-Domain (omxIPPP)

This section defines functions that perform image pre- and post-processing operations.

4.3.1 Functions

4.3.1.1 Filtering

This section describes image processing functions that perform linear and non–linear filtering operations on an image. The tables below provide detailed mathematical definitions of supported filtering operations.

Table 4-4: FIR Filtering Definition

Operation	Description
General FIR filtering	Filters an image using a general rectangular convolution kernel. Basic equation of general 2D filter is the following: $$y_{n,m} = \Psi_{2D}(h, a, x) \equiv \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} h_{i,j} \cdot x_{n+ay-i,m+ax-j}$$ where: h 2D kernel of *WxH* size with anchor location *(ay, ax)* $X_{n,m}$ and $y_{n,m}$ are input and output signals respectively.

Table 4-5: Median Filtering Definition

Operation	Description
single–channel (gray) image	Filters an image using a median filter, defined as $$y = \underset{(WxH)}{MED}(x, a, ROI)$$ where the parameter *x* is the input image, the parameter *y* is the output image, the parameter *a* specifies the anchor, the parameters *W* and *H* specify the width and height, respectively, of the mask, the parameter ROI specifies the region of interest in the input image *x*, and the parameter *a* defines the mask anchor point, which is specified in terms of coordinates *a.x* and *a.y*. The median filter operator, *MED*, generates an output image of dimension WxH. Each output pixel takes the median value of the masked region associated with the corresponding pixel from the input ROI. Median values are computed as follows: for each ROI input pixel, the top-left corner of the mask is offset and overlayed according to (a.x, a.y). Next, the median is identified within the masked region. For a mask of dimension *W*x*H*, the *W*H* pixels are ordered in terms of intensity from smallest to largest and the value of the median (middle) element is returned as output. If *W*H* is odd then the median will be the (*W*H*+1)/2 entry in the list of ordered pixels. The median filter operation can also be described in terms of a distance minimization. In particular, the median filtered output pixels $y_{n,m}$ can be expressed in terms of the input pixels, $x_{n,m}$, as follows $$\underset{(WxH)}{MED}(x, a, ROI) \equiv y_{n,m} = x_{n-a.y+r_M(n,m),m-a.x+c_M(n,m)}$$ where the parameters a.x and a.y define the anchor point, the parameters n and m are the indices of the input and output images, and the parameters $r_M(n,m)$ and $c_M(n,m)$ reflect, respectively, the values of the parameters r and c that minimize for the pixel in location n,m the magnitude of the distance measure $\Psi_{r,c}$, defined as $$\Psi_{r,c} = \sum_{i=0}^{H-1}\sum_{j=0}^{W-1}\left|x_{n-a.y+r,m-a.x+c} - x_{n-a.y+i,m-a.x+j}\right|$$ The search is bounded by the mask region, i.e., $0 \le r \le H-1$ and $0 \le c \le W-1$.

4.3.1.1.1 FilterFIR_U8_C1R

Prototype

```
OMXResult omxIPPP_FilterFIR_U8_C1R(const OMX_U8* pSrc, OMX_INT srcStep,
    OMX_U8* pDst, OMX_INT dstStep, OMXSize roiSize, const OMX_S32* pKernel,
    OMXSize kernelSize, OMXPoint anchor, OMX_INT divider);
```

Description

Performs filtering of the ROI of the source image pointed to by `pSrc` using a general rectangular (WxH size) convolution kernel. The value of the output pixel is normalized by the divider and saturated as:

$$SAT_{U8}\left(\frac{1}{divider}\Psi_{2D}(h,a,x)\right)$$

The result is placed into the ROI of the destination image pointed to by `pDst`.

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels
- `pKernel` – pointer to the 2D FIR filter coefficients
- `kernelSize` – size of the FIR filter kernel. The minimum valid size is 1x1. There is no limit on the maximum size other than the practical limitation imposed by the ROI size and location relative to the image boundaries. The caller should avoid kernel overlap with invalid buffer locations given ROI size, ROI placement relative to the image buffer boundaries, and the FIR operator definition given in Table 4-4.
- `anchor` – anchor cell specifying the alignment of the array of filter taps with respect to the position of the input pixel
- `divider` – value of the divider used to normalize the result

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – No error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value
- `OMX_StsAnchorErr` – the anchor point is outside mask
- `OMX_StsScaleRangeErr` – scale bounds is out of range

Example 4-2: FIR Image Filtering

```
#include <stdlib.h>
#include "extools.h" /* print/display tool */
main()
{
  // One-channel source and destination images
  const OMX_INT SIDE = 16;
  OMX_U8 src[SIDE*SIDE];
  OMX_U8 dst[SIDE*SIDE];
  // Fill source image with data
  for(OMX_INT i=0; i<SIDE*SIDE; i++)
    src[i] = (OMX_U8)(rand()&0xFF);
  // 3x3 filter with anchor at the middle
  const OMX_INT FSIDE = 3;
  OMXSize filterSize = {FSIDE,FSIDE};
  OMXPoint anchor = {1,1};
  OMX_S32 filter[FSIDE][FSIDE] = {
        {1,1,1},
        {1,1,1},
        {1,1,1}
  };
  // Specify ROI location and size
  OMXPoint srcLocation = {1,1};
  OMXPoint dstLocation = {0,0};
  OMXSize roiSize = {7,7};
  // Apply 2D averaging FIR Filter
  omxIPPP_Filter_U8_C1R(AddressOf(src,SIDE,srcLocation),SIDE,
  AddressOf(dst,SIDE,dstLocation),SIDE,roiSize,
           (OMX_S32*)filter,filterSize,anchor,
           FSIDE*FSIDE);
  // Print result
  OMXSize showSize = {9,9};
  PrintROI_C1(_T("Example: OMXFilter (source)"), src, SIDE,
              showSize);
  PrintROI_C1(_T("Example: OMXFilter (destination)"), dst, SIDE,
              roiSize);
  return 0;
}
```

```
//
// Returns address of ROI
//
OMX_U8* AddressOf(OMX_U8* pImg, OMX_INT sLine, OMXPoint aLocation)
{
  return pImg +sLine*aLocation.y +aLocation.x
}
```

4.3.1.1.2 FilterMedian_U8_C1R

Prototype

```
OMXResult omxIPPP_FilterMedian_U8_C1R(const OMX_U8* pSrc, OMX_INT srcStep,
   OMX_U8* pDst, OMX_INT dstStep, OMXSize roiSize, OMXSize maskSize,
   OMXPoint anchor);
```

Description

Performs median filtering of the ROI of the source image pointed to by `pSrc` using the median filter of the size `maskSize` and location `anchor`, and places the result into the ROI of the destination image pointed to by `pDst`.

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI, in pixels
- `maskSize` – size of the mask, in pixels; minimum size is 3x3; maximum size is 255x255.
- `anchor` – anchor cell specifying the mask alignment with respect to the position of the input pixels

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – No error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value
- `OMX_StsMaskSizeErr` – invalid mask size, i.e., smaller than 3x3 or larger than 255x255
- `OMX_StsAnchorErr` – the anchor point is outside mask

4.3.1.2 Statistical

This section describes functions that compute statistical image moments. The associated mathematical definitions are given in Table 4-6.

Table 4-6: Statistical Moments Definitions

Operation	Description
spatial moment	The spatial moment of (p,q) order is defined as follows: $$m_{pq} = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} x^p y^q f(x,y)$$, where: $f(x,y)$ input image pixel at the (x,y) location , and the infinite summation limits indicate that the sum is accumulated over all rows and columns in the image. Reference: *Digital Image Processing Methods*, p. 432, Marcel Dekker, E. Dougherty, Editor, 1994.
central moment	Basic equation of central moment of (p,q) order is the following: $$\mu_{pq} = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} (x-\hat{x})^p (y-\hat{y})^q f(x,y)$$ $\hat{x} = \frac{m_{10}}{m_{00}}$ and $\hat{y} = \frac{m_{01}}{m_{00}}$ are coordinates of the center of mass , and the parameters m_{01}, m_{10}, and m_{00} are the order (1,0), (0,1), and (0,0), spatial moments, respectively. Reference: *Digital Image Processing Methods*, p. 432, Marcel Dekker, E. Dougherty, Editor, 1994.

4.3.1.2.1 MomentGetStateSize_S64

Prototype

```
OMXResult omxIPPP_MomentGetStateSize_S64(OMX_INT* pSize);
```

Description

Get size of state structure in bytes; returned in `*pSize`.

Output Arguments

- `pSize` – pointer to the size of structure

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – "NULL" input pointer received

4.3.1.2.2 MomentInit_S64

Prototype

```
OMXResult omxIPPP_MomentInit_S64(OMXMomentState_S64* pState);
```

Description

Initialize moment state structure.

Input Arguments

- `pState` – pointer to the uninitialized state structure

Output Arguments

- `pState` – pointer to the initialized state structure

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error

4.3.1.2.3 Moments_U8_S64_C1R

4.3.1.2.4 Moments_U8_S64_C3R

Prototype

```
OMXResult omxIPPP_Moments_U8_S64_C1R(const OMX_U8* pSrc, OMX_INT srcStep,
   OMXSize roiSize, OMXMomentState_S64* pState);
OMXResult omxIPPP_Moments_U8_S64_C3R(const OMX_U8* pSrc, OMX_INT srcStep,
   OMXSize roiSize, OMXMomentState_S64* pState);
```

Description

Computes statistical spatial moments of order 0 to 3 for the ROI of the image pointed to by `pSrc`.

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `roiSize` – size of the ROI in pixels

Output Arguments

- `pState` – pointer to the state structure

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value
- `OMX_StsContextMatchErr` – contents of the implementation-specific structure `OMXMomentState_S64` are invalid

4.3.1.2.5 GetSpatialMoment_S64

Prototype

```
OMXResult omxIPPP_GetSpatialMoment_S64(const OMXMomentState_S64* pState,
    OMX_INT mOrd, OMX_INT nOrd, OMX_INT nChannel, OMXPoint roiOffset,
    OMX_S64* pValue, OMX_INT scaleFactor);
```

Description

Returns `nOrd` by `mOrd` spatial moment calculated by the `Moments_U8_S64` function. Places the scaled result into the memory pointed to by `pValue`.

Input Arguments

- `nOrd, mOrd` - moment specifiers
- `pState` – pointer to the state structure
- `nChannel` – specifies the desired image channel from which to extract the spatial moment. For a C3 input image, the valid range is from 0-2. For a C1 input image, the only valid value is 0.
- `roiOffset` – offset in pixels of the ROI origin (top left corner) from the image origin
- `scaleFactor` – value of the scale factor

Output Arguments

- `pValue` - pointer to the computed moment value

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsContextMatchErr` – contents of the implementation-specific structure `OMXMomentState_S64` are invalid
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value
- `OMX_StsChannelErr` – illegal channel number

4.3.1.2.6 GetCentralMoment_S64

Prototype

```
OMXResult omxIPPP_GetCentralMoment_S64(const OMXMomentState_S64* pState,
    OMX_INT mOrd, OMX_INT nOrd, OMX_INT nChannel, OMX_S64* pValue, OMX_INT
    scaleFactor);
```

Description

Returns the `nOrd` by `mOrd` central moment calculated by the `Moments_U8_S64` function, and places the scaled result into the memory pointed to by `pValue`.

Input Arguments

- `pState` – pointer to the state structure
- `mOrd,nOrd` – specify the required spatial moment
- `nChannel` – specifies the desired image channel from which to extract the spatial moment. For a C3 input image, the valid range is from 0-2. For a C1 input image, the only valid value is 0.
- `scaleFactor` – value of the scale factor

Output Arguments

- `pValue` – pointer to the computed moment value

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsContextMatchErr` – contents of the implementation-specific structure `OMXMomentState_S64` are invalid
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value
- `OMX_StsChannelErr` – illegal channel number

4.3.1.3 Deblocking

4.3.1.3.1 Deblock_HorEdge_U8_I

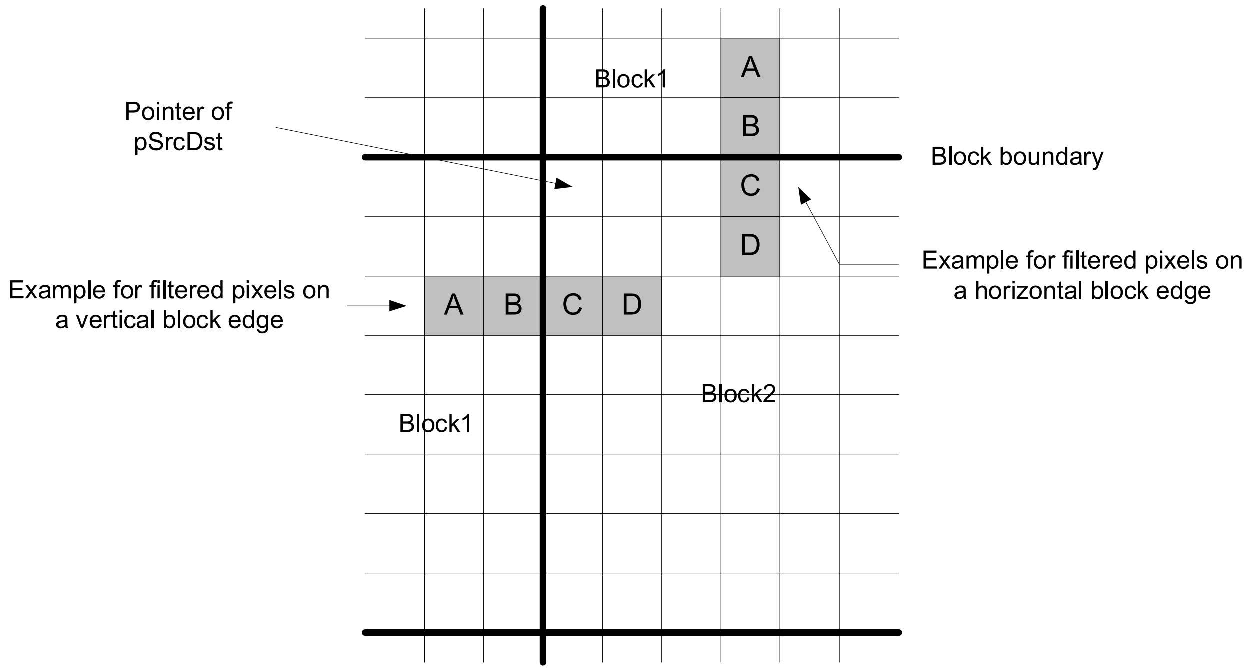
4.3.1.3.2 Deblock_VerEdge_U8_I

Prototype

```
OMXResult omxIPPP_Deblock_HorEdge_U8_I (OMX_U8 *pSrcDst, OMX_INT step,
    OMX_INT QP);
OMXResult omxIPPP_Deblock_VerEdge_U8_I (OMX_U8 *pSrcDst, OMX_INT step,
    OMX_INT QP);
```

Description

Performs deblock filtering for a single 8x8 macroblock along a block edge (horizontal-top or vertical-left), as shown in the figure below. Block edges are represented in the figure by heavy lines. The horizontal edge deblocking function processes the top edge of the block referenced by `pSrcDst`. The vertical edge deblocking function processes the left edge of the block referenced by `pSrcDst`. For each processed column , the horizontal edge deblocking operation modifies two pixels from the source block (pixels C,D) and two pixels in the neighboring block (pixels A,B). For each processed row, the vertical edge deblocking operation modifies two pixels from the source block (pixels C,D) and two pixels from the neighboring block (A,B).



Input Arguments

- `pSrcDst` – pointer to the first pixel of the second block (labeled “block 2” in the figure)
- `step` – distance, in bytes; between start of each line; must be a multiple of 8
- `QP` – quantization parameter, as described in Section J.3 of Annex J in H.263+

Output Arguments

- `pSrcDst` – pointer to the first pixel of the second output block (labeled “block 2” in the figure below) Returns
- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — `pSrcDst` is NULL
  - — `pSrcDst` is not 64-bit aligned
  - — `QP` exceeds [1,31]
  - — `Step` is not a multiple of 8 or step is less than 8

4.4 Color Space Conversion Sub-Domain (omxIPCS)

This section defines functions that perform color space conversion. Table 4-7 presents analytical expressions that define each of the color conversion methodologies referenced in the omxIPCS sub-domain. In addition, color space subsampling conventions that shall be observed are summarized in Table 4-8.

4.4.1 Definitions

4.4.1.1 Color Space Conversions

Table 4-7: Color Model Conversions

Functions	Mathematical Descriptions
Color Twist	Linear transform of an original Color model (RGB) to another (ABC) by user defined transformation $$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} T_{00} T_{01} T_{02} \\ T_{10} T_{11} T_{12} \\ T_{20} T_{21} T_{22} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} T_{03} \\ T_{13} \\ T_{23} \end{bmatrix}$$
RGB to YCbCr	RGB color model is transformed to YCbCr Color model as follows: $Y = 0.257R + 0.504G + 0.098B + 16$ $Cb = -0.148R - 0.291G + 0.439B + 128$ $Cr = 0.439R - 0.368G - 0.071B + 128$ Note: OpenMAX DL color conversion equations are implemented using integer data types. Reference: International Telecommunications Union (ITU-T), Rec. BT.601
YCbCr to RGB	YCbCr color model is transformed to RGB Color model as follows: $R = 1.164(Y - 16) + 1.596(Cr - 128)$ $G = 1.164(Y - 16) - 0.813(Cr - 128) - 0.392(Cb - 128)$ $B = 1.164(Y - 16) + 2.017(Cb - 128)$

Note: OpenMAX DL color conversion equations are implemented using integer data types.

Reference: International Telecommunications Union (ITU-T), Rec. BT.601

4.4.1.2 Color Space Subsampling

Table 4-8: Color Conversion Subsampling Conventions

Image Type	Downsampling	Description
4:4:4 YCbCr	None	Y, Cb, Cr sampled on every pixel. 8 bits per component = 24 bits per pixel.
4:2:2 YCbCr	2:1 horizontal	Y sampled on every pixel; Cb and Cr sampled every 2 pixels horizontally. 8 bits per component = 32 bits per pixel pair.
4:1:1 YCbCr	4:1 horizontal	Y sampled on every pixel; Cb and Cr sampled every 4 pixels horizontally. 8 bits per component = 48 bits for four pixels.
4:2:0 YCbCr	2:1 horizontal, 2:1 vertical	Y sampled on every pixel, Cb and Cr sampled once on each 2x2 pixel block. 8 bits per component = 48 bits for four pixels.

4.4.2 Data Structures and Enumerators

Two enumerated types are defined to support the integrated color space conversion/resize/rotation function set.

4.4.2.1 Interpolation Schemes

`OMXIPInterpolation`, a data type that enumerates image interpolation schemes, is defined as follows:

```
typedef enum {
   OMX_IP_NEAREST = 0,
   OMX_IP_BILINEAR = 1
} OMXIPInterpolation;
```

4.4.2.2 Color Spaces

`OMXIPColorSpace`, a data type that enumerates color spaces, is defined as follows:

```
typedef enum {
```

```
    OMX_IP_RGB565 = 0,
    OMX_IP_RGB555 = 1,
    OMX_IP_RGB444 = 2,
    OMX_IP_RGB888 = 3,
    OMX_IP_YCBCR422 = 4,
    OMX_IP_YCBCR420 = 5,
} OMXIPColorSpace;
```

4.4.3 Functions

4.4.3.1 YCbCr to RGB

4.4.3.1.1 YCbCr444ToRGB888_U8_C3R

4.4.3.1.2 YCbCr444ToRGB565_U8_U16_C3R

Prototype

```
OMXResult omxIPCS_YCbCr444ToRGB888_U8_C3R(const OMX_U8 *pSrc, OMX_INT
   srcStep, OMX_U8 *pDst, OMX_INT dstStep, OMXSize roiSize);
OMXResult omxIPCS_YCbCr444ToRGB565_U8_U16_C3R(const OMX_U8 *pSrc, OMX_INT
   srcStep, OMX_U16 *pDst, OMX_INT dstStep, OMXSize roiSize);
```

Description

Converts a pixel-oriented YCbCr444 image to RGB888 or RGB565 color space. The ROI of the source image is pointed to by `pSrc`, and the result is placed into the ROI of the destination image pointed to by `pDst`. The input and output images are organized, respectively, as specified by the Table 4-1 entries labeled "YCbCr444" and "RGB888"/"RGB565."

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

4.4.3.1.3 YCbCr444ToRGB565_U8_U16_P3C3R

Prototype

```
OMXResult omxIPCS_YCbCr444ToRGB565_U8_U16_P3C3R(const OMX_U8* pSrc[3],
   OMX_INT srcStep, OMX_U16* pDst, OMX_INT dstStep, OMXSize roiSize);
```

Description

Converts a planar YCbCr444 input image to a pixel-oriented RGB565 output image. The ROI of the source image is pointed to by `pSrc`, and the result is placed into the ROI of the destination image pointed to by `pDst`. The YCbCr444 input image is organized in memory as specified in Table 4-2. The pixel-oriented RGB565 output image is organized in memory as specified in Table 4-1.

Input Arguments

- `pSrc` – vector containing pointers to Y, Cb, and Cr planes
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels

Output Arguments

- `pDst` – pointer to the destination buffer

Return Values

- `OMX_StsNoErr` –No error. Any other value indicates an error or a warning
- `OMX_StsNullPtrErr` – indicates an error condition if `pSrc` or `pDst` pointer is NULL
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

4.4.3.1.4 YCbCr420ToRGB565_U8_U16_P3C3R

Prototype

```
OMXResult omxIPCS_YCbCr420ToRGB565_U8_U16_P3C3R(const OMX_U8* pSrc[3],
   OMX_INT srcStep[3], OMX_U16* pDst, OMX_INT dstStep, OMXSize roiSize);
```

Description

This function converts a planar YCbCr420 input image to a pixel-oriented RGB565 output image. The memory organization for a planar YCbCr420 image is specified in Table 4-2. The memory organization for a pixel-oriented RGB565 image is specified in Table 4-1.

Input Arguments

- `pSrc` – an array of pointers to the YCbCr420 source planes
- `srcStep` – an array of three step values; which represent for each image plane, respectively, the distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels

Output Arguments

- `pDst` – pointer to the destination buffer; RGB565 is represented using 16-bit words that are orgranized as specified in Table 4-1.

Return Values

- `OMX_StsNoErr` – No error. Any other value indicates an error or a warning
- `OMX_StsNullPtrErr` – indicates an error condition if `pSrc` or `pDst` pointer is NULL
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

4.4.3.1.5 YCbCr422ToRGB888_U8_C2C3R

4.4.3.1.6 YCbCr422ToRGB565_U8_U16_C2C3R

Prototype

```
OMXResult omxIPCS_YCbCr422ToRGB888_U8_C2C3R (const OMX_U8 *pSrc, OMX_INT
    srcStep, OMX_U8 *pDst, OMX_INT dstStep, OMXSize roiSize);
OMXResult omxIPCS_YCbCr422ToRGB565_U8_U16_C2C3R (const OMX_U8 *pSrc, OMX_INT
    srcStep, OMX_U16 *pDst, OMX_INT dstStep, OMXSize roiSize);
```

Description

Convert a pixel-oriented YCbCr422 input image to a pixel-oriented RGB888 or RGB565 output image. The ROI of the source image is pointed to by `pSrc`, and the result is placed into the ROI of the destination image referenced by `pDst`. Memory organization for pixel-oriented YCbCr422, RGB888, and RGB565 images is specified in Table 4-1.

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

4.4.3.2 Color Twist

4.4.3.2.1 ColorTwistQ14_U8_C3R

Prototype

```
OMXResult omxIPCS_ColorTwistQ14_U8_C3R(const OMX_U8* pSrc, OMX_INT srcStep,
    OMX_U8* pDst, OMX_INT dstStep, OMXSize roiSize, const OMX_S32
    twistQ14[3][4]);
```

Description

Applies a Q17.14 color twist matrix to the ROI of the source image pointed to by `pSrc`. Places the results into the ROI of the destination image pointed to by `pDst`. The Q14 modifier with a parameter of type OMX_S32 is used to indicate the fact that the matrix entries are obtained by multiplying the entries of the equivalent floating-point color twist matrix with (1<<14).

Input Arguments

- `pSrc` – pointer to the source ROI
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image
- `roiSize` – size of the source and destination ROI in pixels
- `twistQ14` – twist matrix

Output Arguments

- `pDst` – pointer to the destination ROI

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsNullPtrErr` – NULL pointer error
- `OMX_StsStepErr` – step value is less or equal zero
- `OMX_StsSizeErr` – indicates an error condition if `roiSize` has a field with zero or negative value

4.4.3.3 Integrated CSC/Rotate/Integer Resize

4.4.3.3.1 YCbCr422RszCscRotRGB_U8_U16_C2R

Prototype

```
OMXResult omxIPCS_YCbCr422RszCscRotRGB_U8_U16_C2R(const OMX_U8 *pSrc,
    OMX_INT srcStep, OMX_U16 *pDst, OMX_INT dstStep, OMXSize roiSize,
    OMX_INT scaleFactor, OMXIPInterpolation interpolation, OMXIPColorSpace
    RGBSpec, OMXIPRotation rotation);
```

Description

This function synthesizes a low-resolution preview image from the input image. In particular, the following sequence of operations is applied to the input image:

1. Scale reduction by an integer scalefactor. First, the input image scale is reduced by an integer scalefactor of either 2, 4, or 8 on both axes using the interpolation methodology specified by the control parameter interpolation. The following interpolation schemes are supported: nearest neighbor, bilinear.
2. Color space conversion. Following scale reduction, color space conversion is applied from the YCbCr422 input color space to a particular RGB target space determined by the control parameter `RGBSpec`.
3. Rotation. After color space conversion, the preview output image is rotated according to the control parameter rotation.

Input Arguments

- `pSrc` – pointer to the start of the buffer containing the pixel-oriented input image
- `srcStep` – distance, in bytes, between the start of lines in the source image
- `dstStep` – distance, in bytes, between the start of lines in the destination image
- `roiSize` – dimensions, in pixels, of the source region of interest
- `scaleFactor` – reduction scalefactor; values other than 2, 4, or 8 are invalid
- `interpolation` – interpolation methodology control parameter; must take one of the following values: `OMX_IP_NEAREST` or `OMX_IP_BILINEAR` for nearest neighbor or bilinear interpolation, respectively
- `RGBSpec` – color conversion control parameter; must be set to one of the following pre-defined values: `OMX_IP_RGB565` or `OMX_IP_RGB555`
- `rotation` – rotation control parameter; must be set to one of the following pre-defined values: `OMX_IP_DISABLE`, `OMX_IP_ROTATE90L`, `OMX_IP_ROTATE90R`, or `OMX_IP_ROTATE180`

Output Arguments

- `pDst` – pointer to the start of the buffer containing the resized, color-converted, and rotated output image

Returns

If the function runs without error, it returns `OMX_StsNoErr`

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pSrc` or `pDst` is NULL
- `pSrc` or `pDst` is not aligned at 8 bytes boundary
- `srcStep` or `dstStep` is less than 1, or `srcStep`, or `dstStep` is not multiple of 8
- `roiSize.width` is larger than half of `srcStep`
- `roiSize.width` or `roiSize.height` is less than `scaleFactor`
- Invalid values of one or more of the following control parameters:
  — `scaleFactor`, `interpolation`, `colorConversion` or `rotation`
  — Half of the `dstStep` is less than width of downscaled, color-converted and rotated image

Alignment Requirement

The start address of `pSrc`, `pDst`, must be aligned at 8-byte boundary; `srcStep` and `dstStep` must be multiple of 8.

Size of Output Image

If `roiSize.width` or `roiSize.height` cannot be divided by `scaleFactor` exactly, it will be cut to be the multiple of scaleFactor. For example, if the rotation control parameter is `OMX_IP_DISABLE` or `OMX_IP_ROTATE180`, the output image's `roiSize.width` is equal to round ((input image's roiSize.width)/scaleFactor).

Example 4-3: Integrated Scaling, Color Space Conversion, and Rotation

```
#include <stdlib.h>
#include "OMXIP.h"
#define _ALIGN8(adr) ((((OMX_U32)(adr))+7)&(~7))
OMX_INT main()
{
 OMX_INT W, H, BufSize;
 /* Variables in preview function */
 OMX_U8 *pSrc, *pSrcAlign;
 OMX_U16 *pDst, *pDstAlign;
 OMX_INT scaleFactor;
 OMX_INT srcStep, dstStep;
 OMXCameraInterpolation interpolation;
 OMXCameraRotation rotation;
 OMXCameraCsc colorConversion;
 OMXSize roiSize;

 /* Initialize Width and Height of the input image */
 W=320;
 H=240;

 /* Initialize control parameters */
 interpolation = omxInterpBilinear;
 rotation = omxRotate90R;
 colorConversion = omxCscYCbCr422ToRGB565 ;

 /* Initialize source parameter */
 roiSize.width = W;
 roiSize.height = H;
 srcStep = 640;
 dstStep = 240;
 scaleFactor = 2;
```

```
 /* Allocate buffer */
 pSrc = (OMX_U8*) malloc(srcStep*H+7);
 pDst = (OMX_U16*) malloc(dstStep*W+7);

 /* Align Source and Output Buffer at 8-byte */
 pDstAlign = (OMX_U16*)_ALIGN8(pDst);
 pSrcAlign = (OMX_U8*)_ALIGN8(pSrc);

 /* Rsz/Csc/Rot */
 omxIPCS_YCbCr422RszCscRotRGB_U8_C2R(pSrcAlign, srcStep, pDstAlign,
dstStep, roiSize,
             scaleFactor,interpolation, colorConversion, rotation);

 /* Free buffer */
 free(pSrc);
 free(pDst);
 return (0);
}
```

4.4.3.4 Integrated Rotate/Fractional Resize

4.4.3.4.1 YCbCr420RszRot_U8_P3R

4.4.3.4.2 YCbCr422RszRot_U8_P3R

Prototype

```
OMXResult omxIPCS_YCbCr420RszRot_U8_P3R(const OMX_U8 *pSrc[3], OMX_INT
    srcStep[3], OMXSize srcSize, OMX_U8 *pDst[3], OMX_INT dstStep[3],
    OMXSize dstSize, OMXIPInterpolation interpolation, OMXIPRotation
    rotation, OMX_INT rcpRatiox, OMX_INT rcpRatioy);
OMXResult omxIPCS_YCbCr422RszRot_U8_P3R(const OMX_U8 *pSrc[3], OMX_INT
    srcStep[3], OMXSize srcSize, OMX_U8 *pDst[3], OMX_INT dstStep[3],
    OMXSize dstSize, OMXIPInterpolation interpolation, OMXIPRotation
    rotation, OMX_INT rcpRatiox, OMX_INT rcpRatioy);
```

Description

This function combines two atomic image processing kernels into a single function. The following sequence of operations is applied:

1. Resize. The input image of dimension `srcSize` is rescaled according to the Q16 reciprocal ratio

scaling control parameters rcpRatiox and rcpRatioy using the interpolation methodology specified by the control parameter interpolation. Nearest neighbor and bilinear interpolation schemes are supported. The rescaled image is clipped to dstSize using a clipping rectangle, the origin of which coincides with the top, left corner of the input image, i.e., pixels are discarded along the right and bottom edges.

2. Rotation. The output image is rotated with respect to the input image according to the control parameter rotation.

The input data should be YCbCr420 planar format for <omxIPCS_YCbCr420RszRot_U8_P3R> and YCbCr422 planar format for <omxIPCS_YCbCr422RszRot_U8_P3R>.

Input Arguments

- pSrc – a 3-element vector containing pointers to the start of each of the YCbCr420 input planes for <omxIPCS_YCbCr420RszRot_U8_P3R> and YCbCr422 input planes for <omxIPCS_YCbCr422RszRot_U8_P3R>
- srcStep – a 3-element vector containing the distance, in bytes, between the start of lines in each of the input image planes
- dstStep – a 3-element vector containing the distance, in bytes, between the start of lines in each of the output image planes
- srcSize – dimensions, in pixels, of the source image
- dstSize – dimensions, in pixels, of the destination image (before applying rotation to the resized image)
- interpolation – interpolation methodology control parameter; must take one of the following values: OMX_IP_NEAREST, or OMX_IP_BILINEAR for nearest neighbor or bilinear interpolation, respectively
- rotation – rotation control parameter; must be set to one of the following pre-defined values: . OMX_IP_DISABLE, OMX_IP_ROTATE90L, OMX_IP_ROTATE90R, OMX_IP_ROTATE180, OMX_IP_FLIP_HORIZONTAL, or OMX_IP_FLIP_VERTICAL
- rcpRatiox – x direction scaling control parameter, specified in terms of a reciprocal resize ratio using a Q16 representation. Valid in the range [1, xrr_max], where xrr_max = ((((srcSize.width&~1)-1) / ((dstSize.width&~1)-1)) << 16). Setting rcpRatiox = xrr_max guarantees that the output image size will be exactly dstSize; otherwise for values less than xrr_max the right hand side of the image will be clipped since the output image will extend beyond dstSize. Expansion in the x direction occurs for values of rcpRatiox > 65536; contraction in the x direction occurs for values < 65536. To avoid clipping, use the value xrr_max. Values larger than xrr_max are invalid, i.e., output images smaller than dstSize are not allowed.
- rcpRatioy – y direction scaling control parameter, specified in terms of a reciprocal resize ratio using a Q16 representation. Valid in the range [1, yrr_max], where yrr_max = ((((srcSize.height&~1)-1) / ((dstSize.height&~1)-1)) << 16). Setting rcpRatioy = yrr_max guarantees that the output image size will be exactly dstSize; otherwise for values less than yrr_max the bottom of the output image will be clipped since the output image will be larger than dstSize. Expansion in the y direction occurs for values of rcpRatioy > 65536; contraction in the y direction occurs for values < 65536. To avoid clipping, use the value yrr_max. Values larger than yrr_max are invalid, i.e., output images smaller than dstSize are not allowed.

Output Arguments

- `pDst` – a 3-element vector containing pointers to the start of each of the YCbCr420 output planes for <`omxIPCS_YCbCr420RszRot_U8_P3R`> or `YCbCr422` output planes for <`omxIPCS_YCbCr422RszRot_U8_P3R`>

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- Any pointer is NULL
- Each of `srcSize.width`, `srcSize.height`, `dstSize.width` and `dstSize.height` <= 0.
- `rcpRatiox` <= 0 or `rcpRatioy` <= 0
- `pDst[0]` is not aligned at 4 bytes boundary, `pSrc[0]` is not aligned at 4 bytes boundary
- `dstSize.width` or `dstSize.height` is not even
- `srcStep[0], srcStep[1]`, `srcStep[2]`or `dstStep[0]`, `dstStep[1]`, `dstStep[2]`is less than 1
- `srcStep[0]` is not multiple of 4, `dstStep[0]` is not multiple of 4
- `roiSize.width` is larger than `srcStep[0]; srcSize.width`>>1 is larger than `srcStep[1]` or `srcStep[2]`.
- Invalid values of one or both of the following control parameters: interpolation or rotation
- `dstSize.height` of output image is larger than `dstStep[0]`, `dstSize.height`>>1 of output image is larger than `dstStep[1]` or `dstStep[2]` when rotation is OMX_IP_ROTATE90L or `OMX_IP_ROTATE90R`; `dstSize.width` of output image is larger than `dstStep[0]`, `dstSize.width`>>1 of output image is larger than `dstStep[1]` or `dstStep[2]` when other valid rotation options.

---

2 ***Note:** The dstSize.width and dstSize.height must be even.*

Alignment Requirement

- The start address of `pDst[0]` must be aligned at 4-byte boundary and `dstStep[0]` must be multiple of 4.
- The start address of `pSrc[0]` must be aligned at 4-byte boundary, and `srcStep[0]` must be multiple of 4.

Size of Output Image

`dstSize` is the size of destination image before rotation, i.e., `dstSize` is the size of the image after resizing.

Below is an example that demonstrates how to call the `omxIPCS_YCbCr420RszRot_U8_P3R` function.

Example 4-4:

```
#include <stdio.h>
#include <stdlib.h>
#include "OMXIP.h"
#define  _ALIGN4(adr)   ((((OMX_U32)(adr))+3)&(~3))

OMXResult resize_rotate();

OMX_INT main()
{
    resize_rotate();
}
OMXResult resize_rotate()
{
OMX_U8    *pSrc1, *pSrc2, *pSrc3, *pDst1, *pDst2, *pDst3;
OMX_U8    *pSrc[3], *pDst[3];
OMXSize srcSize, dstSize;
OMXIPRotation rotation;
OMXIPInterpolationinterpolation;
OMX_INT      rcpRatioX, rcpRatioY;
OMX_INT      srcStep[3], dstStep, size;
OMX_INT      status;

    /* Initialize parameters */
    srcSize.width  = 176;
    srcSize.height = 144;
    dstSize.width  = 320;
    dstSize.height = 240;
    srcStep[0]     = 176;
    srcStep[1]     = 88;
    srcStep[2]     = 88;
dstStep[0]     = 320;
dstStep[1]     = 160;
dstStep[2]     = 160;
interpolation  = OMX_IP_BILINEAR;
rotation    = OMX_IP_DISABLE;
    rcpRatioX = (OMX_INT)(((double)((srcSize.width-
1)<<16))/(dstSize.width-1));
rcpRatioY = (OMX_INT)(((double)((srcSize.height-
1)<<16))/(dstSize.height-1));
```

```
    pSrc1 = (OMX_U8*)malloc(srcSize.height*srcStep[0] + 8);
    pSrc2 = (OMX_U8*)malloc((srcSize.height>>1)*srcStep[1] + 8);
pSrc3 = (OMX_U8*)malloc((srcSize.height>>1)*srcStep[2] + 8);
pSrc[0] = (OMX_U8*)_ALIGN4(pSrc1);
pSrc[1] = pSrc2;
pSrc[2] = pSrc3;

size  = dstSize.height * dstStep[0] + 8;
pDst = (OMX_U8*)malloc(size);
size  = (dstSize.height>>1) * dstStep[1] + 8;
pDst2 = (OMX_U8*)malloc(size);
pDst3 = (OMX_U8*)malloc(size);

pDst[0] = (OMX_U8*)_ ALIGN4(pDst1);
pDst[1] = pDst2;
pDst[2] = pDst3;

/* here to initialize the content of pSrc[0], pSrc[1] and pSrc[2] */
pSrc[0] = ...;
pSrc[1] = ...;
pSrc[2] = ...;
/* here to initialize the content of pSrc[0], pSrc[1] and pSrc[2] */
        /* resize & rotate */
status = omxIPCS_YCbCr420RszRot_U8_P3R(pSrc, srcStep, srcSize, pDst,
 dstStep, dstSize, interpolate, rotation,
 rcpRatioX, rcpRatioY);
    return status;
}
```

4.4.3.5 Integrated CSC/Rotate/Fractional Resize

4.4.3.5.1 YCbCr420RszCscRotRGB_U8_P3C3R

4.4.3.5.2 YCbCr422RszCscRotRGB_U8_P3C3R

Prototype

```
OMXResult omxIPCS_YCbCr420RszCscRotRGB_U8_P3C3R(const OMX_U8 *pSrc[3],
    OMX_INT srcStep[3], OMXSize srcSize, void *pDst, OMX_INT dstStep,
    OMXSize dstSize, OMXIPColorSpace colorConversion, OMXIPInterpolation
    interpolation, OMXIPRotation rotation, OMX_INT rcpRatiox, OMX_INT
    rcpRatioy);
OMXResult omxIPCS_YCbCr422RszCscRotRGB_U8_P3C3R(const OMX_U8 *pSrc[3],
    OMX_INT srcStep[3], OMXSize srcSize, void *pDst, OMX_INT dstStep,
    OMXSize dstSize, OMXIPColorSpace colorConversion, OMXIPInterpolation
    interpolation, OMXIPRotation rotation, OMX_INT rcpRatiox, OMX_INT
    rcpRatioy);
```

Description

This function combines several atomic image processing kernels into a single function. In particular, the following sequence of operations is applied to the input YCbCr image:

1. Resize. The input image of dimension `srcSize` is rescaled according to the Q16 reciprocal ratio scaling control parameters `rcpRatiox` and `rcpRatioy` using the interpolation methodology specified by the control parameter `interpolation`. Nearest neighbor and bilinear interpolation schemes are supported. The rescaled image is clipped to `dstSize` using a clipping rectangle, the origin of which coincides with the top, left corner of the input image, i.e., pixels are discarded along the right and bottom edges.
2. Color space conversion. Following scaling, color space conversion from either YCbCr420 or YCbCr422 to RGB is applied according to the control parameter `colorConversion`. The following target RGB color spaces are supported: RGB565, RGB888, RGB555, or RGB444.
3. Rotation. After color space conversion, the output image is rotated with respect to the input image according to the control parameter `rotation`.

The input data should be in YCbCr420 planar format for the function <`omxIPCS_YCbCr420RszCscRotRGB_U8_P3CR`> or YCbCr422 planar format for the function <`omxIPCS_YCbCr422RszCscRotRGB_U8_P3CR`>.

Input Arguments

- `pSrc` – a 3-element vector containing pointers to the start of each of the YCbCr420 or YCbCr422 input planes
- `srcStep` – a 3-element vector containing the distance, in bytes, between the start of lines in each of the input image planes
- `dstStep` – distance, in bytes, between the start of lines in the destination image
- `srcSize` – dimensions, in pixels, of the source image
- `dstSize` – dimensions, in pixels, of the destination image (before applying rotation to the resizing image)

- `interpolation` – interpolation methodology control parameter; must take one of the following values: OMX_IP_NEAREST, or OMX_IP_BILINEAR for nearest neighbor or bilinear interpolation, respectively.
- `colorConversion` – color conversion control parameter; must be set to one of the following pre-defined values: `OMX_IP_RGB565`, `OMX_IP_RGB555`, `OMX_IP_RGB444`, or `OMX_IP_RGB888`.
- `rotation` – rotation control parameter; must be set to one of the following pre-defined values:
  - `OMX_IP_DISABLE`
  - `OMX_IP_ROTATE90L`
  - `OMX_IP_ROTATE90R`
  - `OMX_IP_ROTATE180`
  - `OMX_IP_FLIP_HORIZONTAL`
  - `OMX_IP_FLIP_VERTICAL`

  Counter-clockwise rotation is denoted by the "L" postfix, and clockwise rotation is denoted by the "R" postfix. A horizontal flip creates a "mirror" image with respect to the vertical image axis, i.e.,

  horizontal flip
  
  ,

  and a vertical flip creates a "mirror" image with respect to the horizontal image axis, i.e.,

  vertical flip
  

- `rcpRatiox` – x direction scaling control parameter, specified in terms of a reciprocal resize ratio using a Q16 representation. Valid in the range [1, xrr_max], where xrr_max = ((((srcSize.width&~1)-1) / ((dstSize.width&~1)-1)) << 16). Setting `rcpRatiox` = xrr_max guarantees that the output image size will be exactly dstSize; otherwise for values less than xrr_max the right hand side of the image will be clipped since the output image will extend beyond dstSize. Expansion in the x direction occurs for values of `rcpRatiox` > 65536; contraction in the x direction occurs for values < 65536. To avoid clipping, use the value xrr_max. Values larger than xrr_max are invalid, i.e., output images smaller than dstSize are not allowed.
- `rcpRatioy` – y direction scaling control parameter, specified in terms of a reciprocal resize ratio using a Q16 representation. Valid in the range [1, yrr_max], where yrr_max = ((((srcSize.height&~1)-1) / ((dstSize.height&~1)-1)) << 16). Setting `rcpRatioy` = yrr_max guarantees that the output image size will be exactly dstSize; otherwise for values less than yrr_max the bottom of the output image will be clipped since the output image will be larger than dstSize. Expansion in the y direction occurs for values of `rcpRatioy` > 65536; contraction in the y direction occurs for values < 65536. To avoid clipping, use the value yrr_max. Values larger than yrr_max are invalid, i.e., output images smaller than dstSize are not allowed.

Output Arguments

- `pDst` – pointer to the start of the buffer containing the resized, color-converted, and rotated output image.

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- Any pointer is NULL
- `rcpRatiox` <= 0 or `rcpRatioy` <= 0
- Each of `srcSize.width`, `srcSize.height`, `dstSize.width` and `dstSize.height` <= 0
- `pDst` is not aligned at 8 bytes boundary, `pSrc[0]` is not aligned at 4 bytes boundary, `pSrc[1]` is not aligned at 2 bytes boundary, or `pSrc[2]` is not aligned at 2 bytes boundary.
- `srcStep[0]`, `srcStep[1]`, `srcStep[2]` or `dstStep` is less than 1.
- `srcStep[0]` is not a multiple of 4, `srcStep[1]` is not multiple of 2, or `srcStep[2]` is not multiple of 2.
- `dstStep` is not a multiple of 8 when `colorConversion` is `OMX_IP_RGB565`, `OMX_IP_RGB555`, or `OMX_IP_RGB444`; `dstStep` is not multiple of 2 when `colorConversion` is `OMX_IP_RGB888`.
- `roiSize.width` is larger than `srcStep`[0]; `roiSize.width`>>1 is larger than half of `srcStep`[1] or `srcStep[2]`.
- Invalid values of one or more of the following control parameters:
  - `interpolation`
  - `colorConversion`
  - `rotation`
- `dstSize.height` * bytes/pixel of output image is larger than `dstStep` when rotation is `OMX_IP_ROTATE90L` or `OMX_IP_ROTATE90R`; `dstSize.width` * bytes/pixel of output image is larger than `dstStep` when other valid rotation options.

Alignment Requirements

The starting address of `pDst` must be aligned at 8-byte boundary; and `dstStep` must be multiple of 8 when output format is RGB444/555/565, and `dstStep` must be multiple of 2 when output format is RGB888.

The starting address of `pSrc`[0] must be aligned at 4-byte boundary, `pSrc`[1] must be aligned at 2-boundary and `pSrc[2]` must be aligned at 2-boundary; and `srcStep`[0] must be multiple of 4, `srcStep`[1] must be multiple of 2 and `srcStep[2]` must be multiple of 2.

Size of Output Image

The parameter `dstSize` specifies the size of the image after the resizing operation, but prior to the rotation operation.

An example is givcen below that demonstrates how to call the function `omxIPCS_YCbCr422RszCscRotRGB_U8_P3C3R`.

Example 4-5:

```
#include <stdio.h>
#include <stdlib.h>
#include "OMXIP.h"
#define  _ALIGN8(adr)   ((((OMX_U32)(adr))+7)&(~7))
#define  _ALIGN4(adr)   ((((OMX_U32)(adr))+3)&(~3))

OMXResult resize_rotate();

OMX_INT main()
{
 resize_rotate();
}
OMXResult resize_rotate()
{
 OMX_U8    *pSrc1, *pSrc2, *pSrc3;
 OMX_U8    *pSrc[3];
 void      *pDst;
 OMXSize   srcSize, dstSize;
 OMXIPInterpolation        .........................................
 OMX_INT   rcpRatioX, rcpRatioY;
 OMX_INT   srcStep[3], dstStep, size;
 OMX_INT   status;

 /* Initialize parameters */
 srcSize.width = 176;
 srcSize.height = 144;
 dstSize.width = 320;
 dstSize.height = 240;
 srcStep[0] = 176;
 srcStep[1] = 88;
 srcStep[2] = 88;
 dstStep = 640;
 colorConversion= OMX_IP_RGB565 ;
 interpolation  = OMX_IP_BILINEAR;
 rotation    = OMX_IP_DISABLE;
 rcpRatioX = (OMX_INT)(((double)((srcSize.width-
1)<<16))/(dstSize.width-1));
 rcpRatioY = (OMX_INT)(((double)((srcSize.height-
1)<<16))/(dstSize.height-1));
```

```
    size = dstSize.height * dstStep + 8;
    pSrc1 = (OMX_U8*)malloc(srcSize.height*srcStep[0] + 8);
    pSrc2 = (OMX_U8*)malloc((srcSize.height>>1)*srcStep[1] + 8);
    pSrc3 = (OMX_U8*)malloc((srcSize.height>>1)*srcStep[2] + 8);
    pSrc[0] = (OMX_U8*)_ALIGN4(pSrc1);
    pSrc[1] = (OMX_U8*)_ALIGN4(pSrc2);
    pSrc[2] = (OMX_U8*)_ALIGN4(pSrc3);
    pDst = malloc(size);
    pDst = (void*)_ALIGN8(pDst);

    /* initialize the content of pSrc[0], pSrc[1] and pSrc[2] */
    pSrc[0] = ...;
    pSrc[1] = ...;
    pSrc[2] = ...;

    /* here to initialize the content of pSrc[0], pSrc[1] and pSrc[2]
   */
    /* resize & rotate */
    status = omxIPCS_YCbCr422RszCscRotRGB_U8_P3C3R(pSrc, srcStep,
   srcSize, pDst,
                 dstStep, dstSize, colorConversion, interpolate,
   rotation,
              rcpRatioX, rcpRatioY);
    return status;
   }
```

4.4.3.6 Integrated CSC/Rotate

4.4.3.6.1 YCbCr422ToYCbCr420Rotate_U8_C2P3R

Prototype

```
OMXResult omxIPCS_YCbCr422ToYCbCr420Rotate_U8_C2P3R(const OMX_U8 *pSrc,
   OMX_INT srcStep, OMX_U8 *pDst[3], OMX_INT dstStep[3], OMXSize roiSize,
   OMXIPRotation rotation);
```

Description

YCbCr422 to YCbCr420 planar format conversion with rotation function. This function decimates and interpolates the color space of the input image from YCbCr 422 to YCbCr 420, applies an optional rotation of -90, +90, or 180 degrees, and then rearranges the data from the pixel-oriented input format to a planar output format.

Input Arguments

- `pSrc` – pointer to the start of the buffer containing the pixel-oriented YCbCr422 input
- `srcStep` – distance, in bytes, between the start of lines in the source image
- `dstStep` – a 3-element vector containing the distance, in bytes, between the start of lines in each of the output image planes
- `roiSize` – dimensions, in pixels, of the source and destination regions of interest
- `rotation` – rotation control parameter; must be set to one of the following pre-defined values: `OMX_IP_DISABLE`, `OMX_IP_ROTATE90L`, `OMX_IP_ROTATE90R`, or `OMX_IP_ROTATE180`.

OutputArguments

- `pDst` – a 3-element vector containing pointers to the start of each of the YCbCr420 output planes

Returns

If the function runs without error, it returns OMX_StsNoErr

If any of the following cases occurs, the function returns OMX_StsBadArgErr:

- pSrc, pDst[0], pDst[1], pDst[2] is NULL
- `pSrc` or `pDst[0]` is not aligned at 8 bytes boundary
- `pDst[1]` or `pDst[2]` is not aligned at 4-byte boundary
- srcStep, dstStep[1], dstStep[2], or dstStep[3] is less than 1
- `srcStep` or `dstStep[0]` is not multiple of 8
- `dstStep[1]` or `dstStep[2]` is not multiple of 4
- `roiSize.width` is larger than half of `srcStep`
- Invalid values of rotation control parameters
- `dstStep[0]` is less than `roiSize.width` of downscaled, color-converted and rotated image; Half of the `dstStep[1]` or half of the `dstStep[2]` is less than `roiSize.width` of downscaled, color-converted and rotated image
- `RoiSize.width` or `roiSize.height` is less than 8

Alignment Requirement

`pSrc`, `pDst`[0], `srcStep` and `dstStep`[0] should be 8 bytes aligned; `pDst`[1], `pDst`[2], `dstStep`[1], and `dstStep`[2] should be 4 bytes aligned.

Size of Output Image

If `roiSize.width` or `roiSize.height` cannot be divided by 8 exactly, it will be cut to be a multiple of 8.

4.4.3.6.2 YCbCr422ToYCbCr420Rotate_U8_P3R

Prototype

```
OMXResult omxIPCS_YCbCr422ToYCbCr420Rotate_U8_P3R(const OMX_U8 *pSrc[3],
    OMX_INT srcStep[3], OMX_U8 *pDst[3], OMX_INT dstStep[3], OMXSize
    roiSize, OMXIPRotation rotation);
```

Description

This function decimates and interpolates the color space of the input image from YCbCr 422 planar data to YCbCr 420 planar data, and then applies an optional rotation of -90, +90, or 180 degrees. Difference between this function and `omxIPCS_YCbCr422ToYCbCr420Rotate_U8_C2P3R` is that this function supports the input YCbCr422 format in planar order.

Input Arguments

- `pSrc` – a 3-element vector containing pointers to the start of each of the YCbCr420 input planes
- `srcStep` – a 3-element vector containing the distance, in bytes, between the start of lines in each of the input image planes
- `dstStep` – a 3-element vector containing the distance, in bytes, between the start of lines in each of the output image planes
- `roiSize` – dimensions, in pixels, of the source and destination regions of interest
- `rotation` – rotation control parameter; must be set to one of the following pre-defined values: OMX_IP_DISABLE, OMX_IP_ROTATE90L, OMX_IP_ROTATE90R, or OMX_IP_ROTATE180

Output Arguments

- `pDst` – a 3-element vector containing pointers to the start of each of the YCbCr420 output planes

Returns

If the function runs without error, it returns `OMX_StsNoErr`

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- Any pointer is NULL
- `pSrc[0]` or `pDst[0]` is not aligned at 8 bytes boundary; `pSrc[1]`, `pSrc[2]`, `pDst[1]` or `pDst[2]` is not aligned at 4-byte boundary
- Any of the steps is less than 1; `srcStep[0]` or `dstStep[0]` is not multiple of 8; `srcStep[1]`, `srcStep[2]`, `dstStep[1]` or `dstStep[2]` is not multiple of 4.
- `roiSize.width` is larger than `srcStep[0]`; `roiSize.width` is larger than twice `srcStep[1]` or twice `srcStep`[2].
- Invalid values of rotation control parameters.
- `dstStep[0]` is less than `roiSize.width` of downscaled, color-converted and rotated image; Half of the `dstStep[1]` or half of the `dstStep[2]` is less than `roiSize.width` of downscaled, color-converted and rotated image.
- `roiSize.width` or `roiSize.height` is less than 8.

Alignment Requirement

`pSrc[0]`, `pDst[0]`, `srcStep[0]` and `dstStep[0]` should be 8 bytes aligned; `pSrc[1]`, `pSrc[2]`, `pDst[1]`, `pDst[2]`, `srcStep[1]`, `srcStep[2]`, `dstStep[1]` and `dstStep[2]` should be 4 bytes aligned.

Size of Output Image

If the `roiSize.width` or `roiSize.height` cannot be divided by 8 exactly, it will be cut to be multiple of 8.

The following is an example of how to call the function `omxIPCS_YCbCr422ToYCbCr420Rotate_U8_C2P3R`

Example 4-6:

```
#include <stdlib.h>
#include "OMXIP.h"

#define  _ALIGN8(adr)   ((((OMX_U32)(adr))+7)&(~7))
#define  _ALIGN4(adr)   ((((OMX_U32)(adr))+3)&(~3))

OMX_INT main()
{
OMX_INT W, H;
OMX_U8  *pDst1, *pDst2, *pDst3;

/*   Variables used in Function   */
OMX_U8       *pSrc, *pDst[3];
OMX_INT       .........................................srcStep, dstS
OMXSize      roiSize;
OMXIPInterpolation rotation;

       /*   Initialize the width and height of input image   */
W = 320;
H = 240;

/*   Initialize source parameters   */
rotation  = OMX_IP_ROTATE180;
roiSize.width = W;
roiSize.height = H;
srcStep = 640;
dstStep[0] = 320;
dstStep[1] = 160;
dstStep[2] = 160;

/*   Allocate Buffer   */
```

```
pSrc = (OMX_U8*) malloc(W*H*2+7);
pDst1 = (OMX_U8*) malloc(W*H+7);
pDst2 = (OMX_U8*) malloc(W*H/2+7);
pDst3 = (OMX_U8*) malloc(W*H/2+7);

/*   Align buffer*/
pSrc = (OMX_U8*)_ALIGN8(pSrc);
pDst1 = (OMX_U8*)_ALIGN8(pDst1);
pDst2 = (OMX_U8*)_ALIGN4(pDst2);
pDst3 = (OMX_U8*)_ALIGN4(pDst3);

pDst[0] = pDst1;
pDst[1] = pDst2;
pDst[2] = pDst3;

/*   Call omxIPCS_YCbCr422ToYCbCr420Rotate_U8_C2P3R   */
omxIPCS_YCbCr422ToYCbCr420Rotate_U8_C2P3R(pSrc, srcStep, pDst,
dstStep, roiSize, rotation);

/*   Free Buffer*/
free(pSrc);
free(pDst1);
free(pDst2);
free(pDst3);

return (0);
}
```

4.4.3.7 JPEG-Specific RGB to YCbCr with Integrated Level Shift

The color conversion function defined below are provided to support MCU-based JPEG codec construction. These functions convert RGB to YCbCr in the CCIR 601 color space. The following equations specify the mathematical definition of forward and inverse. Reference: *JPEG File Interchange Format*, Version 1.02, September 1992.

(Y, Cb and Cr belong to [0, 1].)

- RGB -> YCbCr:
  - Y = 0.29900 * R + 0.58700 * G + 0.11400 * B
  - Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B + 0.5

— Cr = 0.50000 * R - 0.41869 * G - 0.08131 * B + 0.5

- YCbCr -> RGB:
  - — R = Y + 1.402(Cr-0.5)
  - — G = Y – 0.34414(Cb-0.5) –0.71414(Cr-0.5)
  - — B = Y + 1.772(Cb-0.5)

Level shift is integrated into the JPEG color conversion functions. So, the equations were modified as follows:

(Y, Cb and Cr belong to [-0.5, 0.5].)

- RGB -> YCbCr:
  - — Y = 0.29900 * R + 0.58700 * G + 0.11400 * B – 0.5
  - — Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B
  - — Cr = 0.50000 * R - 0.41869 * G - 0.08131 * B
- YCbCr -> RGB:
  - — R = (Y+0.5) + 1.402*Cr
  - — G = (Y+0.5) – 0.34414*Cb –0.71414*Cr
  - — B = (Y+0.5) + 1.772*Cb

The color conversion equations are implemented using integer data types and therefore the Y, Cb, and Cr channels are represented using Q8 such that each pixel takes a value between -128 and 127, inclusive.

4.4.3.7.1 RGB888ToYCbCr444LS_MCU_U8_S16_C3P3R

4.4.3.7.2 RGB888ToYCbCr422LS_MCU_U8_S16_C3P3R

4.4.3.7.3 RGB888ToYCbCr420LS_MCU_U8_S16_C3P3R

Prototype

```
OMXResult omxIPCS_RGB888ToYCbCr444LS_MCU_U8_S16_C3P3R (const OMX_U8 * pSrc,
   OMX_INT srcStep, OMX_S16 * pDstMCU[3]);
OMXResult omxIPCS_RGB888ToYCbCr422LS_MCU_U8_S16_C3P3R (const OMX_U8 * pSrc,
   OMX_INT srcStep, OMX_S16 * pDstMCU[3]);
OMXResult omxIPCS_RGB888ToYCbCr420LS_MCU_U8_S16_C3P3R (const OMX_U8 * pSrc,
   OMX_INT srcStep, OMX_S16 * pDstMCU[3]);
```

Description

These functions convert an input RGB888 image to one of the following sub-sampled color spaces with level-shift: YCbCr4:4:4, YCbCr4:2:2, and YCbCr4:2:0. Data is processed in MCUs for which the Y blocks have the following dimensions: YCbCr4:4:4: 8x8, YCbCr 4:2:2: 16x8; and YCbCr 4:2:0: 16x16.

Input Arguments

- `pSrc` – pointer to the source image data buffer. The source image data are stored in interleaved order as BGRBGRBGR… The image data buffer `pSrc` can support bottom-up storage formats. For bottom-up images, `srcStep` can be less than 0.

- `srcStep` – distance in bytes between the starts of consecutive lines in the source image; can be less than 0 to support down to top storage format.

Output Arguments

- `pDstMCU[3]` – output MCU pointers; all of them must be 8-byte aligned. The buffers referenced by `pDstMCU[]` support top-down storage format only. The output components are expressed using a Q8 representation and are bounded on the interval [-128, 127]. `pDstMCU[0]` points to the Y block, `pDstMCU[1]` points to the Cb block, and `pDstMCU[2]` points to the Cr block.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – one or more of the following bad argument conditions was detected:
  - — a pointer was NULL
  - — the absolute value of `srcStep` was smaller than 24 for YCbCr 444 or 48 for YCbCr422/ YCbCr 420
  - — the start address of each pointer in `pDstMCU[]` was not 8-byte aligned

4.4.3.7.4 RGB565ToYCbCr444LS_MCU_U16_S16_C3P3R

4.4.3.7.5 RGB565ToYCbCr422LS_MCU_U16_S16_C3P3R

4.4.3.7.6 RGB565ToYCbCr420LS_MCU_U16_S16_C3P3R

Prototype

```
OMXResult omxIPCS_RGB565ToYCbCr444LS_MCU_U16_S16_C3P3R (const OMX_U16 *
    pSrc, OMX_INT srcStep, OMX_S16 * pDstMCU[3]);
OMXResult omxIPCS_RGB565ToYCbCr422LS_MCU_U16_S16_C3P3R (const OMX_U16 *
    pSrc, OMX_INT srcStep, OMX_S16 * pDstMCU[3]);
OMXResult omxIPCS_RGB565ToYCbCr420LS_MCU_U16_S16_C3P3R (const OMX_U16 *
    pSrc, OMX_INT srcStep, OMX_S16 * pDstMCU[3]);
```

Description

This function converts packed RGB565 image data to the following sub-sampled color spaces: YCbCr4:4:4, YCbCr4:2:2, and YCbCr4:2:0. Data is processed in MCUs for which the Y blocks have the following dimensions: YCbCr4:4:4: 8x8, YCbCr 4:2:2: 16x8; and YCbCr 4:2:0: 16x16.

Input Arguments

- `pSrc` – references the source image data buffer. Pixel intensities are interleaved as shown in Table 4-1, and G, B, and R are represented using, respectively, 6, 5, and 5 bits. The image data buffer `pSrc` supports bottom-up storage format, for which `srcStep` can be less than 0.
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image; can be less than 0 to support down to top storage format.

Output Arguments

- `pDstMCU[3]` – output MCU pointers: `pDstMCU[0]` points to the Y block, `pDstMCU[1]` points to Cb the block, and `pDstMCU[2]` points to the Cr block; all three pointers must be aligned on 8-byte boundaries. The buffers referenced by `pDstMCU[]` support only top-down storage format. The output components are expressed using a Q8 representation and are bounded on the interval [-128, 127].

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - — a pointer was NULL
  - — `srcStep` was an odd number or its absolute value was less than 16 for YCbCr444 or 32 for YCbCr422/YCbCr420.
  - — a pointer in `pDstMCU[]` was not 8-byte aligned

4.4.3.8 JPEG-Specific YCbCr to RGB with Integrated Level Shift

4.4.3.8.1 YCbCr444ToRGB888LS_MCU_S16_U8_P3C3R

4.4.3.8.2 YCbCr422ToRGB888LS_MCU_S16_U8_P3C3R

4.4.3.8.3 YCbCr420ToRGB888LS_MCU_S16_U8_P3C3R

Prototype

```
OMXResult omxIPCS_YCbCr444ToRGB888LS_MCU_S16_U8_P3C3R(const OMX_S16
    *pSrcMCU[3], OMX_U8 *pDst, OMX_INT dstStep);
OMXResult omxIPCS_YCbCr422ToRGB888LS_MCU_S16_U8_P3C3R(const OMX_S16
    *pSrcMCU[3], OMX_U8 * pDst, OMX_INT dstStep);
OMXResult omxIPCS_YCbCr420ToRGB888LS_MCU_S16_U8_P3C3R(const OMX_S16
    *pSrcMCU[3], OMX_U8 *pDst, OMX_INT dstStep);
```

Description

These functions convert sub-sampled YCbCr data to RGB888 data with level-shift. Data is processed in MCUs for which the Y blocks have the following dimensions: YCbCr4:4:4: 8x8, YCbCr 4:2:2: 16x8; and YCbCr 4:2:0: 16x16.

Input Arguments

- `pSrcMCU` – buffer containing input MCU pointers: `pSrcMCU[0]` points to the Y block, `pSrcMCU[1]` points to Cb the block, and `pSrcMCU[2]` points to the Cr block; all three pointers must be aligned on 8-byte boundaries. Only top-down storage format is supported. Input components are expressed using a Q8 representation and are bounded on the interval [-128, 127].
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image; values less than 0 are allowed to support bottom-up storage format.

Output Arguments

- `pDst` – points to the output image buffer in which the output data (C3 representation) are interleaved as follows: BGRBGRBGR... Bottom-up storage format is supported. Outputs are saturated to the OMX_U8 range [0, 255]. The parameter `dstStep` can take negative values to support bottom-up storage format.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments. Returned if one or more of the following was true:
  - a pointer was NULL
  - the absolute value of `dstStep` was smaller than 24 (for YCbCr444) or 48 (for YCbCr422/YCbCr420)
  - a pointer in `pSrcMCU[]` was not 8-byte aligned

4.4.3.8.4 YCbCr444ToRGB565LS_MCU_S16_U16_P3C3R

4.4.3.8.5 YCbCr422ToRGB565LS_MCU_S16_U16_P3C3R

4.4.3.8.6 YCbCr420ToRGB565LS_MCU_S16_U16_P3C3R

Prototype

```
OMXResult omxIPCS_YCbCr444ToRGB565LS_MCU_S16_U16_P3C3R (const OMX_S16
    *pSrcMCU[3], OMX_U16 *pDst, OMX_INT dstStep);
OMXResult omxIPCS_YCbCr422ToRGB565LS_MCU_S16_U16_P3C3R (const OMX_S16
    *pSrcMCU[3], OMX_U16 *pDst, OMX_INT dstStep);
OMXResult omxIPCS_YCbCr420ToRGB565LS_MCU_S16_U16_P3C3R (const OMX_S16
    *pSrcMCU[3], OMX_U16 *pDst, OMX_INT dstStep);
```

Description

These functions convert sub-sampled YCbCr data to RGB565 data with level-shift. Data is processed in MCUs for which the Y blocks have the following dimensions: YCbCr4:4:4: 8x8, YCbCr 4:2:2: 16x8; and YCbCr 4:2:0: 16x16.

Input Arguments

- `pSrcMCU` – buffer containing input MCU pointers: `pSrcMCU[0]` points to the Y block, `pSrcMCU[1]` points to Cb the block, and `pSrcMCU[2]` points to the Cr block; all three must be aligned on 8-byte boundaries. Only top-down storage format is supported. Input components are represented using Q8 and are bounded on the interval [-128, 127].
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image; can take negative values to support bottom-up storage format.

Output Arguments

- `pDst` – output image buffer pointer; data are interleaved as follows: [R G B R G B R G B…], where G is represented using 6 bits, and B and R are represented using 5 bits. Output components are saturated. The parameter `dstStep` can take negative values to support bottom-up storage.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments - returned if one or more of the following is true:
  - — a pointer was NULL
  - — the absolute value of `dstStep` was smaller than 16 (for YCbCr444) or 32 (for YCbCr422/YCbCr420)
  - — a pointer in `pSrcMCU[]` was not 8-byte aligned

Image Coding

5

This section defines the functions and data types that comprise the OpenMAX DL image coding domain (omxIC) API, including functions that support construction of JPEG image codecs (omxICJP).

5.1 JPEG Sub-domain (omxICJP)

5.1.1 Definitions

5.1.1.1 JPEG Coefficient Buffer Organization

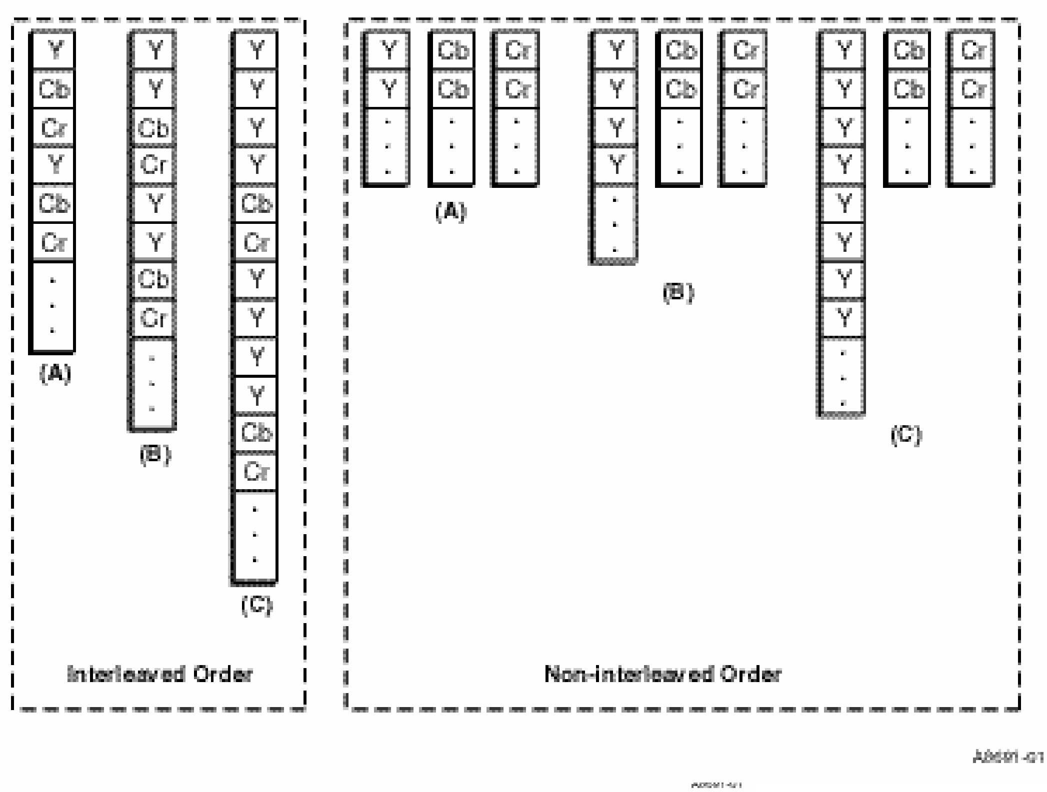
The omxICJP coefficient buffers contain 8x8 blocks of 16-bit signed values.

Figure 5-1 illustrates the memory organization of omxICJP buffers for both interleaved (left side) and planar (right side) images. In the buffers shown in the figure, each square represents an 8x8 block. For each color space, the example shows the representation of two MCUs. In particular,

- (A) depicts the non-subsampled YCbCr444 buffer organization. Each MCU contains three blocks (1 Y, 1 Cb, 1 Cr) for the planar representation or one block (1 Y or 1 Cb or 1 Cr) for the interleaved representation.
- (B) depicts YCbCr4:2:2 buffer organization. Each MCU contains four blocks (2 Y, 1 Cb, 1 Cr) for the planar representation or in general (2N blocks in Y buffer, N blocks in Cb and Cr buffer). The interleaved representation contains one block (1Y or 1Cb or 1Cr).
- (C) depicts YCbCr4:2:0 buffer organization. Each MCU contains six blocks (4 Y, 1 Cb, 1 Cr) or in general (4N blocks in Y buffer, N blocks in Cb and Cr buffer). The interleaved representation or one block (1Y or 1Cb or 1Cr) for interleaved order.

The start address of each coefficient buffer must be aligned on an 8-byte boundary.

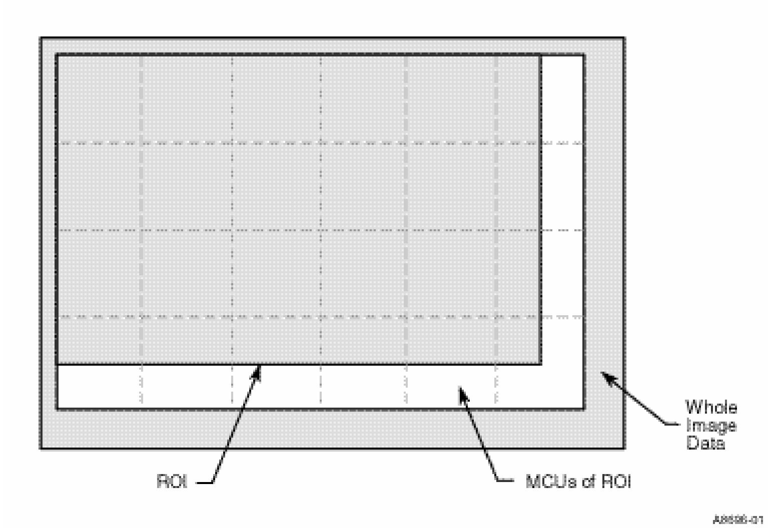
Figure 5-1: Interleaved and Non-Interleaved Image Data Formats



5.1.1.2 Image Representation

Figure 5-2: Rectangle Of Interest (ROI) for Encoding Procedure

Figure 5-2 illustrates the relationship between ROI, MCU of ROI, and the whole image data:



These parameters are used to indicate ROI:

- Start Pointer which points to the start address of ROI in the image data buffer
- Width is less than or equal to the width of the image
- Height is less than or equal to the height of the image
- Line Step of image data buffer

The procedure is:

1. Move the Start Pointer to indicate the start address of ROI.
2. Process MCUs on the boundary with padding function (Start Point, Width, Height, Line Step).
3. Encode each MCU

5.1.2 Data Structures

The following vendor-specific Huffman specification structures are required to support the Huffman encoder and decoder functions. Helper functions are defined to maintain portability.

- void OMXICJPHuffmanEncodeSpec;
- void OMXICJPHuffmanDecodeSpec;

5.1.3 Functions

5.1.3.1 Copy with Padding

5.1.3.1.1 CopyExpand_U8_C3

Prototype

```
OMXResult omxICJP_CopyExpand_U8_C3(const OMX_U8 *pSrc, OMX_INT srcStep,
   OMXSize srcSize, OMX_U8 *pDst, OMX_INT dstStep, OMXSize dstSize);
```

Description

This function copies an interleaved image from the source buffer to a larger buffer and pads the extra space with copies of the pixel values from the edges of the input image. For example, given positive source and destination step values (top-down source and destination images), the function first copies the source buffer to the destination buffer. In the process, the extra space in the larger destination rectangle to the right of the source rectangle is padded with copies of the right-most pixel from the source image on each scanline. The extra space in the larger destination rectangle below the source rectangle is padded with copies of the bottom-most pixel from the source image. This function processes only interleaved (C3) images.

Input Arguments

- `pSrc` – pointer to the source buffer; bottom-up storage is supported
- `srcStep` – distance in bytes between the starts of consecutive lines in the source image; can take negative values to support bottom-up storage
- `srcSize` – size of the source rectangle
- `dstStep` – distance in bytes between the starts of consecutive lines in the destination image; can take negative values to support bottom-up storage
- `dstSize` –size of destination rectangle

Output Arguments

- `pDst` – pointer to the destination buffer; bottom-up storage is supported

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments; returned under one or more of the following conditions:
  - — a pointer was NULL
  - — one of the source or destination region rectangle dimensions was 0
  - — the destination size was smaller than the source size
  - — absolute value of either `srcStep` or `dstStep` was less than 3

5.1.3.2 Forward DCT and Quantization

The forward DCT (FDCT) and inverse DCT (IDCT) used in all DCT/DCTQuant/IDCT/IDCTQuantInv functions defined in section 5.1.3.2 and 5.1.3.3 are given by

$$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

$$S_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v s_{vu} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

where

$$C_u, C_v = 1/\sqrt{2} \text{ for } u, v = 0$$

otherwise

$$C_u, C_v = 1$$

The quantization and inverse quantization operations are defined as follows.

$$Sq_{v,u} = round\_up\left(\frac{R_{v,u}}{Q_{v,u}}\right)$$

$$R_{v,u} = Sq_{v,u} \times Q_{v,u}$$

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992, sections A.3.3 FDCT and IDCT equations.

5.1.3.2.1 DCTQuantFwdTableInit

Prototype

```
OMXResult omxICJP_DCTQuantFwdTableInit(const OMX_U8 *pQuantRawTable,
   OMX_U16 *pQuantFwdTable);
```

Description

Initializes the JPEG DCT quantization table for 8-bit per component image data.

Input Arguments

- `pQuantRawTable` – pointer to the raw quantization table; must be aligned on an 8-byte boundary. The table must contain 64 entries.

Output Arguments

- `pQuantFwdTable` – pointer to the initialized quantization table; must be aligned on an 8-byte boundary. The table must contain 64 entries, and the implementation-specific contents must match the table contents expected by the associated set of forward DCT quantization functions in the same OpenMAX DL implementation, including the functions `DCTQuantFwd_S16`, `DCTQuantFwd_S16_I`, and `DCTQuantFwd_Multiple_S16`.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - a pointer was NULL
  - the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3 FDCT equation, and section A.3.4.

5.1.3.2.2 DCTQuantFwd_S16

5.1.3.2.3 DCTQuantFwd_S16_I

Prototype

```
OMXResult omxICJP_DCTQuantFwd_S16 (const OMX_S16* pSrc, OMX_S16 *pDst, const
    OMX_U16 *pQuantFwdTable);
OMXResult omxICJP_DCTQuantFwd_S16_I (OMX_S16* pSrcDst, const OMX_U16
    *pQuantFwdTable);
```

Description

Computes the forward DCT and quantizes the output coefficients for the 8-bit image data; processes a single 8x8 block.

Input Arguments

- `pSrc` – pointer to the input coefficient block (8x8) buffer. This start address must be 8-byte aligned. The input components are bounded on the interval [-128, 127].
- `pQuantFwdTable` – pointer to the 64 entry quantization table generated using `DCTQuantFwdTableInit`. The start address must be aligned on an 8-byte boundary.

Output Arguments

- `pDst` – pointer to the output coefficient block(8x8) buffer. This start address must be 8-byte aligned.

To achieve better performance, the output 8x8 matrix is the transpose of the explicit result. This transpose will be handled in Huffman encoding.

In-Out Arguments

`pSrcDst` – pointer to the coefficient block(8x8) buffer for in-place processing. This start address must be

8-byte aligned.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - — a pointer was NULL
  - — the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3 FDCT equation, and section A.3.4.

5.1.3.2.4 DCTFwd_S16

Prototype

```
OMXResult omxICJP_DCTFwd_S16(const OMX_S16* pSrc, OMX_S16 *pDst);
```

Description

Performs an 8x8 block forward discrete cosine transform (DCT). This function implements forward DCT for the 8-bit image data (packed into signed 16-bit). The output matrix is the transpose of the explicit result. As a result, the Huffman coding functions in this library handle transpose as well.

Input Arguments

- `pSrc` – pointer to the input coefficient block(8x8) buffer. This start address must be 8-byte aligned. The input components are bounded on the interval [-128, 127].

Output Arguments

- `pDst` – pointer to the output DCT coefficient block(8x8) buffer. This start address must be 8-byte aligned. To achieve better performance, the output 8x8 matrix is the transpose of the explicit result. This transpose can be handled in later processing stages (e.g. Huffman encoding).

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - — a pointer was NULL
  - — the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3, FDCT equation.

5.1.3.2.5 DCTFwd_S16_I

Prototype

```
OMXResult omxICJP_DCTFwd_S16_I(OMX_S16 *pSrcDst);
```

Description

This function implements forward DCT for the 8-bit image data (packed into signed 16-bit). It processes one block (8x8) in-place. The output matrix is the transpose of the explicit result. As a result, the Huffman coding functions in this library handle transpose as well.

In-Out Arguments

- `pSrcDst` – pointer to the input coefficient block(8x8) buffer for in-place processing. This start address must be 8-byte aligned. The input components are bounded on the interval [-128, 127] within a 16-bit container. To achieve better performance, the output 8x8 matrix is the transpose of the explicit result. This transpose can be handled in later processing stages (e.g. Huffman encoding).

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - a pointer was NULL
  - the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3, FDCT equation.

5.1.3.2.6 DCTQuantFwd_Multiple_S16

Prototype

```
OMXResult omxICJP_DCTQuantFwd_Multiple_S16 (const OMX_S16* pSrc, OMX_S16
    *pDst, OMX_INT nBlocks, const OMX_U16 *pQuantFwdTable);
```

Description

This function implements forward DCT with quantization for the 8-bit image data. It processes multiple adjacent blocks (8x8). The blocks are assumed to be part of a planarized buffer. This function needs to be called separately for luma and chroma buffers with the respective quantization table. The output matrix is the transpose of the explicit result. As a result, the Huffman coding functions in this library handle transpose as well.

Input Arguments

- `pSrc` – pointer to the input coefficient block(8x8) buffer. This start address must be 8-byte aligned. The input components are bounded on the interval [-128, 127] within a signed 16-bit container. Each 8x8 block in the buffer is stored as 64 entries (16-bit) linearly in a buffer, and the multiple blocks to be processed must be adjacent.

- `pQuantFwdTable` – pointer to the quantization table generated by "`DCTQuantFwdTableInit`". The table length is 64. This start address must be 8-byte aligned.
- `nBlocks` – the number of 8x8 blocks to be processed.

Output Arguments

- `pDst` – pointer to the output coefficient block (8x8) buffer. This start address must be 8-byte aligned. To achieve better performance, the output 8x8 matrix is the transpose of the explicit result. This transpose will be handled in Huffman encoding.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3, FDCT equation.

5.1.3.3 Inverse DCT and Inverse Quantization

5.1.3.3.1 DCTQuantInvTableInit

Prototype

```
OMXResult omxICJP_DCTQuantInvTableInit(const OMX_U8 *pQuantRawTable,
    OMX_U32 *pQuantInvTable);
```

Description

Initializes the JPEG IDCT inverse quantization table for 8-bit image data.

Input Arguments

- `pQuantRawTable` – pointer to the raw (unprocessed) quantization table, containing 64 entries; must be aligned on an 8-byte boundary.

Output Arguments

- `pQuantInvTable` – pointer to the initialized inverse quantization table; must be aligned on an 8-byte boundary. The table must contain 64 entries, and the implementation-specific contents must match the table contents expected by the associated set of inverse DCT quantization functions in the same OpenMAX DL implementation, including the functions `DCTQuantInv_S16`, `DCTQuantInv_S16_I`, and `DCTQuantInv_Multiple_S16`.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - — a pointer was NULL
  - — the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3 IDCT equation, and section A.3.4.

5.1.3.3.2 DCTQuantInv_S16

5.1.3.3.3 DCTQuantInv_S16_I

Prototype

```
OMXResult omxICJP_DCTQuantInv_S16(const OMX_S16 *pSrc, OMX_S16 *pDst, const
    OMX_U32 *pQuantInvTable);
OMXResult omxICJP_DCTQuantInv_S16_I (OMX_S16* pSrcDst, const OMX_U32
    *pQuantInvTable);
```

Description

Computes an inverse DCT and inverse quantization for 8-bit image data; processes one block (8x8).

Input Arguments

- `pSrc` – pointer to the input coefficient block (8x8) buffer. The start address must be 8-byte aligned.
- `pQuantInvTable` – pointer to the quantization table initialized using the function DCTQuantInvTableInit. The table contains 64 entries and the start address must be 8-byte aligned.

Output Arguments

- `pDst` – pointer to the output pixel block(8x8) buffer. The start address must be 8-byte aligned.

In-Out Arguments

- `pSrcDst` – pointer to the input coefficient block/output pixel block buffer (8x8) for in-place processing. The start address must be 8-byte aligned.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - — a pointer was NULL
  - — the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3 IDCT equation, and section A.3.4.

5.1.3.3.4 DCTInv_S16

Prototype

```
OMX_RESULT omxICJP_DCTInv_S16(const OMX_S16 *pSrc, OMX_S16 *pDst);
```

Description

This function implements inverse DCT for 8-bit image data. It processes one block (8x8).

Input Arguments

- `pSrc` – pointer to the input DCT coefficient block (8x8) buffer. The start address must be 8-byte aligned.

Output Arguments

- `pDst` – pointer to the output image pixel data block(8x8) buffer. The start address must be 8-byte aligned.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  — a pointer was NULL
  — the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3, IDCT equation.

5.1.3.3.5 DCTInv_S16_I

Prototype

```
OMXResult omxICJP_DCTInv_S16_I(OMX_S16 *pSrcDst);
```

Description

This function implements an inverse DCT for 8-bit image data. It processes one block (8x8) in an in-place fashion.

In-Out Arguments

- `pSrcDst` – pointer to the shared buffer for the input DCT coefficient block (8x8) and the output image pixel data block. The start address must be 8-byte aligned.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  — a pointer was NULL
  — the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3, IDCT equation.

5.1.3.3.6 DCTQuantInv_Multiple_S16

Prototype

```
OMXResult omxICJP_DCTQuantInv_Multiple_S16(const OMX_S16 *pSrc, OMX_S16
   *pDst, OMX_INT nBlocks, const OMX_U32 *pQuantInvTable);
```

Description

Multiple block dequantization and IDCT function. This function implements inverse DCT with dequantization for 8-bit image data. It processes multiple blocks (each 8x8). The blocks are assumed to be part of a planarized buffer. This function needs to be called separately for luma and chroma buffers with the respective quantization table. The start address of `pQuantInvTable` must be 8-byte aligned.

Input Arguments

- `pSrc` – pointer to the input coefficient block (8x8) buffer. The start address must be 8-byte aligned.
- `nBlocks` – the number of 8x8 blocks to be processed.
- `pQuantInvTable` – pointer to the quantization table initialized using the function DCTQuantInvTableInit. The table contains 64 entries and the start address must be 8-byte aligned..

Output Arguments

- `pDst` – pointer to the output pixel block (8x8) buffer. The start address must be 8-byte aligned.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - a pointer was NULL
  - the start address of a pointer was not 8-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992. Section A.3.3 IDCT equation, and section A.3.4.

5.1.3.4 Huffman Encoding

5.1.3.4.1 EncodeHuffmanSpecGetBufSize_U8

Prototype

```
OMXResult omxICJP_EncodeHuffmanSpecGetBufSize_U8 (OMX_INT* pSize);
```

Description

Returns the size, in bytes, of the buffer required to store the Huffman encoder table.

Input Arguments

- none

Output Arguments

- `pSize` - pointer to the size

Returns

- `OMX_StsNoErr` - no error
- `OMX_StsBadArgErr` - bad arguments
  — a pointer was NULL

5.1.3.4.2 EncodeHuffmanSpecInit_U8

Prototype

```
OMXResult omxICJP_EncodeHuffmanSpecInit_U8 (const OMX_U8 *pHuffBits, const
    OMX_U8 *pHuffValue, OMXICJPHuffmanEncodeSpec *pHuffTable);
```

Description

Initializes the DC or AC Huffman encoder table specification.

Input Arguments

- `pHuffBits` - Pointer to the array of HUFFBITS, which contains the number of Huffman codes for size 1-16.
- `pHuffValue` - Pointer to the array of HUFFVAL, which contains the symbol values to be associated with the Huffman codes ordering by size.

Output Arguments

- `pHuffTable` – pointer to the an `OMXICJPHuffmanEncodeSpec` data structure; must be aligned on a 4-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  — a pointer was NULL

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992, section C.2 and figures C-1 through C-3.

5.1.3.4.3 EncodeHuffman8x8_Direct_S16_U1_C1

Prototype

```
OMXResult omxICJP_EncodeHuffman8x8_Direct_S16_U1_C1 (const OMX_S16 *pSrc,
   OMX_U8 *pDst, OMX_INT *pDstBitsLen, OMX_S16 *pDCPred, const
   OMXICJPHuffmanEncodeSpec *pDCHuffTable, const OMXICJPHuffmanEncodeSpec
   *pACHuffTable);
```

Description

Implements the Huffman encoder for baseline mode. The DC prediction coefficient (`*pDCPred`) should be initialized to zero and reset to zero after every restart interval.

Input Arguments

- `pSrc` – pointer to the source data block (8x8); must be aligned on a 32-byte boundary.
- `pDCHuffTable` – pointer to the `OMXICJPHuffmanEncodeSpec` data structure containing the DC Huffman encoder table; must be aligned on a 4-byte boundary.
- `pACHuffTable` – pointer to the `OMXICJPHuffmanEncodeSpec` data structure containing the AC Huffman encoder table; must be aligned on a 4-byte boundary.

Output Arguments

- `pDst` – pointer to the output bitstream buffer; must be aligned on a 4-byte boundary

Input-Output Arguments

- `pDstBitsLen` – stream length parameter; upon input indicates the number of valid bits in `pDst`. Updated upon return to indicate the number of valid bits in `pDst` after block encoding has been completed; must be aligned on a 4-byte boundary.
- `pDCPred` – pointer to the DC prediction coefficient. Upon input should contain the value of the quantized DC coefficient from the most recently coded block. Updated upon return to contain the DC coefficient from the current block; the pointer must be aligned on a 4-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - — a pointer was NULL
  - — the start address of a pointer was not 4-byte aligned.
  - — `*pDstBitsLen` was less than 0.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992, Annex D-1.

5.1.3.5 Huffman Decoding

5.1.3.5.1 DecodeHuffmanSpecGetBufSize_U8

Prototype

```
OMXResult omxICJP_DecodeHuffmanSpecGetBufSize_U8 (OMX_INT* pSize);
```

Description

Returns the size, in bytes, of the buffer required to store the Huffman decoder table.

Input Arguments

- none

Output Arguments

- `pSize` - pointer to the size

Returns

- `OMX_StsNoErr` - no error
- `OMX_StsBadArgErr` - bad arguments
- If following conditions are not satisfied, this function returns `OMX_StsBadArgErr`:
  - pointer cannot be NULL

5.1.3.5.2 DecodeHuffmanSpecInit_U8

Prototype

```
OMXResult omxICJP_DecodeHuffmanSpecInit_U8 (const OMX_U8 *pHuffBits, const
   OMX_U8 *pHuffValue, OMXICJPHuffmanDecodeSpec *pHuffTable);
```

Description

Initializes the DC or AC Huffman decoder table specification.

Input Arguments

- `pHuffBits` – Pointer to the array of HUFFBITS, which contains the number of Huffman codes for size 1-16
- `pHuffValue` – Pointer to the array of HUFFVAL, which contains the symbol values to be associated with the Huffman codes ordering by size

Output Arguments

- `pHuffTable` – pointer to a `OMXICJPHuffmanDecodeSpec` data structure; must be aligned on a 4-byte boundary.

Returns

- `OMX_StsNoErr` – no error

- `OMX_StsBadArgErr` – Bad arguments. Returned for any of the following conditions:
  - a pointer was NULL

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992, section C.2 and figures C-1 through C-3.

5.1.3.5.3 DecodeHuffman8x8_Direct_S16_C1

Prototype

```
OMXResult omxICJP_DecodeHuffman8x8_Direct_S16_C1 (const OMX_U8 *pSrc,
   OMX_INT *pSrcBitsLen, OMX_S16 *pDst, OMX_S16 *pDCPred, OMX_INT *pMarker,
   OMX_U32 *pPrefetchedBits, OMX_INT *pNumValidPrefetchedBits, const
   OMXICJPHuffmanDecodeSpec *pDCHuffTable, const OMXICJPHuffmanDecodeSpec
   *pACHuffTable);
```

Description

Implements the JPEG baseline Huffman decoder. Decodes an 8x8 block of quantized DCT coefficients using the tables referenced by the parameters `pDCHuffTable` and `pACHuffTable` in accordance with the Huffman decoding procedure defined in ISO10918, Annex F.2.2, *Baseline Huffman Decoding Procedures*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to the location indicated by `pMarker`. The DC coefficient prediction parameter `pDCPred` should be set to 0 during initialization and after every restart interval. The parameter `pMarker` should set to 0 during initialization or after the found marker has been processed. The parameter `pNumValidPrefetchedBits` should be set to 0 in the following cases: 1) during function initialization, 2) after each restart interval, and 3) after each found marker has been processed. The parameter `pPrefetchedBits` should be set to 0 during function initialization.

Input Arguments

- `pSrc` – pointer to the source JPEG bitstream buffer
- `pDCHuffTable` – pointer to the `OMXICJPHuffmanDecodeSpec` structure containing the DC Huffman decoding table; must be aligned on a 4-byte boundary.
- `pACHuffTable` – pointer to the `OMXICJPHuffmanDecodeSpec` structure containing the AC Huffman decoding table; must be aligned on a 4-byte aligned.

Output Arguments

- `pDst` – pointer to the output buffer; must be aligned on a 32-byte boundary.

In-Out Arguments

- `pDCPred` – pointer to the DC prediction coefficient. Upon input contains the quantized DC coefficient decoded from the most recent block. Upon output returns the quantized value of the DC coefficient from the current block. Should be set to 0 upon function initialization and after each restart interval.

- `pMarker` – pointer to the most recently encountered marker. If a marker is detected during decoding, the function stops decoding and returns the encountered marker using this parameter. The caller should set this parameter to 0 during function initialization and after a found marker has been processed.
- `pSrcBitsLen` – pointer to the number of bits that have been decoded.
- `pPrefetchedBits` – implementation-specific pre-fetch parameter; should be set to 0 during function initialization.
- `pNumValidPrefetchedBits` – pointer to the number of valid bits in the pre-fetch buffer; should be set to 0 under the following conditions: 1) function initialization, 2) after each restart interval, 3) after each found marker has been processed.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsJPEGMarkerWarn` – JPEG marker encountered; Huffman decoding terminated early.
- `OMX_StsBadArgErr` – Bad arguments. Returnedu nder any of the following conditions:
  - a pointer was NULL
  - `*pSrcBitsLen` was less than 0.
  - `*pNumValidPrefetchedBits` was less than 0.
  - the start address of `pDst` was not 32-byte aligned.

Reference

CCITT T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Image – Requirements and Guidelines, Sep. 1992, Annex D-2.

Video Coding

6

This section defines the functions and data types that comprise the OpenMAX DL video coding domain (omxVC), including functions that can be used to construct the MPEG-4 simple profile encoder and decoder (omxVCM4P2), as well as functions that can be used to construct the H.264 baseline encoder and decoder (omxVCM4P10). A sub-domain containing a set of functions common to MPEG-4 and H.264 sub-domains is also defined (omxVCCOMM).

6.1 Common Sub-Domain (omxVCCOMM)

The omxVCCOMM sub-domain defines data structures and functions that could be used in conjunction with data structures and functions from other omxVC sub-domains including omxVCM4P2 and omxVCM4P10 for implementation of simple profile MPEG-4 as well as baseline profile H.264 encoders or decoders.

6.1.1 Data Structures and Enumerators

The omxVCCOMM sub-domain defines data structures and bitstream buffer conventions that are common across all omxVC sub-domains.

6.1.1.1 Motion Vectors

In omxVC, motion vectors are represented as follows:

```
typedef struct {
   OMX_S16 dx;
   OMX_S16 dy;
} OMXVCMotionVector;
```

Unless otherwise specified, texture motion vectors are represented using Q1.

6.1.1.2 Rectangle

The geometric position and size of a rectangle are represented as follows:

```
typedef struct {
   OMX_INT x;
   OMX_INT y;
   OMX_INT width;
   OMX_INT height;
} OMXRect;
```

where x and y specify the coordinates of the top left corner of the rectangle, and the parameters width and height specify dimensions in the x- and y- directions, respectively.

6.1.2 Buffer Conventions

6.1.2.1 Bitstream Buffers

In omxVC, bitstreams are represented using two parameters, namely, a double pointer to the stream buffer, `**ppBitStream`, and a pointer to the next available bit in the stream, `*pBitOffset`. Unless otherwise specified in the description for a particular function, the standard conventions that are observed for stream buffers and buffer pointer maintainence are as follows:

- The parameter `**ppBitStream` points to the current byte in the stream upon function entry, and is updated by the function such that it references the current byte in the stream upon function exit.

- The parameter `*pBitOffset` points to the next available bit in the stream upon function entry, and is updated by the function such that it points to the next available bit in the stream upon function exit. `*pBitOffset` is valid in the range 0 to 7.
- Stream buffer space is allocated outside of the function and is maintained by the DL user, client, or application.
- It is recommended in all cases that eight additional padding bytes beyond the minimum required buffer size be allocated to a stream buffer in order to protect against data aborts under exception conditions.

These standard bitstream conventions apply to a particular set of functions from the omxVCM4P2 and omxVCM4P10 sub-domains, including:

```
omxVCM4P2_EncodeVLCZigzag_IntraDCVLC_S16
omxVCM4P2_EncodeVLCZigzag_IntraACVLC_S16
omxVCM4P2_EncodeVLCZigzag_Inter_S16
omxVCM4P2_DecodeVLCZigzag_IntraDCVLC_S16
omxVCM4P2_DecodeVLCZigzag_IntraACVLC_S16
omxVCM4P2_EncodeMV_U8_S16
omxVCM4P2_DecodePadMV_PVOP
omxVCM4P2_DecodeVLCZigzag_Inter_S16
omxVCM4P2_DecodeBlockCoef_Intra_U8
omxVCM4P2_DecodeBlockCoef_Inter_S16
omxVCM4P10_DecodeChromaDcCoeffsToPairCAVLC_U8
omxVCM4P10_DecodeCoeffsToPairCAVLC_U8
```

6.1.3 Functions

6.1.3.1 ComputeTextureErrorBlock_SAD

Prototype

```
OMXResult omxVCCOMM_ComputeTextureErrorBlock_SAD(const OMX_U8 *pSrc, OMX_INT
    srcStep, const OMX_U8 *pSrcRef, OMX_S16 * pDst, OMX_INT *pDstSAD);
```

Description

Computes texture error of the block; also returns SAD.

Input Arguments

- `pSrc` – pointer to the source plane; must be aligned on an 8-byte boundary.
- `srcStep` – step of the source plane
- `pSrcRef` – pointer to the reference buffer, an 8x8 block; must be aligned on an 8-byte boundary.

Output Arguments

- `pDst` – pointer to the destination buffer, an 8x8 block; must be aligned on an 8-byte boundary.
- `pDstSAD` – pointer to the Sum of Absolute Differences (SAD) value

Returns

- `OMX_StsNoErr` – no error

- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers is NULL: `pSrc`, `pSrcRef`, `pDst` and `pDstSAD`.
  - — `pSrc` is not 8-byte aligned.
  - — `SrcStep` <= 0 or `srcStep` is not a multiple of 8.
  - — `pSrcRef` is not 8-byte aligned.
  - — `pDst` is not 8-byte aligned.

6.1.3.2 ComputeTextureErrorBlock

Prototype

```
OMXResult omxVCCOMM_ComputeTextureErrorBlock(const OMX_U8 *pSrc, OMX_INT
   srcStep, const OMX_U8 *pSrcRef, OMX_S16 * pDst);
```

Description

Computes the texture error of the block.

Input Arguments

- `pSrc` – pointer to the source plane. This should be aligned on an 8-byte boundary.
- `srcStep` – step of the source plane
- `pSrcRef` – pointer to the reference buffer, an 8x8 block. This should be aligned on an 8-byte boundary.

Output Arguments

- `pDst` – pointer to the destination buffer, an 8x8 block. This should be aligned on an 8-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers is NULL: `pSrc`, `pSrcRef`, `pDst`.
  - — `pSrc` is not 8-byte aligned.
  - — `SrcStep` <= 0 or `srcStep` is not a multiple of 8.
  - — `pSrcRef` is not 8-byte aligned.
  - — `pDst` is not 8-byte aligned

6.1.3.3 LimitMVToRect

Prototype

```
OMXResult omxVCCOMM_LimitMVToRect(const OMXVCMotionVector * pSrcMV,
   OMXVCMotionVector *pDstMV, OMXRect * pRectVOPRef, OMX_INT Xcoord,
   OMX_INT Ycoord, OMX_INT size);
```

Description

Limit the motion vector of current block/macroblock into the expanded bounding rectangle.

Input Arguments

- `pSrcMV` – pointer to the motion vector of current block or macroblock
- `pRectVOPRef` – pointer to the bounding rectangle
- `Xcoord`, `Ycoord` – the coordinates of the current block or macroblock
- `size` – the size of block or macroblock

Output Arguments

- `pDstMV` – pointer to the limited motion vector

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers is NULL: `pSrcMV`, `pDstMV`, or `pRectVOPRef`. or
  - — At least one of following case is true: size is neither `BLOCK_SIZE` nor `MB_SIZE`; the width (or height) of rectangle is less than twice the of size.

6.1.3.4 SAD_16x

Prototype

```
OMXResult omxVCCOMM_SAD_16x (OMX_U8* pSrcOrg, OMX_U32 iStepOrg OMX_U8*
   pSrcRef, OMX_U32 iStepRef, OMX_S32* pDstSAD, OMX_U32 iHeight);
```

Description

This function calculates the SAD for 16x16 and 16x8 blocks.

Input Arguments

- `pSrcOrg` - Pointer to the original block; must be aligned on a 16-byte boundary.
- `iStepOrg` - Step of the original block buffer
- `pSrcRef` - Pointer to the reference block
- `iStepRef` - Step of the reference block buffer
- `iHeight` - Height of the block

Output Parameters

- `pDstSAD` - Pointer of result SAD

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `iHeight` is not equal to either 8 or 16.

6.1.3.5 SAD_8x

Prototype

```
OMXResult omxVCCOMM_SAD_8x (OMX_U8* pSrcOrg, OMX_U32 iStepOrg, OMX_U8*
   pSrcRef, OMX_U32 iStepRef, OMX_S32*pDstSAD, OMX_U32iHeight)
```

Description

This function calculates the SAD for 8x16, 8x8, 8x4 blocks.

Input Arguments

- `pSrcOrg` - Pointer to the original block; must be aligned on a 8-byte boundary.
- `iStepOrg` - Step of the original block buffer
- `pSrcRef` - Pointer to the reference block
- `iStepRef` - Step of the reference block buffer
- `iHeight` - Height of the block

Output Parameters

- `pDstSAD` - Pointer of result SAD

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `iHeight` is not equal to either 4, 8, or 16.

6.1.3.6 Average_8x

Prototype

```
OMXResult omxVCCOMM_Average_8x (OMX_U8* pPred0, OMX_U8* pPred1, OMX_U32
   iPredStep0, OMX_U32 iPredStep1, OMX_U8* pDstPred, OMX_U32 iDstStep,
   OMX_U32 iHeight);
```

Description

This function calculates the average of two 8x4, 8x8, or 8x16 blocks. The result is rounded according to (a+b+1)/2. The block average function can be used in conjunction with half-pixel interpolation to obtain quarter pixel motion estimates, as described in subclause 8.4.2.2.1 of ISO/IEC 14496-10.

Input Parameters

- `pPred0` - Pointer to the top-left corner of reference block 0
- `pPred1` - Pointer to the top-left corner of reference block 1
- `iPredStep0` - Step of reference block 0
- `iPredStep1` - Step of reference block 1
- `iDstStep` - Step of the destination buffer.
- `iHeight` - Height of the blocks

Output Parameters

- `pDstPred` - Pointer to the destination buffer. 8-byte aligned.

Reference

ISO/IEC 14496-10, subclause 8.4.2.2.1, Eq. 8.194 – 8.205

6.1.3.7 Average_16x

Prototype

```
OMXResult omxVCCOMM_Average_16x (OMX_U8* pPred0, OMX_U8* pPred1, OMX_U32
   iPredStep0, OMX_U32 iPredStep1, OMX_U8* pDstPred, OMX_U32 iDstStep,
   OMX_U32 iHeight);
```

Description

This function calculates the average of two 16x16 or 16x8 blocks. The result is rounded according to (a+b+1)/2. The block average function can be used in conjunction with half-pixel interpolation to obtain quarter pixel motion estimates, as described in subclause 8.4.2.2.1 of ISO/IEC 14496-10.

Input Parameters

- `pPred0` - Pointer to the top-left corner of reference block 0
- `pPred1` - Pointer to the top-left corner of reference block 1
- `iPredStep0` - Step of reference block 0
- `iPredStep1` - Step of reference block 1
- `iDstStep` - Step of the destination buffer
- `iHeight` - Height of the blocks

Output Parameters

- `pDstPred` - Pointer to the destination buffer. 16-byte aligned.

Reference

ISO/IEC 14496-10, subclause 8.4.2.2.1, Eq. 8.194 – 8.205

6.1.3.8 ExpandFrame

Prototype

```
OMXResult omxVCCOMM_ExpandFrame(OMX_U8* pSrcDstPlane, OMX_U32 iFrameWidth,
   OMX_U32 iFrameHeight, OMX_U32 iExpandPels, OMX_U32 iPlaneStep);
```

Description

This function expands a reconstructed frame. The unexpanded frame is stored in a plane buffer with preserved space for edge expansion, so what we do is only filling out the edge pixels. (A plane is like a container, the frame is in the center of it).

Input Parameters

- `pSrcDstPlane` - Pointer to the top-left corner of the frame to be expanded. 32-byte aligned.
- `iFrameWidth` - Width of the frame.
- `iFrameHeight` - Height of the frame.
- `iExpandPels` - Number of pixels to be expanded in one direction.
- `iPlaneStep` - Width of the plane buffer. iPlaneStep should be greater than (`iFrameWidth` + 2 * `iExpandPels`).

Output Parameters

- `pSrcDstPlane` - Pointer to the top-left corner of the frame (NOT the top-left corner of the plane).

6.2 MPEG-4 Simple Profile Sub-Domain (omxVCM4P2)

This section defines the omxVCM4P2 sub-domain, which includes data structures and functions that could be used to construct an MPEG-4 simple profile encoder or decoder.

6.2.1 Data Structures and Enumerators

6.2.1.1 Direction

The direction enumerator is used with functions that perform AC/DC prediction and zig-zag scan.

```
enum {
    OMX_VC_NONE= 0,
    OMX_VC_HORIZONTAL= 1,
    OMX_VC_VERTICAL= 2
};
```

6.2.1.2 Bilinear Interpolation

The bilinear interpolation enumerator is used with motion estimation, motion compensation, and reconstruction functions.

```
enum {
    OMX_VC_INTEGER_PIXEL= 0,    /* case a */
    OMX_VC_HALF_PIXEL_X= 1,     /* case b  */
    OMX_VC_HALF_PIXEL_Y= 2,     /* case c  */
    OMX_VC_HALF_PIXEL_XY= 3     /* case d  */
};
```

6.2.1.3 Neighboring Macroblock Availability

Neighboring macroblock availability is indicated using the following flags:

```
enum {
    OMX_VC_UPPER  = 1,          /** above macroblock is available */
    OMX_VC_LEFT   = 2,          /** left macroblock is available */
    OMX_VC_CENTER = 4,
    OMX_VC_RIGHT  = 8,
    OMX_VC_LOWER  = 16,
    OMX_VC_UPPER_LEFT  = 32,    /** above-left macroblock is available */
    OMX_VC_UPPER_RIGHT = 64,    /** above-right macroblock is available */
    OMX_VC_LOWER_LEFT  = 128,
    OMX_VC_LOWER_RIGHT = 256
```

```
};
```

6.2.1.4 Video Components

A data type that enumerates video components is defined as follows:

```
typedef enum {
    OMX_VC_LUMINANCE,      /** Luminance component */
    OMX_VC_CHROMINANCE,    /** chrominance component */
    OMX_VC_ALPHA           /** Alpha component */
} OMXVCM4P2VideoComponent;
```

6.2.1.5 MacroblockTypes

A data type that enumerates macroblock types is defined as follows:

```
typedef enum {
    OMX_VC_INTER    = 0,   /** P picture or P-VOP */
    OMX_VC_INTER_Q= 1,     /** P picture or P-VOP */
    OMX_VC_INTER4V= 2,     /** P picture or P-VOP */
    OMX_VC_INTRA    = 3,   /** I and P picture, I- and P-VOP */
    OMX_VC_INTRA_Q= 4,     /** I and P picture, I- and P-VOP */
    OMX_VC_INTER4V_Q= 5,   /** P picture or P-VOP (H.263)*/
} OMXVCM4P2MacroblockType;
```

6.2.1.6 Coordinates

Coordinates are represented as follows:

```
typedef struct {
    OMX_INT x;
    OMX_INT y;
} OMXVCM4P2Coordinate;
```

6.2.1.7 Motion Estimation Algorithms

A data type that enumerates motion estimation modes is defined as follows:

```
typedef enum
{
    OMX_VCM4P2_FAST_SEARCH = 0, /** Fast motion search */
    OMX_VCM4P2_FULL_SEARCH = 1; /** Full motion search */
} OMXVCM4P2MEMode;
```

6.2.1.8 Quantization Weighting Coefficients

Quantization weighting coefficients are represented as follows:

```
typedef struct {
  OMX_U8  Quant[64];           /** weights, represented using Q0 */
  OMX_U32 RecipQuant[64];      /** reciprocal weights, represented using Q21
} OMXVCM4P2QuantWeightCoef;
```

6.2.2 Buffer Conventions

6.2.2.1 Pixel Planes

The encoder's input and output is stored in pixel planes denoted by Y plane (luminance component), Cb plane and Cr plane (chrominance components).

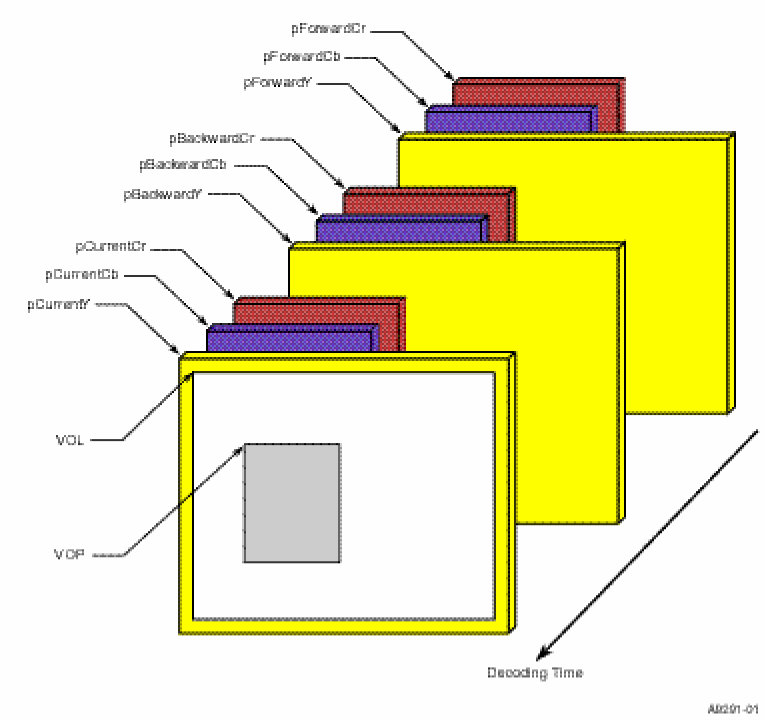
The size of Y plane relates to, but is not equal to, that defined in the VOL as a result of the VOP expansion. Since luminance VOP is expanded (and padded) with 16 pixels to each of the four directions, the width and height of the Y plane are 32 pixels larger than those defined in the VOL respectively.

The size (W, H) of Cb or Cr plane is half the size of Y plane, because chrominance VOPs are expanded with eight pixels in each direction.

Figure 6-1 shows the relationship among the pixel planes VOL and VOP.

Allocate three sets of 32-bit word-aligned pixel planes, each consisting of a Y plane, a Cb plane and a Cr plane.

Figure 6-1: Pixel Plane, VOL, and VOP



6.2.2.2 Texture Motion Vectors (in Q1 format)

Zero to four valid motion vector(s) are associated with each MB, depending on MB type. A motion vector (MV) buffer contains four block-based elements stored contiguously. One MV buffer per MB for shall be allocated for a P-VOP. The following conventions are adopted:

- Two buffers per MB in P-VOP are contained in `pMVForward[4]`
- Elements are block-based and stored contiguously
- If the MB type is "`OMX_VC_INTER`"or "`OMX_VC_INTER_Q` ", and if not transparent, `pMVForward[0]-[3]` must be filled with the same decoded MV.
- If MB is INTRA coded or skipped, pMVForward[0]-[3] must be padded with zero MVs.

Coordinates are related to the absolute coordinate system shown in Figure 7-19 of *ISO/IEC 14496-2*: Information Technology - Generic Coding of Audio-Visual Objects - Part 2: Visual (FD, October 1998).

6.2.2.3 Quantization Parameter

Quantization parameters of intra-coded macroblocks must be stored to perform DC and AC prediction for the intra-coded macroblocks spatially to the right and/or below, if they exist.

- One row buffer for the current VOP is used for coefficient prediction
- Before decoding an `intra` or `intra+q` MB, the buffer saves the QPs of the upper MB and left MB if they exist.

- After an `intra` or `intra+q` MB is decoded, the corresponding QP buffer, which stored the MB spatially above before must be updated by the current QP.
- Each element is one byte (`OMX_U8`) for one MB.

6.2.2.4 Coefficient buffers

Two coefficient buffers should be allocated for Intra DC/AC prediction – a row buffer that contains ((mb_num_per_row * 2 + 1) * 8) elements of `OMX_S16`, and a column buffer that contains 16 elements of `OMX_S16`.

Every eight elements of both row and column buffers, plus one element eight units ahead in row buffer, are used to perform DC/AC prediction for an INTRA coded block in a MB. Each group stores the coefficient predictors of the neighbor block spatially above or to the left of the block currently to be decoded. Within every group of eight elements, the first element stores the DC coefficient and the others store quantized AC coefficients. A negative-valued DC coefficient signals that this neighbor block is not INTRA coded, and therefore neither the DC nor the AC coefficients are valid.

All DC elements in the row buffer must be initialized to -1 prior to decoding each VOP. In addition, the two DC elements in column buffer should also be initialized to -1 prior decoding each MB row.

If the current MB_Type is either `OMX_VC_INTER/ OMX_VC_INTER_Q/ OMX_VC_INTER_4V`, then the corresponding DC elements in the row buffer and column buffer must also be initialized to –1 to indicate that no predictor for later AC/DC prediction.

The detailed coefficient buffer layout is illustrated in Figure 6-3.

6.2.2.4.1 Internal Prediction Coefficient Buffer Update Procedures

The following prediction coefficient update procedures are followed inside of the omxM4P2 functions that use row and column prediction coefficient buffers:

1. INTRA Frame AC Coefficient Buffer Update

After encoding each 8x8 block, the AC coefficients from the first row and column are copied, respectively, to the row and column coefficient prediction buffers (Fig. 6-2).

2. INTRA Frame DC Coefficient Buffer Update

A special DC coefficient update procedure is followed inside of the function to avoid overwriting the top-left DC prediction component for the right-hand block. The DC coefficient buffers are updated as follows:

(pPredBufRow - 8) = pPredBufCol; /* Fig. 6-2, Step 1 */

*pPredBufCol = *pDCTcoef; /* Fig. 6-2, Step 2 */

Figure 6-1: Row/Column Coefficient Buffer Updates for A Single Block

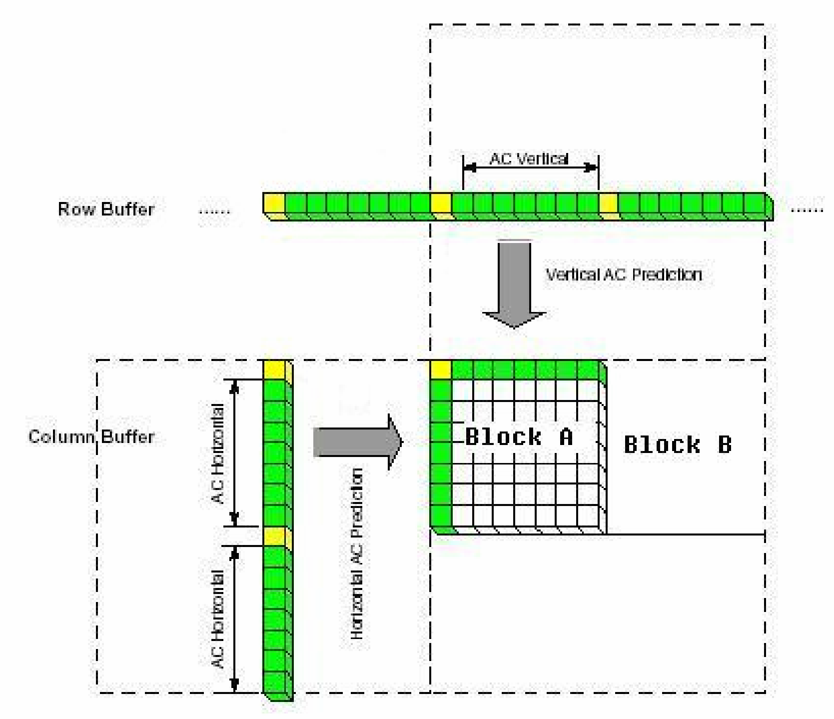
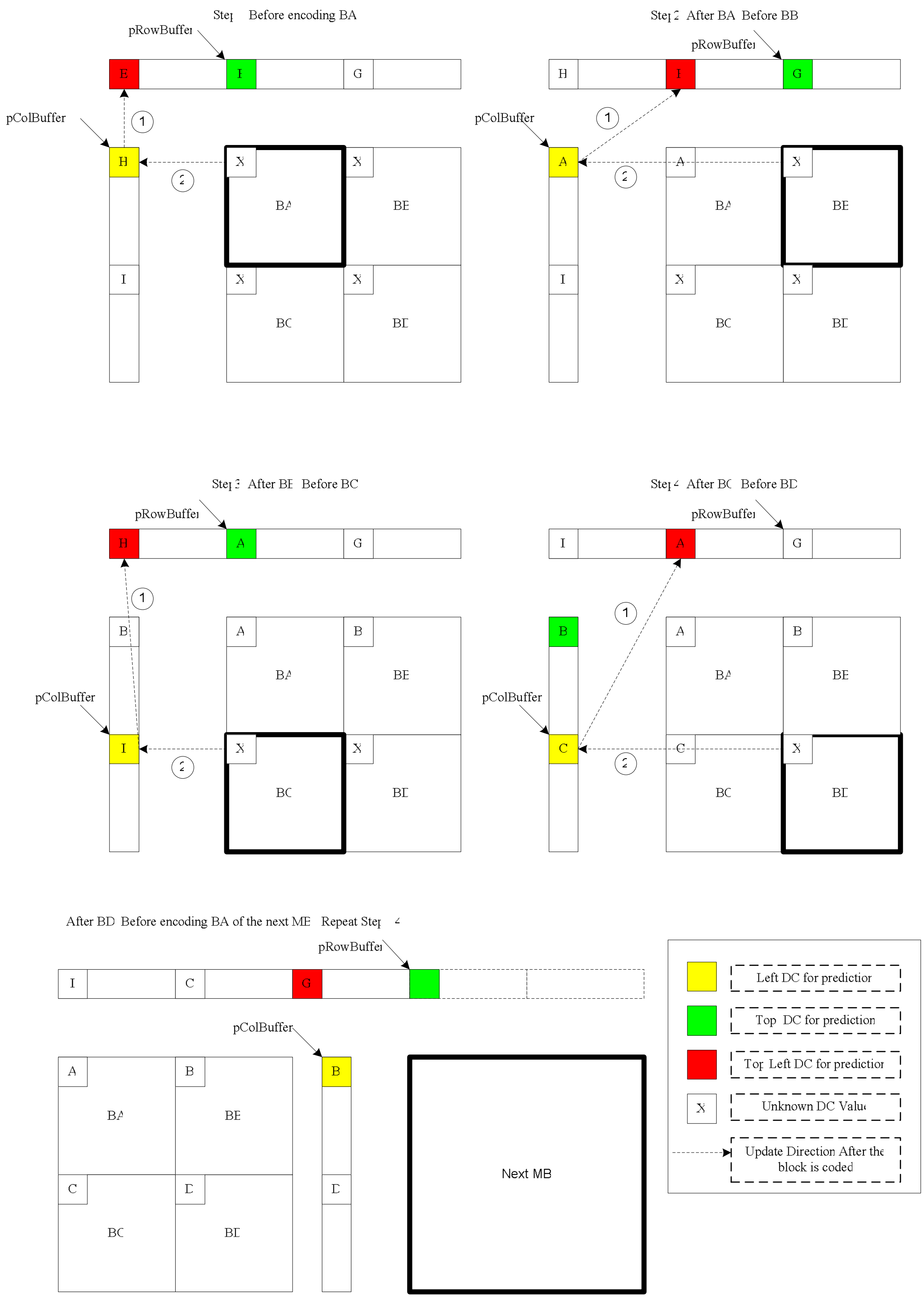


Figure 6-2 illustrates the DC coefficient update process during encoding of a complete macroblock. As shown in the figure, after block A is encoded, the DC value will first be copied to the column buffer. Then, after block B is encoded, this same DC coefficient will be moved to the row buffer from the column buffer**.** The dotted lines illustrate how the DC coefficients are updated after encoding the 8x8 block. For blocks BA, BB, and BC, the top DC component will be stored in location pPredBulRow[0]. For block BD, the top DC coefficient will be stored in location pPredBufCol[0], i.e.,

```
if( 3 == blockIndex ) {
tempPred = *(pPredBufCol-8);
} else {
tempPred = * pPredBufRow;
}
```

Figure 6-2: Row/Column Coefficient Buffer Updates for A Complete Macroblock



6.2.2.4.2 External Prediction Coefficient Buffer Update Procedures

The following prediction coefficient update procedures should be implemented outside of the omxM4P2 functions that use row and column prediction coefficient buffers:

1. INTRA Frame DC Coefficient Buffer Update

I (the DC coefficient of block BD in the previous MB) is updated to the location of H after encoding BC of the current MB. Similarly, D (the DC coefficient of block BD in the current MB) coefficient will be copied to the location of G after encoding BC of the next macroblock. This procedure works except the last MB in a row because the DC coefficient of block BD in the last MB of a row will not be updated into the row buffer. For an INTRA MB, the row coefficient prediction buffer must be updated externally after encoding the last MB of a row using the following procedure:

```
//Special Upate for the last MB in the ROW
//pCoefBufRow – Pointers to the current position in Row Buffer
//pCoefBufCol – Pointers to the current position in Column Buffer
//pCoefBufRowStart – Pointers to the start of Row Buffer
//pCoefBufColStart – Pointers to the start of Column Buffer

if (Last MB in a row) {
  if (MBType == INTRA) {
    *( pCoefBufRow->pYPtr + 8 ) = *( pCoefBufCol->pYPtr + 8 );
    *( pCoefBufRow->pCbPtr ) = *( pCoefBufCol->pCbPtr );
    *( pCoefBufRow->pCrPtr ) =  *( pCoefBufCol->pCrPtr );
  }

  //Update Row Buffer Pointers
  pCoefBufRow->pYPtr= pCoefBufRowStart->pYPtr +MB_SIZE;
  pCoefBufRow->pCbPtr= pCoefBufRowStart->pCbPtr + BLOCK_SIZE;
  pCoefBufRow->pCrPtr= pCoefBufRowStart->pCrPtr + BLOCK_SIZE;

} else {
  //Update Row Buffer Pointers
  pCoefBufRow->pYPtr+=MB_SIZE;
  pCoefBufRow->pCbPtr+=BLOCK_SIZE;
  pCoefBufRow->pCrPtr+=BLOCK_SIZE;
}
```

2. P-Frame Prediction Coefficient Buffer Update

The prediction coefficient buffer pointers are not updated during encoding P-MBs, and therefore the following update procedure should be implemented after encoding a P-MB:

```
//Update the right-bottom block DC of BD of previous MB
*(pCoefBufRow->pYPtr - 8) = *( pCoefBufCol->pYPtr + 8 );
//Mark the buffer as invalid for AC/DC prediction since this is an INTER MB
*(pCoefBufRow->pYPtr) = -1;
```

```
*(pCoefBufCol->pYPtr) = -1;
*(pCoefBufCol->pYPtr + 8) = -1;
```

6.2.3 Encoder/Decoder Functions

This section defines omxVCM4P2 functions that could be used to construct either an MPEG-4 simple profile encoder or an MPEG-4 simple profile decoder.

6.2.3.1 FindMVpred

Prototype

```
OMXResult omxVCM4P2_FindMVpred(OMXVCMotionVector* pSrcMVCurMB,
   OMXVCMotionVector* pSrcCandMV1, OMXVCMotionVector* pSrcCandMV2,
   OMXVCMotionVector* pSrcCandMV3, OMXVCMotionVector* pDstMVPred,
   OMXVCMotionVector* pDstMVPredME, OMX_INT iBlk);
```

Description

Predicts a motion vector for the current block using the procedure specified in ISO/IEC 14496-2 subclause 7.6.5. The resulting predicted MV is returned in `pDstMVPred`. If the parameter `pDstMVPredME` if is not NULL then the set of three MV candidates used for prediction is also returned, otherwise `pDstMVPredME` is NULL upon return.

Input Arguments

- `pSrcMVCurMB` – pointer to the MV buffer associated with the current Y macroblock; a value of `NULL` indicates inavailablity.
- `pSrcCandMV1` – pointer to the MV buffer containing the 4 MVs associated with the MB located to the left of the current MB; set to NULL if there is no MB to the left.
- `pSrcCandMV2` – pointer to the MV buffer containing the 4 MVs associated with the MB located above the current MB; set to NULL if there is no MB located above the current MB.
- `pSrcCandMV3` – pointer to the MV buffer containing the 4 MVs associated with the MB located to the right and above the current MB; set to NULL if there is no MB located to the above-right.
- `iBlk` – the index of block in the current macroblock
- `pDstMVPredME` – MV candidate return buffer; if set to NULL then prediction candidate MVs are not returned and `pDstMVPredME` will beNULL upon function return; if `pDstMVPredME` is non-NULL then it must point to a buffer containing sufficient space for three return MVs.

Output Arguments

- `pDstMVPred` – pointer to the predicted motion vector
- `pDstMVPredME` – if non-NULL upon input then `pDstMVPredME` points upon return to a buffer containing the three motion vector candidates used for prediction as specified in ISO/IEC 14496-2, subclause 7.6.5, otherwise if NULL upon input then `pDstMVPredME` is NULL upon output.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

Reference

ISO/IEC 14496-2, subclause 7.6.5

6.2.3.2 IDCT8x8blk

Prototype

```
OMXResult omxVCM4P2_IDCT8x8blk(OMX_S16 *pSrc, OMX_S16 *pDst);
```

Description

Computes a 2D inverse DCT for a single 8x8 block, as defined in ISO/IEC 14496-2.

Input Arguments

- `pSrc` – pointer to the start of the linearly arranged IDCT input buffer; must be aligned on a 16-byte boundary.

Output Arguments

- `pDst` – pointer to the start of the linearly arranged IDCT output buffer; must be aligned on a 16-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — Either `pSrc` or `pDst` is NULL.
  - — Either `pSrc` or `pDst` is not 16-byte aligned.

Reference

ISO/IEC 14496-2

6.2.4 Encoder Functions

This section defines the omxVCM4P2 sub-domain functions that could be used to construct an MPEG-4 simple profile encoder, including functions that support motion estimation, 2D discrete cosine transform (DCT), quantization, motion vector encoding, zig-zag scan, transform coefficient encoding, and variable-length coding (VLC). Both high-level and low-level motion estimation functions are defined, and helper functions are defined to initialize the necessary vendor-specific motion estimation specification structures.

6.2.4.1 Motion Estimation Helper Functions

6.2.4.1.1 MEGetBufSize

Prototype

```
OMXResult omxVCM4P2_MEGetBufSize (OMXVCM4P2MEMode MEmode, OMX_U32
    searchRange, OMX_U32* pSize);
```

Description

Computes the size, in bytes, of the vendor-specific specification structure for the following motion estimation functions: `BlockMatch_Integer_8x8`, `BlockMatch_Integer_16x6`, and `MotionEstimateMB`.

Input Arguments

- `MEmode` – motion estimation mode; available modes are defined by the enumerated type `OMXVCM4P2MEMode`
- `searchRange` – motion search range, specified in terms of integer pixel units

Output Arguments

- `pSize` – pointer to the number of bytes required for the specification structure

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – one or more of the following is true:
    - — an invalid value was specified for the parameter `MEmode`
    - — a negative or zero value was specified for the parameter `searchRange`

6.2.4.1.2 MEInit

Prototype

```
OMXResult omxVCM4P2_MEInit (OMXVCM4P2MEMode MEmode, OMX_U32 searchRange,
    void *pMESpec);
```

Description

Initializes the vendor-specific specification structure required for the following motion estimation functions: `BlockMatch_Integer_8x8`, `BlockMatch_Integer_16x16`, and `MotionEstimateMB`. Memory for the specification structure `*pMESpec` must be allocated prior to calling the function, and should be aligned on a 4-byte boundary. The number of bytes required for the specification structure can be determined using the function `omxVCM4P2_MEGetBufSize`.

Input Arguments

- `MEmode` – motion estimation mode; available modes are defined by the enumerated type `OMXVCM4P2MEMode`
- `searchRange` – motion search range, specified in terms of integer pixel units

- pMESpec – pointer to the uninitialized ME specification structure

Output Arguments

- pMESpec – pointer to the initialized ME specification structure

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – one or more of the following is true:
    - an invalid value was specified for the parameter MEmode
    - a negative or zero value was specified for the parameter searchRange

6.2.4.2 Motion Estimation Functions, Low-Level

6.2.4.2.1 BlockMatch_Integer_16x16

Prototype

```
OMXResult omxVCM4P2_BlockMatch_Integer_16x16(OMX_U8 *pSrcRefBuf, OMX_INT
   refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate *
   *pCurrPointPos, OMXVCMotionVector *pSrcPreMV, OMX_INT *pSrcPreSAD,
   OMX_INT searchRange, void *pMESpec, OMXVCMotionVector *pDstMV, OMX_INT
   *pDstSAD);
```

Description

Performs a 16x16 block search; estimates motion vector and associated minimum SAD. Both the input and output motion vectors are represented using half-pixel units, and therefore a shift left or right by 1 bit may be required, respectively, to match the input or output MVs with other functions that either generate output MVs or expect input MVs represented using integer pixel units.

Input Arguments

- pSrcRefBuf – pointer to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- refWidth – width of the reference plane
- pRefRect – pointer to the valid reference plane rectangle; coordinates are specified relative to the image origin. Rectangle boundaries may extend beyond image boundaries if the image has been padded.
- pSrcCurrBuf – pointer to the current block in the current macroblock buffer extracted from the original plane (linear array, 256 entries); must be aligned on a 16-byte boundary. The number of bytes between lines (step) is 16.
- pCurrPointPos – position of the current macroblock in the current plane
- pSrcPreMV – pointer to predicted motion vector; NULL indicates no predicted MV
- pSrcPreSAD – pointer to SAD associated with the predicted MV (referenced by pSrcPreMV)
- searchRange – search range for 16X16 integer block; the search range is the same in all directions. It is in inclusive of the boundary and specified in terms of integer pixel units.

- pMESpec – vendor-specific motion estimation specification structure; must have been allocated and then initialized using omxVCM4P2_MEInit prior to calling the block matching function.

Output Arguments

- pDstMV – pointer to estimated MV
- pDstSAD – pointer to minimum SAD

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments. Returned if one of the following conditions is true:
  - at least one of the following pointers is NULL: pSrcRefBuf, pRefRect, pSrcCurrBuff, pCurrPointPos, pSrcPreSAD, or pMESpec, or
  - pSrcCurrBuf is not 16-byte aligned

6.2.4.2.2 BlockMatch_Integer_8x8

Prototype

```
OMXResult omxVCM4P2_BlockMatch_Integer_8x8(OMX_U8 *pSrcRefBuf, OMX_INT
   refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate *
   *pCurrPointPos, OMXVCMotionVector *pSrcPreMV, OMX_INT *pSrcPreSAD,
   OMX_INT searchRange, void *pMESpec, OMXVCMotionVector *pDstMV, OMX_INT
   *pDstSAD);
```

Description

Performs an 8x8 block search; estimates motion vector and associated minimum SAD. Both the input and output motion vectors are represented using half-pixel units, and therefore a shift left or right by 1 bit may be required, respectively, to match the input or output MVs with other functions that either generate output MVs or expect input MVs represented using integer pixel units.

Input Arguments

- pSrcRefBuf – pointer to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- refWidth – width of the reference plane
- pRefRect – pointer to the valid reference plane rectangle; coordinates are specified relative to the image origin. Rectangle boundaries may extend beyond image boundaries if the image has been padded.
- pSrcCurrBuf – pointer to the current block in the current macroblock buffer extracted from the original plane (linear array, 128 entries); must be aligned on a 16-byte boundary. The number of bytes between lines (step) is 16.
- pCurrPointPos – position of the current macroblock in the current plane
- pSrcPreMV – pointer to predicted motion vector; NULL indicates no predicted MV
- pSrcPreSAD – pointer to SAD associated with the predicted MV (referenced by pSrcPreMV)
- searchRange – search range for 8x8 integer block; the search range is the same in all directions. It is in inclusive of the boundary and specified in terms of integer pixel units.

- `pMESpec` – vendor-specific motion estimation specification structure; must have been allocated and then initialized using `omxVCM4P2_MEInit` prior to calling the block matching function.

Output Arguments

- `pDstMV` – pointer to estimated MV
- `pDstSAD` – pointer to minimum SAD

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments. Returned if one of the following conditions is true:
  - at least one of the following pointers is NULL: `pSrcRefBuf`, `pRefRect`, `pSrcCurrBuff`, `pCurrPointPos`, `pSrcPreSAD`, or `pMESpec`, or
  - `pSrcCurrBuf` is not 16-byte aligned

6.2.4.2.3 BlockMatch_Half_16x16

Prototype

```
OMXResult omxVCM4P2_BlockMatch_Half_16x16(OMX_U8 *pSrcRefBuf, OMX_INT
   refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate
   *pSearchPointRefPos, OMX_INT rndVal, OMXVCMotionVector *pSrcDstMV,
   OMX_INT *pDstSAD);
```

Description

Performs a 16x16 block match with half-pixel resolution. Returns the estimated motion vector and associated minimum SAD. This function estimates the half-pixel motion vector by interpolating the integer resolution motion vector referenced by the input parameter `pSrcDstMV`, i.e., the initial integer MV is generated externally. The input parameters `pSrcRefBuf` and `pSearchPointRefPos` should be shifted by the winning MV of 16x16 integer search prior tocalling `BlockMatch_Half_16x16`. The function `BlockMatch_Integer_16x16` may be used for integer motion estimation.

Input Arguments

- `pSrcRefBuf` – pointer to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- `RefWidth` – width of the reference plane
- `pRefRect` – reference plane valid region rectangle
- `pSrcCurrBuf` – pointer to the current block in the current macroblock buffer extracted from the original plane (linear array, 256 entries); must be aligned on a 16-byte boundary. The number of bytes between lines (step) is 16.
- `pSearchPointRefPos` – position of the starting point for half pixel search (specified in terms of integer pixel units) in the reference plane.
- `rndVal` – rounding control bit for half pixel motion estimation; 0=rounding control disabled; 1=rounding control enabled
- `pSrcDstMV` – pointer to the initial MV estimate; typically generated during a prior 16X16 integer search; specified in terms of half-pixel units.

Output Arguments

- `pDstMV` – pointer to estimated MV
- `pDstSAD` – pointer to minimum SAD

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments. Returned if one of the following conditions is true:
  - — at least one of the following pointers is NULL: `pSrcRefBuf`, `pRefRect`, `pSrcCurrBuff`, `pSearchPointRefPos`, `pSrcDstMV`, or
  - — `pSrcCurrBuf` is not 16-byte aligned, or
  - — `rndVal` has value other than 0 or 1.

6.2.4.2.4 BlockMatch_Half_8x8

Prototype

```
OMXResult omxVCM4P2_BlockMatch_Half_8x8(OMX_U8 *pSrcRefBuf, OMX_INT
    refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate
    *pSearchPointRefPos, OMX_INT rndVal, OMXVCMotionVector *pSrcDstMV,
    OMX_INT *pDstSAD);
```

Description

Performs an 8x8 block match with half-pixel resolution. Returns the estimated motion vector and associated minimum SAD. This function estimates the half-pixel motion vector by interpolating the integer resolution motion vector referenced by the input parameter `pSrcDstMV`, i.e., the initial integer MV is generated externally. The input parameters `pSrcRefBuf` and `pSearchPointRefPos` should be shifted by the winning MV of 8x8 integer search prior tocalling `BlockMatch_Half_8x8`. The function `BlockMatch_Integer_8x8` may be used for integer motion estimation.

Input Arguments

- `pSrcRefBuf` – `pointer` to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- `refWidth` – width of the reference plane
- `pRefRect` – reference plane valid region rectangle
- `pSrcCurrBuf` – pointer to the current block in the current macroblock buffer extracted from the original plane (linear array, 128 entries); must be aligned on a 16-byte boundary. The number of bytes between lines (step) is 16.
- `pSearchPointRefPos` – position of the starting point for half pixel search (specified in terms of integer pixel units) in the reference plane.
- `rndVal` – rounding control bit for half pixel motion estimation; 0=disable rounding control, 1=enable rounding control.
- `pSrcDstMV` – pointer to the initial MV estimate; typically generated during a prior 8x8 integer search, specified in terms of half-pixel units.

Output Arguments

- `pSrcDstMV` – pointer to estimated MV

- pDstSAD – pointer to minimum SAD

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments

6.2.4.3 MotionEstimation Functions, High-Level

6.2.4.2.1 MotionEstimationMB

Prototype

```
OMXResult omxVCM4P2_MotionEstimationMB (OMX_U8 *pSrcRefBuf, OMX_INT
   RefWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate
   *pCurrPointPos, OMXVCMotionVector *pSrcPreMV, OMX_INT *pSrcPreSAD,
   OMX_INT rndVal, OMX_INT searchRange, void *pMESpec,
   OMXVCM4P2MacroblockType *pDstMBType, OMXVCMotionVector *pDstMV, OMX_INT
   *pDstSAD);
```

Description

Performs motion search for a 16x16 macroblock. Selects best motion search strategy from among inter-1MV, inter-4MV, and intra modes. Supports integer and half pixel resolution.

Input Arguments

- pSrcRefBuf – pointer to the reference Y plane; points to the reference plane location corresponding to the location of the current macroblock in the current plane.
- refWidth – width of the reference plane
- pRefRect – reference plane valid region rectangle
- pSrcCurrBuf – pointer to the current block in the current macroblock buffer extracted from the original plane (linear array, 256 entries); must be aligned on a 16-byte boundary. The number of bytes between lines (step) is 16.
- pCurrPointPos – position of the current macroblock in the current plane
- pSrcPreMV – pointer to predicted motion vector; NULL indicates no predicted MV
- pSrcPreSAD – pointer to SAD associated with the predicted MV (referenced by pSrcPreMV)
- rndVal – rounding control bit for half pixel motion estimation
- searchRange – search range for 16X16 integer block match
- pMESpec – vendor-specific motion estimation specification structure; must have been allocated and then initialized using omxVCM4P2_MEInit prior to calling the block matching function.

Output Arguments

- pDstMBType – indicates the motion search strategy selected during the ME process. Takes one of the following values: OMX_VC_INTER, OMX_VC_INTER4V, or OMX_VC_INTRA.
- pDstMV – pointer to the destination motion vectors; returns one of the following: 4 MVs for Inter4v, 1 MV for Inter, or the 0-vector (0, 0) for Intra.
- pDstSAD – pointer to the minimum SAD

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

6.2.4.4 DCT and Quantization Functions

6.2.4.4.1 DCT8x8blk

Prototype

```
OMXResult omxVCM4P2_DCT8x8blk(OMX_S16 *pSrc, OMX_S16 *pDst);
```

Description

Computes a 2D forward DCT for a single 8x8 block, as defined in ISO/IEC 14496-2.

Input Arguments

- `pSrc` – pointer to the start of the linearly arranged DCT input buffer; must be aligned on a 16-byte boundary.

Output Arguments

- `pDst` – pointer to the start of the linearly arranged DCT output buffer; must be aligned on a 16-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — Either `pSrc` or `pDst` is NULL.
  - — Either `pSrc` or `pDst` is not 16-byte aligned.

Reference

ISO/IEC 14496-2

6.2.4.4.2 QuantIntra_I

Prototype

```
OMXResult omxVCM4P2_QuantIntra_I(OMX_S16 * pSrcDst, OMX_U8 QP, OMX_INT
   blockIndex, const OMXVCM4P2QuantWeightCoef *pQMatrix, OMX_INT
   shortVideoHeader);
```

Description

Performs quantization on intra block coefficients. This function supports bits_per_pixel == 8.

Input Arguments

- `pSrcDst` – pointer to the input intra block coefficients; must be aligned on a 16-byte boundary.
- `QP` – quantization parameter (quantizer_scale).
- `blockIndex` – block index indicating the component type and position as defined in subclause 6.1.3.8, of *ISO/IEC 14496-2*. Indices 6 to 9 indicate the alpha blocks spatially corresponding to luminance blocks 0 to 3 in the same macroblock.
- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9; must be aligned on a 16-byte boundary.
- `shortVideoHeader` – binary flag indicating presence of short_video_header; `shortVideoHeader==1` selects linear intra DC mode, and `shortVideoHeader==0` selects non-linear intra DC mode.

Output Arguments

- `pSrcDst` – pointer to the output (quantized) interblock coefficients. When `shortVideoHeader==1`, AC coefficients are saturated on the interval [-127, 127], and DC coefficients are saturated on the interval [1, 254]. When `shortVideoHeader==0`, AC coefficients are saturated on the interval [-2047, 2047].

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — `pSrcDst` is NULL.
  - — blockIndex < 0 or blockIndex >= 10
  - — QP <= 0 or QP >= 32.
  - — for a non-NULL `pQMatrix, pQMatrix`[0] * `pQMatrix`[64] != (1<<21)

6.2.4.4.3 QuantInter_I

Prototype

```
OMXResult omxVCM4P2_QuantInter_I(OMX_S16 * pSrcDst, OMX_U8 QP, const
   OMXVCM4P2QuantWeightCoef *pQMatrix, OMX_INT shortVideoHeader);
```

Description

Performs quantization on an inter coefficient block; supports bits_per_pixel == 8.

Input Arguments

- `pSrcDst` – pointer to the input inter block coefficients; must be aligned on a 16-byte boundary.
- `QP` – quantization parameter (`quantizer_scale`)

- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9; must be aligned on a 16-byte boundary.

Output Arguments

- `pSrcDst` – pointer to the output (quantized) interblock coefficients. When `shortVideoHeader==1`, AC coefficients are saturated on the interval [-127, 127], and DC coefficients are saturated on the interval [1, 254]. When `shortVideoHeader==0`, AC coefficients are saturated on the interval [-2047, 2047].

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - `pSrcDst` is NULL.
  - `QP` <= 0 or `QP` >= 32.
  - `pQMatrix[0]*` `pQMatrix`[64] != (1<<21) while `pQMatrix` is not NULL

6.2.4.4.4 TransRecBlockCoef_intra

Prototype

```
OMXResult omxVCM4P2_TransRecBlockCoef_intra(OMX_U8 *pSrc, OMX_S16 * pDst,
    OMX_U8 * pRec, OMX_S16 *pPredBufRow, OMX_S16 *pPredBufCol, OMX_S16 *
    pPreACPredict, OMX_INT *pSumErr, OMX_INT blockIndex, OMX_U8 curQp,
    OMX_U8 *pQpBuf, OMX_INT srcStep, OMX_INT dstStep, const
    OMXVCM4P2QuantWeightCoef *pQMatrix, OMX_INT shortVideoHeader);
```

Description

Quantizes the DCT coefficients, implements intra block AC/DC coefficient prediction, and reconstructs the current intra block texture for prediction on the next frame. Quantized row and column coefficients are returned in the updated coefficient buffers.

Input Arguments

- `pSrc` – pointer to the pixels of current intra block; must be aligned on an 8-byte boundary.
- `pPredBufRow` – pointer to the coefficient row buffer containing ((num_mb_per_row * 2 + 1) * 8) elements of type `OMX_S16`. Coefficients are organized into blocks of eight as described below (Internal Prediction Coefficient Update Procedures). The DC coefficient is first, and the remaining buffer locations contain the quantized AC coefficients. Each group of eight row buffer elements combined with one element eight elements ahead contains the coefficient predictors of the neighboring block that is spatially above or to the left of the block currently to be decoded. A negative-valued DC coefficient indicates that this neighboring block is not INTRA-coded or out of bounds, and therefore the AC and DC coefficients are invalid. Pointer must be aligned on an 8-byte boundary.
- `pPredBufCol` – pointer to the prediction coefficient column buffer containing 16 elements of type `OMX_S16`. Coefficients are organized as described in section 6.2.2.5. Pointer must be aligned on an 8-byte boundary.

- `pSumErr` – pointer to a flag indicating whether or not AC prediction is required; AC prediction is enabled if `*pSumErr >=0`, but the value is not used for coefficient prediction, i.e., the sum of absolute differences starts from 0 for each call to this function. Otherwise AC prediction is disabled if `*pSumErr < 0`.
- `blockIndex` – block index indicating the component type and position as defined in subclause 6.1.3.8, of *ISO/IEC 14496-2*. Indexes 6 to 9 indicate the alpha blocks spatially corresponding to luminance blocks 0 to 3 in the same macroblock.
- `curQp` – quantization parameter of the macroblock to which the current block belongs
- `pQpBuf` – pointer to a 2-element quantization parameter buffer; pQpBuf[0] contains the quantization parameter associated with the 8x8 block left of the current block (QPa), and pQpBuf[1] contains the quantization parameter associated with the 8x8 block above the current block (QPc). In the event that the corresponding block is outside of the VOP bound, the Qp value will not effect the intra prediction process, as described in sub-clause 7.4.3.3 of *ISO/IEC 14496-2*, "Adaptive AC Coefficient Prediction."
- `srcStep` – width of the source buffer; must be a multiple of 8.
- `dstStep` – width of the reconstructed destination buffer; must be a multiple of 8.
- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9.
- `shortVideoHeader` – binary flag indicating presence of short_video_header; `shortVideoHeader==1` selects linear intra DC mode, and `shortVideoHeader==0` selects non-linear intra DC mode.

Output Arguments

- `pDst` – pointer to the quantized DCT coefficient buffer; pDst[0] contains the predicted DC coefficient. The pDst pointer must be aligned on a 16-byte boundary.
- `pRec` – pointer to the reconstructed texture; must be aligned on an 8-byte boundary.
- `pPredBufRow` – pointer to the updated coefficient row buffer
- `pPredBufCol` – pointer to the updated coefficient column buffer
- `pPreACPredict` – if prediction is enabled, the parameter points to the start of the buffer containing the coefficient differences for VLC encoding. The first element indicates prediction direction for the current block and takes one of the following values: OMX_VC_NONE (prediction disabled), OMX_VC_HORIZONTAL, or OMX_VC_VERTICAL. If prediction is disabled (*pSumErr<0) then the contents of this buffer are undefined upon return from the function
- `pSumErr` – pointer to the value of the accumulated AC coefficient errors, i.e., sum of the absolute differences between predicted and unpredicted AC coefficients

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments
  - At least one of the following pointers is NULL: `pSrc`, `pDst`, `pRec`, `pCoefBufRow`, `pCoefBufCol`, `pQpBuf`, `pPreACPredict`, `pSumErr`.
  - `BlockIndex` < 0 or `blockIndex` >= 10; `curQP` <= 0 or `curQP` >= 32.
  - `SrcStep`, `dstStep` <= 0 or not a multiple of 8.

— At least one of the following pointers is not 64-bit aligned: `pSrc`, `pDst`, `pRec`.

— `pQMatrix[0] * pQMatrix[64]` != (1<<21) while `pQMatrix` is not NULL.

---

2 **Note:** *The coefficient buffers must be updated in accordance with the update procedures defined in section in 6.2.2.*

6.2.4.4.5 TransRecBlockCoef_inter

Prototype

```
OMXResult omxVCM4P2_TransRecBlockCoef_inter(OMX_S16 *pSrc, OMX_S16 * pDst,
    OMX_S16 * pRec, OMX_U8 QP, const OMXVCM4P2QuantWeightCoef *pQMatrix);
```

Description

Implements DCT, and quantizes the DCT coefficients of the inter block while reconstructing the texture residual. There is no boundary check for the bit stream buffer.

Input Arguments

- `pSrc` – For the current InterBlock, points to the residuals to be encoded.
- `QP` – quantization parameter.
- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9.

Output Arguments

- `pDst` – pointer to the quantized DCT coefficients buffer
- `pRec` – pointer to the reconstructed texture residuals

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

  — At least one of the following pointers is NULL or is not 64-bit aligned: `pSrc`, `pDst`, `pRec`.

  — `QP` <= 0 or `QP` >= 32.

  — `pQMatrix[0] * pQMatrix[64]` != (1<<21) while `pQMatrix` is not NULL.

6.2.4.5 Motion Vector Encoding and VLC Functions

6.2.4.5.1 EncodeVLCZigzag_IntraDCVLC

6.2.4.5.2 EncodeVLCZigzag_IntraACVLC

Prototype

```
OMXResult omxVCM4P2_EncodeVLCZigzag_IntraDCVLC(OMX_U8 **ppBitStream, OMX_INT
    *pBitOffset, OMX_S16 *pQDctBlkCoef, OMX_U8 predDir, OMX_U8 pattern,
    OMXVCM4P2VideoComponent videoComp);
```

```
OMXResult omxVCM4P2_EncodeVLCZigzag_IntraACVLC(OMX_U8 **ppBitStream, OMX_INT
    *pBitOffset, OMX_S16 *pQDctBlkCoef, OMX_U8 predDir, OMX_U8 pattern);
```

Description

Performs zigzag scan and VLC encoding of AC and DC coefficients for one intra block. Two versions of the function (DCVLC and ACVLC) are provided in order to support the two different methods of processing DC coefficients, as described in ISO/IEC 14496-2, subclause 7.4.1.4, "Intra DC Coefficient Decoding for the Case of Switched VLC Encoding."

Input Arguments

- `ppBitStream` – double pointer to the current byte in the bitstream
- `pBitOffset` – pointer to the bit position in the byte pointed by *ppBitStream. Valid within 0 to 7.
- `pQDctBlkCoef` – pointer to the quantized DCT coefficient
- `predDir` – AC prediction direction, which is used to decide the zigzag scan pattern; takes one of the following values:
  - `OMX_VC_NONE` – AC prediction not used. Performs classical zigzag scan.
  - `OMX_VC_HORIZONTAL` – Horizontal prediction. Performs alternate-vertical zigzag scan.
  - `OMX_VC_VERTICAL` – Vertical prediction. Performs alternate-horizontal zigzag scan.
- `pattern` – block pattern which is used to decide whether this block is encoded
- `videoComp` – video component type (luminance, chrominance) of the current block

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is encoded, so that it points to the current byte in the bit stream buffer.
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by `*ppBitStream`.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments
  - At least one of the following pointers is NULL: `ppBitStream`, `*ppBitStream`, `pBitOffset`, `pQDctBlkCoef`.
  - *pBitOffset < 0, or *pBitOffset >7.
  - PredDir is not one of: `OMX_VC_NONE, OMX_VC_HORIZONTAL, or OMX_VC_VERTICAL.`

— `VideoComp` is not one component of enum `OMXVCM4P2VideoComponent`.

Reference

ISO/IEC 14496-2, subclause 7.4.1.4

6.2.4.5.3 EncodeVLCZigzag_Inter

Prototype

```
OMXResult omxVCM4P2_EncodeVLCZigzag_Inter(OMX_U8 **ppBitStream, OMX_INT *
    pBitOffset, OMX_S16 *pQDctBlkCoef, OMX_U8 pattern);
```

Description

Performs classical zigzag scanning and VLC encoding for one inter block.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream
- `pBitOffset` – pointer to the bit position in the byte pointed by `*ppBitStream`. Valid within 0 to 7
- `pQDctBlkCoef` – pointer to the quantized DCT coefficient
- `pattern` – block pattern which is used to decide whether this block is encoded

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is encoded so that it points to the current byte in the bit stream buffer.
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by `*ppBitStream`.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – Bad arguments
    - — At least one of the pointers: is NULL: `ppBitStream`, `*ppBitStream`, `pBitOffset`, `pQDctBlkCoef`
    - — `*pBitOffset` < 0, or `*pBitOffset` >7.

6.2.4.5.4 EncodeMV

Prototype

```
OMXResult omxVCM4P2_EncodeMV(OMX_U8 **ppBitStream, OMX_INT *pBitOffset,
    OMXVCMotionVector * pMVCurMB, OMXVCMotionVector * pSrcMVLeftMB,
    OMXVCMotionVector * pSrcMVUpperMB, OMXVCMotionVector *
    pSrcMVUpperRightMB, OMX_INT fcodeForward, OMXVCM4P2MacroblockType
    MBType);
```

Description

Predicts a motion vector for the current macroblock, encodes the difference, and writes the output to the stream buffer. The input MVs pMVCurMB, pSrcMVLeftMB, pSrcMVUpperMB, and pSrcMVUpperRightMB should lie within the ranges associated with the input parameter fcodeForward, as described in ISO/IEC 14496-2, subclause 7.6.3. This function provides a superset of the functionality associated with the function omxVCM4P2_FindMVpred.

Input Arguments

- ppBitStream – double pointer to the current byte in the bitstream buffer
- pBitOffset – index of the first free (next available) bit in the stream buffer referenced by *ppBitStream, valid in the range 0 to 7.
- pMVCurMB – pointer to the current macroblock motion vector; a value of NULL indicates inavailablity.
- pSrcMVLeftMB – pointer to the source left macroblock motion vector; a value of NULL indicates inavailablity.
- pSrcMVUpperMB – pointer to source upper macroblock motion vector; a value of NULL indicates inavailablity.
- pSrcMVUpperRightMB – pointer to source upper right MB motion vector; a value of NULL indicates inavailablity.
- fcodeForward – an integer with values from 1 to 7; used in encoding motion vectors related to search range, as described in ISO/IEC 14496-2, subclause 7.6.3.
- MBType – macro block type, valid in the range 0 to 9

Output Arguments

- ppBitStream – updated pointer to the current byte in the bit stream buffer
- pBitOffset – updated index of the next available bit position in stream buffer referenced by *ppBitStream

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments
  - At least one of the following pointers is NULL: ppBitStream, *ppBitStream, pBitOffset, pMVCurMB
  - *pBitOffset < 0, or *pBitOffset >7.
  - fcodeForward <= 0, or fcodeForward > 7, or MBType < 0.

Reference

ISO/IEC 14496-2, subclause 7.6.3

6.2.5 Decoder Functions

This section describes omxVCM4P2 functions that can be used to construct an MPEG-4 simple profile decoder.

6.2.5.1 Motion Vector Decoding Functions

6.2.5.1.1 DecodePadMV_PVOP

Prototype

```
OMXResult omxVCM4P2_DecodePadMV_PVOP(const OMX_U8 ** ppBitStream, OMX_INT *
    pBitOffset, OMXVCMotionVector * pSrcMVLeftMB, OMXVCMotionVector
    *pSrcMVUpperMB, OMXVCMotionVector * pSrcMVUpperRightMB,
    OMXVCMotionVector * pDstMVCurMB, OMX_INT fcodeForward,
    OMXVCM4P2MacroblockType MBType);
```

Description

Decodes and pads the four motion vectors associated with a non-intra P-VOP macroblock. For macroblocks of type `OMX_VC_INTER4V`, the output MV is padded as specified in subclause 7.6.1.6 of *ISO/IEC 14496-2*. Otherwise, for macroblocks of types other than `OMX_VC_INTER4V`, the decoded MV is copied to all four output MV buffer entries.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pSrcMVLeftMB`, `pSrcMVUpperMB`, and `pSrcMVUpperRightMB` – pointers to the motion vector buffers of the macroblocks specially at the left, upper, and upper-right side of the current macroblock, respectively; a value of `NULL` indicates inavailablity.

---

2 ***Note:*** *Any neighborhood macroblock outside the current VOP or video packet or outside the current GOB (when short_video_header is "1") for which gob_header_empty is "0" is treated as transparent, according to subclause 7.6.5 in ISO/IEC 14496-2.*

- `fcodeForward` – a code equal to `vop_fcode_forward` in MPEG-4 bit stream syntax
- `MBType` – the type of the current macroblock. If `MBType` is not equal to `OMX_VC_INTER4V`, the destination motion vector buffer is still filled with the same decoded vector.

Output Arguments

- `ppBitStream` – *ppBitStream is updated after the block is decoded, so that it points to the current byte in the bit stream buffer
- `pBitOffset` – *pBitOffset is updated so that it points to the current bit position in the byte pointed by *ppBitStream
- `pDstMVCurMB` – pointer to the motion vector buffer for the current macroblock; contains four decoded motion vectors

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

— At least one of the following pointers is NULL: `ppBitStream`, `*ppBitStream`, `pBitOffset`, `pDstMVCurMB`
or

— At least one of following cases is true: `*pBitOffset` exceeds [0,7], `fcodeForward` exceeds (0,7], `MBType` less than zero, motion vector buffer is not 32-bit aligned.

- `OMX_StsErr` – status error

6.2.5.2 VLC Decoding/Inverse Zig-Zag Scan Functions

6.2.5.2.1 DecodeVLCZigzag_IntraDCVLC

6.2.5.2.2 DecodeVLCZigzag_IntraACVLC

Prototype

```
OMXResult omxVCM4P2_DecodeVLCZigzag_IntraDCVLC(const OMX_U8 ** ppBitStream,
   OMX_INT * pBitOffset, OMX_S16 * pDst, OMX_U8 predDir,
   OMXVCM4P2VideoComponent videoComp);
OMXResult omxVCM4P2_DecodeVLCZigzag_IntraACVLC(const OMX_U8 ** ppBitStream,
   OMX_INT * pBitOffset, OMX_S16 * pDst, OMX_U8 predDir);
```

Description

Performs VLC decoding and inverse zigzag scan of AC and DC coefficients for one intra block. Two versions of the function (DCVLC and ACVLC) are provided in order to support the two different methods of processing DC coefficients, as described in ISO/IEC 14496-2, subclause 7.4.1.4, “Intra DC Coefficient Decoding for the Case of Switched VLC Encoding.”

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bitstream buffer
- `pBitOffset` – pointer to the bit position in the current byte referenced by *ppBitStream. The parameter *pBitOffset is valid in the range [0-7].

Bit Position in one byte:	|Most						Least|	
*pBitOffset	|0	1	2	3	4	5	6	7|

- `predDir` – AC prediction direction; used to select the zigzag scan pattern; takes one of the following values:

  `OMX_VC_NONE` – AC prediction not used; performs classical zigzag scan.

  `OMX_VC_HORIZONTAL` – Horizontal prediction; performs alternate-vertical zigzag scan;

  `OMX_VC_VERTICAL` – Vertical prediction; performs alternate-horizontal zigzag scan.

- `videoComp` – video component type (luminance, chrominance or alpha) of the current block

Output Arguments

- `ppBitStream` – *ppBitStream is updated after the block is decoded such that it points to the current byte in the bit stream buffer
- `pBitOffset` – *pBitOffset is updated such that it points to the current bit position in the byte pointed by *ppBitStream

- pDst – pointer to the coefficient buffer of current block; must be 32-bit aligned.

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments
  - At least one of the following pointers is NULL: ppBitStream, *ppBitStream, pBitOffset, pDst, or
  - At least one of the following conditions is true: *pBitOffset exceeds [0,7], preDir exceeds [0,2], or
  - pDst is not 32-bit aligned
- OMX_StsErr
  - In DecodeVLCZigzag_IntraDCVLC_S16, dc_size > 12
  - At least one of mark bits equals zero
  - Illegal stream encountered; code cannot be located in VLC table
  - Forbidden code encountered in the VLC FLC table
  - The number of coefficients is greater than 64

Reference

ISO/IEC 14496-2, subclause 7.4.1.4

6.2.5.2.3 DecodeVLCZigzag_Inter_S16

Prototype

```
OMXResult omxVCM4P2_DecodeVLCZigzag_Inter(const OMX_U8 ** ppBitStream,
    OMX_INT * pBitOffset, OMX_S16 * pDst);
```

Description

Performs VLC decoding and inverse zigzag scan for one inter-coded block.

Input Arguments

- ppBitStream – double pointer to the current byte in the stream buffer
- pBitOffset – pointer to the next available bit in the current stream byte referenced by *ppBitStream. The parameter *pBitOffset is valid within the range [0-7].

Output Arguments

- ppBitStream – *ppBitStream is updated after the block is decoded such that it points to the current byte in the stream buffer
- pBitOffset – *pBitOffset is updated after decoding such that it points to the next available bit in the stream byte referenced by *ppBitStream
- pDst – pointer to the coefficient buffer of current block; must be 32-bit aligned.

Returns

- OMX_StsBadArgErr – bad arguments

— At least one of the following pointers is NULL: ppBitStream, `*ppBitStream`, `pBitOffset`, `pDst`, or

— `pDst` is not 32-bit aligned, or

— `*pBitOffset` exceeds [0,7].

- `OMX_StsErr` – status error

— At least one mark bit is equal to zero

— Encountered an illegal stream code that cannot be found in the VLC table

— Encountered and illegal code in the VLC FLC table

— The number of coefficients is greater than 64

6.2.5.3 Inverse Quantization Functions

6.2.5.3.1 QuantInvIntra_S16_I

6.2.5.3.2 QuantInvInter_S16_I

Prototype

```
OMXResult omxVCM4P2_QuantInvIntra_I(OMX_S16 * pSrcDst, OMX_INT QP, const
   OMXVCM4P2QuantWeightCoef *pQMatrix, OMXVCM4P2VideoComponent videoComp,
   OMX_INT shortVideoHeader);
OMXResult omxVCM4P2_QuantInvInter_I(OMX_S16 * pSrcDst, OMX_INT QP, const
   OMXVCM4P2QuantWeightCoef *pQMatrix, OMX_INT shortVideoHeader);
```

Description

Performs inverse quantization on intra/inter coded block. This function supports `bits_per_pixel` = 8. Mismatch control is performed for the first MPEG-4 mode inverse quantization method.

- The output coefficients are clipped to the range: [-2048, 2047].
- Mismatch control is performed for the MPEG-4 quantization method.

Input Arguments

- `pSrcDst` – pointer to the input (quantized) intra/inter block; must be aligned on a 16-byte boundary.
- `QP` – quantization parameter (quantiser_scale)
- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9; must be aligned on a 16-byte boundary.
- `videoComp` – (Intra version only.) Video component type of the current block. Takes one of the following flags: `OMX_VC_LUMINANCE, OMX_VC_CHROMINANCE, OMX_VC_ALPHA`.
- `shortVideoHeader` – binary flag indicating presence of short_video_header.

Output Arguments

- `pSrcDst` – pointer to the output (dequantized) intra/inter block

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - If `pSrcDst` is NULL.
    or
  - If `QP` <= 0.
    or
  - videoComp is none of `OMX_VC_LUMINANCE, OMX_VC_CHROMINANCE and OMX_VC_ALPHA`.

6.2.5.4 Inverse Quantization/Zig-Zag Scan/DCT Functions

6.2.5.4.1 DecodeBlockCoef_Intra

Prototype

```
OMXResult omxVCM4P2_DecodeBlockCoef_Intra(const OMX_U8 ** ppBitStream,
   OMX_INT *pBitOffset, OMX_U8 *pDst, OMX_INT step, OMX_S16 *pCoefBufRow,
   OMX_S16 *pCoefBufCol, OMX_U8 curQP, OMX_U8 *pQPBuf, const
   OMXVCM4P2QuantWeightCoef *pQMatrix, OMX_INT blockIndex, OMX_INT
   intraDCVLC, OMX_INT ACPredFlag, OMX_INT shortVideoHeader);
```

Description

Decodes the INTRA block coefficients. Inverse quantization, inversely zigzag positioning, and IDCT, with appropriate clipping on each step, are performed on the coefficients. The results are then placed in the output frame/plane on a pixel basis.

---

2 ***Note:*** *This function will be used only when at least one non-zero AC coefficient of current block exists in the bit stream. DC only condition will be handled in another function.*

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer. There is no boundary check for the bit stream buffer.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `step` – width of the destination plane
- `pCoefBufRow` – pointer to the coefficient row buffer; must be aligned on an 8-byte boundary.
- `pCoefBufCol` – pointer to the coefficient column buffer; must be aligned on an 8-byte boundary.
- `curQP` – quantization parameter of the macroblock which the current block belongs to
- `pQPBuf` – pointer to the quantization parameter buffer

- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9; must be aligned on a 16-byte boundary.
- `blockIndex` – block index indicating the component type and position as defined in subclause 6.1.3.8, Figure 6-5 of *ISO/IEC 14496-2*. Furthermore, index 6 to 9 indicate the alpha blocks spatially corresponding to luminance block 0 to 3 in the same macroblock.
- `intraDCVLC` – a code determined by intra_dc_vlc_thr and QP. This allows a mechanism to switch between two VLC for coding of Intra DC coefficients as per Table 6-21 of *ISO/IEC 14496-2*. If the current block is a alpha block, the parameter "intraDCVLC" should not be zero.
- `ACPredFlag` – a flag equal to ac_pred_flag (of luminance) or ac_pred_flag_alpha (of alpha block) indicating if the ac coefficients of the first row or first column are differentially coded for intra coded macroblock.
- `shortVideoHeader` – binary flag indicating presence of short_video_header; `shortVideoHeader==1` selects linear intra DC mode, and `shortVideoHeader==0` selects non-linear intra DC mode.

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is decoded, so that it points to the current byte in the bit stream buffer
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by `*ppBitStream`
- `pDst` – pointer to the block in the destination plane; must be aligned on a 16-byte boundary.
- `pCoefBufRow` – pointer to the updated coefficient row buffer.
- `pCoefBufCol` – pointer to the updated coefficient column buffer

---

2 ***Note:*** *The coefficient buffers must be updated in accordance with the update procedure defined in section 6.2.2.*

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers is NULL: `ppBitStream`, `*ppBitStream`, `pBitOffset`, `pCoefBufRow`, `pCoefBufCol`, `pQPBuf`, `pDst`.
    or
  - — At least one of the below case: `*pBitOffset` exceeds [0,7], curQP exceeds (1, 31), `blockIndex` exceeds [0,9], step is not the multiple of 8, `intraDCVLC` is zero while `blockIndex` greater than 5.
    or
  - — At least one of the pointer alignment requirements was violated.
- `OMX_StsErr` – status error

Refer to “DecodeVLCZigzag_Intra_S16”.

6.2.5.4.2 DecodeBlockCoef_Inter

Prototype

```
OMXResult omxVCM4P2_DecodeBlockCoef_Inter(const OMX_U8 ** ppBitStream,
    OMX_INT * pBitOffset, OMX_S16 * pDst, OMX_INT QP, const
    OMXVCM4P2QuantWeightCoef * pQMatrix);
```

Description

Decodes the INTER block coefficients. This function performs inverse quantization, inverse zigzag positioning, and IDCT (with appropriate clipping on each step) on the coefficients. The results (residuals) are placed in a contiguous array of 64 elements.

For INTER block, the output buffer holds the residuals for further reconstruction.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer. There is no boundary check for the bit stream buffer.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7]
- `QP` – quantization parameter
- `pQMatrix` – pointer to the quantization coefficient weighting matrix. To select the H.263 quantization method, the parameter pQMatrix must be set ==NULL. To select the MPEG-4 quantization method, the parameter pQMatrix must point to a structure containing quantization weighting coefficients and reciprocols, as described in section 6.2.1.9; must be aligned on a 16-byte boundary.

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is decoded, so that it points to the current byte in the bit stream buffer
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by `*ppBitStream`
- `pDst` – pointer to the decoded residual buffer (a contiguous array of 64 elements of `OMX_S16` data type); must be aligned on a 16-byte boundary.

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
    — At least one of the following pointers is Null: `ppBitStream`, `*ppBitStream`, `pBitOffset` , `pDst`
    — At least one of the below case:
        —`*pBitOffset` exceeds [0,7], `QP` <= 0;
        —pDst not 64-bit aligned
- `OMX_StsErr` – status error.

Refer to `OMX_StsErr` of "DecodeVLCZigzag_Inter_S16".

6.2.5.4.2 PredictReconCoefIntra

Prototype

```
OMXResult omxVCM4P2_PredictReconCoefIntra(OMX_S16 * pSrcDst, OMX_S16 *
    pPredBufRow, OMX_S16 * pPredBufCol, OMX_INT curQP, OMX_INT predQP,
    OMX_INT predDir, OMX_INT ACPredFlag, OMXVCM4P2VideoComponent videoComp);
```

Description

Performs adaptive DC/AC coefficient prediction for an intra block. Prior to the function call, prediction direction (`predDir`) should be selected as specified in subclause 7.4.3.1 of *ISO/IEC 14496-2*.

Input Arguments

- `pSrcDst` – pointer to the coefficient buffer which contains the quantized coefficient residuals (PQF) of the current block
- `pPredBufRow` – pointer to the coefficient row buffer
- `pPredBufCol` – pointer to the coefficient column buffer
- `curQP` – quantization parameter of the current block. `curQP` may equal to `predQP` especially when the current block and the predictor block are in the same macroblock.
- `predQP` – quantization parameter of the predictor block
- `predDir` – indicates the prediction direction which takes one of the following values:
    - `OMX_VC_HORIZONTAL` – predict horizontally
    - `OMX_VC_VERTICAL` – predict vertically
- `ACPredFlag` – a flag indicating if AC prediction should be performed. It is equal to `ac_pred_flag` in the bit stream syntax of MPEG-4
- `videoComp` – video component type (luminance, chrominance or alpha) of the current block

Output Arguments

- `pSrcDst` – pointer to the coefficient buffer which contains the quantized coefficients (QF) of the current block
- `pPredBufRow` – pointer to the updated coefficient row buffer
- `pPredBufCol` – pointer to the updated coefficient column buffer

2 ***Note:*** *Buffer update: Update the AC prediction buffer (both row and column buffer).*

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
    - At least one of the pointers is NULL: `pSrcDst`, `pPredBufRow`, or `pPredBufCol`. or

At least one the following cases: `curQP` <= 0, `predQP` <= 0, `preDir` exceeds [1,2].
or

— At least one of the pointers `pSrcDst`, `pPredBufRow`, or `pPredBufCol` is not 32-bit aligned.

6.2.6 Limitations

The encoder and decoder APIs defined on the omxVCM4P2 sub-doman in the foregoing sections (6.2.1-6.2.5) are compatible with data partitioning, but omxVCM4P2 does not currently provide primitives that support RVLC.

6.3 MPEG-4 Part 10 (H.264) Sub-Domain (omxVCM4P10)

This section defines the omxVCM4P10 sub-domain, which includes data structures and functions that could be used to construct an H.264 baseline profile encoder or decoder.

6.3.1 Data Structures and Enumerators

6.3.1.1 Intra 16x16 Prediction Modes

A data type that enumerates intra_16x16 macroblock prediction modes is defined as follows:

```
typedef enum {
    OMX_VC_16X16_VERT = 0,      /** Intra_16x16_Vertical */
    OMX_VC_16X16_HOR = 1,       /** Intra_16x16_Horizontal */
    OMX_VC_16X16_DC = 2,        /** Intra_16x16_DC */
    OMX_VC_16X16_PLANE = 3      /** Intra_16x16_Plane */
} OMXVCM4P10Intra16x16PredMode;
```

6.3.1.2 Intra 4x4 Prediction Modes

A data type that enumerates intra_4x4 macroblock prediction modes is defined as follows:

```
typedef enum
{
    OMX_VC_4X4_VERT = 0,        /** Intra_4x4_Vertical */
    OMX_VC_4X4_HOR  = 1,        /** Intra_4x4_Horizontal */
    OMX_VC_4X4_DC   = 2,        /** Intra_4x4_DC */
    OMX_VC_4X4_DIAG_DL = 3,     /** Intra_4x4_Diagonal_Down_Left */
    OMX_VC_4X4_DIAG_DR = 4,     /** Intra_4x4_Diagonal_Down_Right */
    OMX_VC_4X4_VR = 5,          /** Intra_4x4_Vertical_Right */
    OMX_VC_4X4_HD = 6,          /** Intra_4x4_Horizontal_Down */
    OMX_VC_4X4_VL = 7,          /** Intra_4x4_Vertical_Left */
    OMX_VC_4X4_HU = 8           /** Intra_4x4_Horizontal_Up */
} OMXVCM4P10Intra4x4PredMode;
```

6.3.1.3 Chroma Prediction Modes

A data type that enumerates intra chroma prediction modes is defined as follows:

```
typedef enum
{
    OMX_VC_CHROMA_DC = 0,       /** Intra_Chroma_DC */
    OMX_VC_CHROMA_HOR = 1,      /** Intra_Chroma_Horizontal */
```

```
    OMX_VC_CHROMA_VERT = 2,     /** Intra_Chroma_Vertical */
    OMX_VC_CHROMA_PLANE = 3     /** Intra_Chroma_Plane */
} OMXVCM4P10IntraChromaPredMode;
```

6.3.1.4 Motion Estimation Modes

A data type that enumerates H.264 motion estimation modes is defined as follows:

```
typedef enum {
    OMX_VCM4P10_FAST_SEARCH  = 0, /** Fast motion search */
    OMX_VCM4P10_FULL_SEARCH  = 1  /** Full motion search */
} OMXVCM4P10MEMode;
```

6.3.1.5 Macroblock Types

A data type that enumerates H.264 macroblock types is defined as follows:

```
typedef enum {
    OMX_VC_P_16x16          = 0,     // defined by ISO/IEC 14496-10
    OMX_VC_P_16x8           = 1,
    OMX_VC_P_8x16           = 2,
    OMX_VC_P_8x8            = 3,
    OMX_VC_PREF0_8x8        = 4,
    OMX_VC_INTER_SKIP       = 5,
    OMX_VC_INTRA_4x4        = 8,
    OMX_VC_INTRA_16x16      = 9,
    OMX_VC_INTRA_PCM        = 10
} OMXVCM4P10MacroblockType;
```

6.3.1.6 Sub-Macroblock Types

A data type that enumerates H.264 sub-macroblock types is defined as follows:

```
typedef enum {
    OMX_VC_SUB_P_8x8        = 0,     // defined by ISO/IEC 14496-10
    OMX_VC_SUB_P_8x4        = 1,
    OMX_VC_SUB_P_4x8        = 2,
    OMX_VC_SUB_P_4x4        = 3
} OMXVCM4P10SubMacroblockType;
```

6.3.1.7 Variable Length Coding (VLE) Information

```
typedef struct {
    OMX_U8      uTrailing_Ones;          /* Trailing ones; 3 at most */
    OMX_U8      uTrailing_One_Signs;     /* Trailing ones signal */
```

```
    OMX_U8      uNumCoeffs;             /* Total number of non-zero coefs,
                                           including trailing ones */
    OMX_U8      uTotalZeros;            /* Total number of zero coefs */
    OMX_S16     iLevels[16];            /* Levels of non-zero coefs, in
                                           reverse zig-zag order */
    OMX_U8      uRuns[16];              /* Runs for levels and trailing
                                           ones, in reverse zig-zag order */
} OMXVCM4P10RLEInfo;
```

The field uTrailing_One_Signs is formatted as follows: Bit0 indicates the sign of the last trailing one, Bit1indicates the sign of the 2nd-to-last trailing one, and so on. ISO/IEC 14496-2 specifies that up to 3 trailing ones are allowed, and trailing ones are ordered in inverse zig-zag scan order.

Example: `uTrailing_Ones`

Given 0 3 -1 0
0 -1 1 0
1 0 0 0
0 0 0 0

The trailing ones are [1, -1 , -1], the signs are [+/-/-], and therefore uTrailing_One_Signs will contain the value 0b 0000 0011

2 ***Note:*** *uNumCoeffs and uTotalZeros are not redundant because this structure covers blocks with 4, 15 and 16 possible coded coefficients.*

6.3.1.8 Macroblock Information

```
typedef struct {
    OMX_S32         nSliceId;             /* slice number */
    OMXVCM4P10MacroblockType    mbType;  /* MB type */
    OMXVCM4P10SubMacroblockType subMBType[4]; /* sub-block type */
    OMX_S32         qpy;                 /* qp for luma */
    OMX_S32         qpc;                 /* qp for chroma */
    OMX_U32         cbpy;                /* CBP Luma */
    OMX_U32         cbpc;                /* CBP Chroma */
    OMXMotionVector pMV0[4][4];          /* motion vector, represented
                                            using 1/4-pel units,
```

```
                                       pMV0[blocky][blockx]
                                      (blocky = 0~3, blockx =0~3) */
    OMXMotionVector pMVPred[4][4];     /* motion vector prediction,
                                       Represented using 1/4-pel
                                       units, pMVPred[blocky][blockx]
                                       (blocky = 0~3, blockx = 0~3) */
    OMX_U8          pRefL0Idx[4];      /* reference picture indices */
    OMXVCM4P10Intra16x16PredMode Intra16x16PredMode; /* best intra 16x16
                                                     prediction mode */
    OMXVCM4P10Intra4x4PredMode   pIntra4x4PredMode[16]; /* best intra 4x4
                                                        prediction mode
                                                        for each block,
                                                        pMV0 indexed as
                                                        above */
} OMXVCM4P10MBInfo, *OMXVCM4P10MBInfoPtr;
```

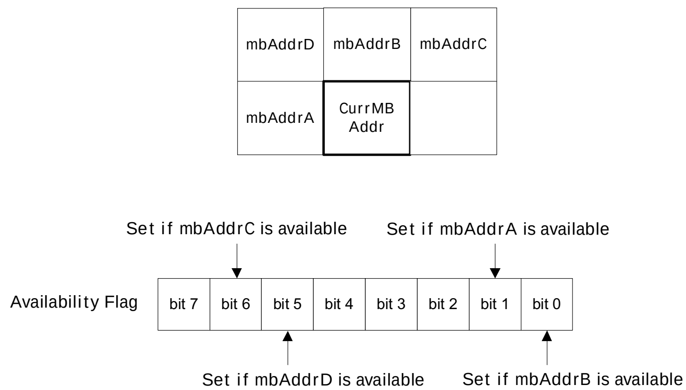
6.3.1.9 Macroblock Modes

```
typedef struct
{
     OMX_S32 b8x8BlockSplitEnable;     // enables 16x8, 8x16, 8x8
     OMX_S32 b4x4BlockSplitEnable;     // enable splitting of 8x8 blocks
     OMX_S32 bHalfSearchEnable;
     OMX_S32 bQuarterSearchEnable;
     OMX_S32 bIntra4x4Enable;          // 1=enable, 0=disable
     OMX_S32 i16x16SearchRange;        // integer pixel units
     OMX_S32 i8x8SearchRange;
     OMX_S32 i4x4SearchRange;
} OMXVCM4P10MBMode;
```

6.3.2 Buffer Conventions

6.3.2.1 Neighboring Macroblock Availability

Figure 6-4: Neighboring Macroblock Availability

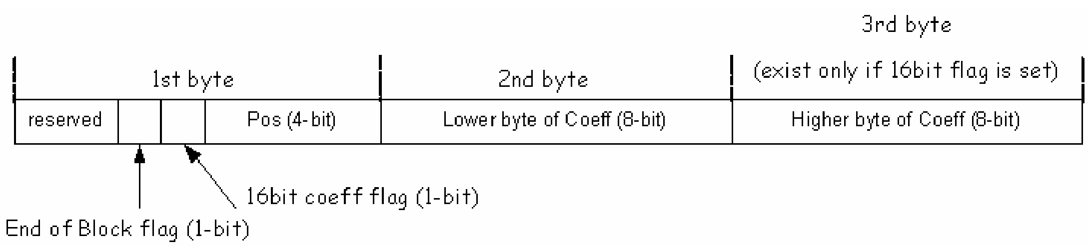


6.3.2.2 Coefficient-Position Pair Buffer

The interface between CAVLC decoding output and Transform/Dequantisation input is formed as a buffer storage structure called Coefficient-Position Pair Buffer. It stores all non-zero coefficients in 4x4 block units (or 2x2 block units for chroma DC), along with their position in the block.

The Coefficient-Position Pair Buffer Definition is shown in Figure 6-5. This figure shows how each non-zero coefficient occupies two to three bytes in the pair buffer.

Figure 6-5: Coefficient-Position Pair Buffer Definition



6.3.3 Encoder/Decoder Functions

This section defines functions that could be used to construct an H.264 baseline profile encoder or an H.264 baseline profile decoder.

6.3.3.1 Intra Prediction Functions

6.3.3.1.1 PredictIntra_4x4_C1R

Prototype

```
OMXResult omxVCM4P10_PredictIntra_4x4_C1R(OMX_U8* pSrcLeft, OMX_U8
    *pSrcAbove, OMX_U8 *pSrcAboveLeft, OMX_U8* pDst, OMX_INT leftStep,
    OMX_INT dstStep, OMXVCM4P10Intra4x4PredMode predMode, OMX_S32
    availability);
```

Description

Perform `Intra_4x4` prediction for luma samples. If the upper-right block is not available, then duplication work should be handled inside the function. Users need not define them outside.

Input Arguments

- `pSrcLeft` - Pointer to the buffer of 4 left pixels: p[x, y] (x = -1, y = 0..3); must be aligned on a 4-byte boundary.
- `pSrcAbove` - Pointer to the buffer of 8 above pixels: p[x,y] (x = 0..7, y =-1); must be aligned on a 4-byte boundary.
- `pSrcAboveLeft` - Pointer to the above left pixels: p[x,y] (x = -1, y = -1)
- `leftStep` - Step of left pixel buffer; must be a multiple of 4.
- `dstStep` - Step of the destination buffer; must be a multiple of 4.
- `predMode` - `Intra_4x4` prediction mode.
- `availability` - Neighboring 4x4 block availability flag, refer to "Neighboring Macroblock Availability".

Output Arguments

- `pDst` - Pointer to the destination buffer; must be aligned on a 4-byte boundary.

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pDst` is NULL.
- `dstStep` < 4.
- `predMode` is not in the valid range of enumeration `OMXVCM4P10Intra4x4PredMode`.
- `predMode` is `OMX_VC_4x4_VERT`, but availability doesn't set `OMX_VC_UPPER` indicating p[x,-1] (x = 0..3) is not available.
- `predMode` is `OMX_VC_4x4_HOR`, but availability doesn't set `OMX_VC_LEFT` indicating p[-1,y] (y = 0..3) is not available.
- `predMode` is `OMX_VC_4x4_DIAG_DL`, but availability doesn't set `OMX_VC_UPPER` indicating p[x,-1] (x = 0..3) is not available.
- `predMode` is `OMX_VC_4x4_DIAG_DR`, but availability doesn't set `OMX_VC_UPPER_LEFT` or `OMX_VC_UPPER` or `OMX_VC_LEFT` indicating p[x,-1] (x = 0..3), or p[-1,y] (y = 0..3) or p[-1,-1] is not available.

- `predMode` is `OMX_VC_4x4_VR`, but availability doesn't set `OMX_VC_UPPER_LEFT` or `OMX_VC_UPPER` or `OMX_VC_LEFT` indicating p[x,-1] (x = 0..3), or p[-1,y] (y = 0..3) or p[-1,-1] is not available.
- `predMode` is `OMX_VC_4x4_HD`, but availability doesn't set `OMX_VC_UPPER_LEFT` or `OMX_VC_UPPER` or `OMX_VC_LEFT` indicating p[x,-1] (x = 0..3), or p[-1,y] (y = 0..3) or p[-1,-1] is not available.
- `predMode` is `OMX_VC_4x4_VL`, but availability doesn't set `OMX_VC_UPPER` indicating p[x,-1] (x = 0..3) is not available.
- `predMode` is `OMX_VC_4x4_HU`, but availability doesn't set `OMX_VC_LEFT` indicating p[-1,y] (y = 0..3) is not available.
- availability sets `OMX_VC_UPPER`, but `pSrcAbove` is NULL.
- availability sets `OMX_VC_LEFT`, but `pSrcLeft` is NULL.
- availability sets `OMX_VC_UPPER_LEFT`, but `pSrcAboveLeft` is NULL.

---

2 ***Note:*** *pSrcAbove, pSrcAbove, pSrcAboveLeft may be invalid pointers if they are not used by intra prediction as implied in predMode.*

6.3.3.1.2 PredictIntra_16x16_C1R

Prototype

```
OMXResult omxVCM4P10_PredictIntra_16x16_C1R(OMX_U8* pSrcLeft, OMX_U8
    *pSrcAbove, OMX_U8 *pSrcAboveLeft, OMX_U8* pDst, OMX_INT leftStep,
    OMX_INT dstStep, OMXVCM4P10Intra16x16PredMode predMode, OMX_S32
    availability);
```

Description

Perform Intra_16x16 prediction for luma samples. If the upper-right block is not available, then duplication work should be handled inside the function. Users need not define them outside.

Input Arguments

- `pSrcLeft` - Pointer to the buffer of 16 left pixels: p[x, y] (x = -1, y = 0..15); must be aligned on a 16-byte boundary.
- `pSrcAbove` - Pointer to the buffer of 16 above pixels: p[x,y] (x = 0..15, y= -1); must be aligned on a 16-byte boundary.
- `pSrcAboveLeft` - Pointer to the above left pixels: p[x,y] (x = -1, y = -1)
- `leftStep` - Step of left pixel buffer; must be a multiple of 16.
- `dstStep` - Step of the destination buffer; must be a multiple of 16.
- `predMode` - Intra_16x16 prediction mode, please refer to section 3.4.1.
- `availability` - Neighboring 16x16 MB availability flag. Refer to section 3.4.4.

Output Arguments

- `pDst` - Pointer to the destination buffer; must be aligned on a 16-byte boundary.

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pDst` is NULL.
- `dstStep` < 16.
- `predMode` is not in the valid range of enumeration `OMXVCM4P10Intra16x16PredMode`
- `predMode` is `OMX_VC_16X16_VERT`, but availability doesn't set `OMX_VC_UPPER` indicating p[x,-1] (x = 0..15) is not available.
- `predMode` is `OMX_VC_16X16_HOR`, but availability doesn't set `OMX_VC_LEFT` indicating p[-1,y] (y = 0..15) is not available.
- `predMode` is `OMX_VC_16X16_PLANE`, but availability doesn't set `OMX_VC_UPPER_LEFT` or `OMX_VC_UPPER` or `OMX_VC_LEFT` indicating p[x,-1](x = 0..15), or p[-1,y] (y = 0..15), or p[-1,-1] is not available.
- availability sets `OMX_VC_UPPER`, but `pSrcAbove` is NULL.
- availability sets `OMX_VC_LEFT`, but `pSrcLeft` is NULL.
- availability sets `OMX_VC_UPPER_LEFT`, but `pSrcAboveLeft` is NULL.

---

2 ***Note:*** *pSrcAbove, pSrcAbove, pSrcAboveLeft may be invalid pointers if they are not used by intra prediction implied in predMode.*

---

2 ***Note:*** *OMX_VC_UPPER_RIGHT is not used in intra_16x16 luma prediction.*

6.3.3.1.3 PredictIntraChroma8x8_C1R

Prototype

```
OMXResult omxVCM4P10_PredictIntraChroma8x8_C1R(OMX_U8* pSrcLeft, OMX_U8
    *pSrcAbove, OMX_U8 *pSrcAboveLeft, OMX_U8* pDst, OMX_INT leftStep,
    OMX_INT dstStep, OMXVCM4P10IntraChromaPredMode predMode, OMX_S32
    availability);
```

Description

Performs Intra prediction for chroma samples.

Input Arguments

- `pSrcLeft` - Pointer to the buffer of 8 left pixels: p[x, y] (x = -1, y= 0..7); must be aligned on an 8-byte boundary.

- `pSrcAbove` - Pointer to the buffer of 8 above pixels: p[x,y] (x = 0..7, y = -1); must be aligned on an 8-byte boundary.
- `pSrcAboveLeft` - Pointer to the above left pixels: p[x,y] (x = -1, y = -1)
- `leftStep` - Step of left pixel buffer; must be a multiple of 8.
- `dstStep` - Step of the destination buffer; must be a multiple of 8.
- `predMode` - Intra chroma prediction mode, please refer to section 3.4.3.
- `availability` - Neighboring chroma block availability flag, please refer to "Neighboring Macroblock Availability".

Output Arguments

- `pDst` - Pointer to the destination buffer; must be aligned on an 8-byte boundary.

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If any of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pDst` is NULL.
- `dstStep` < 8.
- `predMode` is not in the valid range of enumeration `OMXVCM4P10IntraChromaPredMode`.
- predMode is `OMX_VC_CHROMA_VERT`, but availability doesn't set `OMX_VC_UPPER` indicating p[x,-1] (x = 0..7) is not available.
- `predMode` is `OMX_VC_CHROMA_HOR`, but availability doesn't set `OMX_VC_LEFT` indicating p[-1,y] (y = 0..7) is not available.
- `predMode` is `OMX_VC_CHROMA_PLANE`, but availability doesn't set `OMX_VC_UPPER_LEFT` or `OMX_VC_UPPER` or `OMX_VC_LEFT` indicating p[x,-1](x = 0..7), or p[-1,y] (y = 0..7), or p[-1,-1] is not available.
- availability sets `OMX_VC_UPPER`, but `pSrcAbove` is NULL.
- availability sets `OMX_VC_LEFT`, but `pSrcLeft` is NULL.
- availability sets `OMX_VC_UPPER_LEFT`, but `pSrcAboveLeft` is NULL.

---

2 ***Note:*** *pSrcAbove, pSrcAbove, pSrcAboveLeft may be invalid pointer if they are not used by intra prediction implied in predMode.*

---

2 ***Note:*** *OMX_VC_UPPER_RIGHT is not used in intra chroma prediction.*

6.3.3.2 Interpolation Functions

6.3.3.2.1 InterpolateLuma_C1R

Prototype

```
OMXResult omxVCM4P10_InterpolateLuma_C1R (const OMX_U8* pSrc, OMX_S32
    srcStep, OMX_U8* pDst, OMX_S32 dstStep, OMX_S32 dx, OMX_S32 dy, OMXSize
    roi);
```

Description

Performs quarter-pixel interpolation for inter luma MB. It is assumed that the frame is already padded when calling this function.

Input Arguments

- `pSrc` - Pointer to the source reference frame buffer
  if roi.width==4, 4-byte alignment required
  if roi.width==8, 8-byte alignment required
  if roi.width==16, 16-byte alignment required
- `srcStep` - reference frame step, in bytes; must be a multiple of roi.width
- `dstStep` - destination frame step, in bytes; must be a multiple of roi.width
- `dx` - Fractional part of horizontal motion vector component in 1/4 pixel unit
- `dy` - Fractional part of vertical motion vector y component in 1/4 pixel unit
- `roi` - Dimension of the interpolation region

Output Arguments

- `pDst` - Pointer to the destination frame buffer
  if roi.width==4, 4-byte alignment required
  if roi.width==8, 8-byte alignment required
  if roi.width==16, 16-byte alignment required

Returns

If the function runs without error, it returns `OMX_StsNoErr.`

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pSrc` or `pDst` is NULL.
- `srcStep` or `dstStep` < `roi.width`.
- `dx` or `dy` is out of range [0-3].
- `roi.width` or roi.height is out of range {4, 8, 16}.
- `roi.width` is equal to 4, but `pDst` is not 4 byte aligned.
- `roi.width` is equal to 8 or 16, but `pDst` is not 8 byte aligned.
- `srcStep` or `dstStep` is not a multiple of 8.

6.3.3.2.2 InterpolateChroma_C1R

Prototype

```
OMXResult omxVCM4P10_InterpolateChroma_C1R (const OMX_U8* pSrc, OMX_S32
    srcStep, OMX_U8* pDst, OMX_S32 dstStep, OMX_S32 dx, OMX_S32 dy, OMXSize
    roi);
```

Description

Performs 1/8-pixel interpolation for inter chroma MB.

Input Arguments

- `pSrc` - Pointer to the source reference frame buffer
- `srcStep` - Reference frame step in bytes
- `dstStep` - Destination frame step in bytes; must be a multiple of roi.width.
- `dx` - Fractional part of horizontal motion vector component in 1/8 pixel unit
- `dy` - Fractional part of vertical motion vector component in 1/8 pixel unit
- `roi` - Dimension of the interpolation region

Output Arguments

- `pDst` - Pointer to the destination frame buffer
  if roi.width==2, 2-byte alignment required
  if roi.width==4, 4-byte alignment required
  if roi.width==8, 8-byte alignment required

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pSrc` or `pDst is` NULL.
- `srcStep` or `dstStep` < 8.
- `dx` or `dy` is out of range [0-7].
- `roi.width` or `roi.height` is out of range {2,4,8}.
- `roi.width` is equal to 2, but `pDst` is not 2-byte aligned.
- `roi.width` is equal to 4, but `pDst` is not 4-byte aligned.
- `roi.width` is equal to 8, but `pDst` is not 8 byte aligned.
- `srcStep` or `dstStep` is not a multiple of 8.

6.3.3.3 Deblocking Functions

6.3.3.3.1 FilterDeblockingLuma_VerEdge_I

Prototype

```
OMXResult omxVCM4P10_FilterDeblockingLuma_VerEdge_I(OMX_U8* pSrcDst, OMX_S32
   srcdstStep, OMX_U8* pAlpha, OMX_U8* pBeta, OMX_U8* pThresholds, OMX_U8
   *pBS);
```

Description

Performs deblocking filtering on four vertical edges of the luma macroblock(16x16).

Input Arguments

- `pSrcDst` - Pointer to the initial and resultant coefficients; must be aligned on an 8-byte boundary.
- `srcdstStep` - Step of the arrays; must be a multiple of 8.
- `pAlpha` - Array of size 2 of Alpha Thresholds (the first item is alpha threshold for external vertical edge, and the second item is for internal vertical edge)
- `pBeta` - Array of size 2 of Beta Thresholds (the first item is alpha threshold for external vertical edge, and the second item is for internal vertical edge)
- `pTresholds` - Array of size 16 of Thresholds (TC0) (values for the left edge of each 4x4 block, arranged in vertical block order); must be aligned on a 4-byte boundary.
- `pBS` - Array of size 16 of BS parameters (arranged in vertical block order); must be aligned on a 4-byte boundary.

Output Arguments

- `pSrcDst` - Pointer to the initial and resultant coefficients

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- Either of the pointers in `pSrcDst`, `pAlpha`, `pBeta`, `pTresholds`, or `pBS` is NULL.
- `pSrcDst` is not 8-byte aligned.
- `srcdstStep` is not a multiple of 8.

6.3.3.3.2 FilterDeblockingLuma_HorEdge_I

Prototype

```
OMXResult omxVCM4P10_FilterDeblockingLuma_HorEdge_I(OMX_U8* pSrcDst, OMX_S32
   srcdstStep, OMX_U8* pAlpha, OMX_U8* pBeta, OMX_U8* pThresholds, OMX_U8
   *pBS);
```

Description

Performs deblocking filtering on four horizontal edges of the luma macroblock (16x16).

Input Arguments

- `pSrcDst` - Pointer to the initial and resultant coefficients; must be aligned on an 8-byte boundary.
- `srcdstStep` - Step of the arrays; must be a multiple of 8.
- `pAlpha` - Array of size 2 of Alpha Thresholds (the first item is alpha threshold for external vertical edge, and the second item is for internal horizontal edge)
- `pBeta` - Array of size 2 of Beta Thresholds (the first item is alpha threshold for external horizontal edge, and the second item is for internal horizontal edge)
- `pThresholds` - Array of size 16 containing thresholds, TC0, for the top horizontal edge of each 4x4 block, arranged in horizontal block order; must be aligned on a 4-byte boundary.
- `pBS` - Array of size 16 of BS parameters (arranged in horizontal block order); must be aligned on a 4-byte boundary.

Output Arguments

- `pSrcDst` - Pointer to the initial and resultant coefficients

Returns

- If the function runs without error, it returns `OMX_StsNoErr`.
- If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:
  - — Either of the pointers in `pSrcDst`, `pAlpha`, `pBeta`, `pTresholds`, or `pBS` is NULL.
  - — `pSrcDst` is not 8-byte aligned.
  - — srcdstStep is not a multiple of 8.

6.3.3.3.3 FilterDeblockingChroma_VerEdge_I

Prototype

```
OMXResult omxVCM4P10_FilterDeblockingChroma_VerEdge_I(OMX_U8* pSrcDst,
    OMX_S32 srcdstStep, OMX_U8* pAlpha, OMX_U8* pBeta, OMX_U8* pThresholds,
    OMX_U8 *pBS);
```

Description

Performs deblocking filtering on four vertical edges of the chroma macroblock (8x8).

Input Arguments

- `pSrcDst` - Pointer to the initial and resultant coefficients; must be aligned on an 8-byte boundary.
- `srcdstStep` - Step of the arrays; must be a multiple of 8.
- `pAlpha` - Array of size 2 of alpha thresholds (the first item is alpha threshold for external vertical edge, and the second item is for internal vertical edge)
- `pBeta` - Array of size 2 of beta thresholds (the first item is alpha threshold for external vertical edge, and the second item is for internal vertical edge)
- `pThresholds` - Array of size 8 containing thresholds, TC0, for the left vertical edge of each 4x2 chroma block, arranged in vertical block order; must be aligned on a 4-byte boundary.
- `pBS` - Array of size 16 of BS parameters (values for each 2x2 chroma block, arranged in vertical block order).This is the same with the pBS passed into `FilterDeblockLuma_VerEdge`; must be aligned on a 4-byte boundary.

Output Arguments

- `pSrcDs` - Pointer to the initial and resultant coefficients

Returns

- If the function runs without error, it returns `OMX_StsNoErr`.
- If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:
  - Either of the pointers in `pSrcDst`, `pAlpha`, `pBeta`, `pTresholds`, or `pBS` is NULL.
  - `pSrcDst` is not 8-byte aligned.
  - `srcdstStep` is not a multiple of 8.
  - `pTresholds` is not 4-byte aligned.
  - `pBS` is not 4-byte aligned.

6.3.3.3.4 FilterDeblockingChroma_HorEdge_I

Prototype

```
OMXResult omxVCM4P10_FilterDeblockingChroma_HorEdge_I(OMX_U8* pSrcDst,
   OMX_S32 srcdstStep, OMX_U8* pAlpha, OMX_U8* pBeta, OMX_U8* pThresholds,
   OMX_U8 *pBS);
```

Description

Performs in-place deblock filtering on the horizontal edges of the chroma macroblock (8x8).

Input Arguments

- `pSrcDst` - pointer to the input coefficients; must be aligned on an 8-byte boundary.
- `srcdstStep` – array step; must be a multiple of 8.
- `pAlpha` - array of size 2 containing alpha thresholds; the first element contains the threshold for the external horizontal edge, and the second element contains the threshold for internal horizontal edge
- `pBeta` - array of size 2 containing beta thresholds; the first element contains the threshold for the external horizontal edge, and the second element contains the threshold for the internal horizontal edge
- `pThresholds` - array of size 8 containing thresholds, TC0, for the top horizontal edge of each 2x4 chroma block, arranged in horizontal block order; must be aligned on a 4-byte boundary.
- `pBS` - array of size 16 containing BS parameters for each 2x2 chroma block, arranged in horizontal block order; must be aligned on a 4-byte boundary.

Output Arguments

- `pSrcDst` - pointer to the updated coefficients

Returns

- If the function runs without error, it returns `OMX_StsNoErr`.
- If one or more of the following conditions occurs, the function returns `OMX_StsBadArgErr`:
  - any of the following pointers is NULL: `pSrcDst`, `pAlpha`, `pBeta`, `pTresholds`, or `pBS`.
  - `pSrcDst` is not 8-byte aligned.
  - `srcdstStep` is not a multiple of 8.

— `pTresholds` is not 4-byte aligned.

— `pBS` is not 4-byte aligned.

6.3.3.3.5 DeblockLuma_I

Prototype

```
OMXResult omxVCM4P10_DeblockLuma_I (OMX_U8* pSrcDst, OMX_S32 srcdstStep,
   OMX_U8* pAlpha, OMX_U8* pBeta, OMX_U8* pThresholds, OMX_U8 *pBS);
```

Description

This function performs deblock filtering the horizontal and vertical edges of a luma macroblock (16x16).

Input Arguments

- `pSrcDst` - pointer to the input macroblock; must be aligned on an 8-byte boundary.
- `srcdstStep` - image width; must be a multiple of 8.
- `pAlpha` - pointer to a 2x2 table of alpha thresholds, organized as follows: {external vertical edge, internal vertical edge, external horizontal edge, internal horizontal edge }
- `pBeta` - pointer to a 2x2 table of beta thresholds, organized as follows: {external vertical edge, internal vertical edge, external horizontal edge, internal horizontal edge }
- `pTresholds` - pointer to a 16x2 table of threshold (TC0), organized as follows: {values for the left or above edge of each 4x4 block, arranged in vertical block order and then in horizontal block order); must be aligned on a 4-byte boundary.
- `pBS` - pointer to a 16x2 table of BS parameters arranged in scan block order for vertical edges and then horizontal edges; must be aligned on a 4-byte boundary.

Output Arguments

- `pSrcDst` - pointer to the processed macroblock

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

— Either of the pointers in `pSrcDst, pAlpha`, `pBeta`, `pTresholds` or `pBS` is NULL.

— `pSrcDst` is not 8-byte aligned.

— `srcdstStep` is not a multiple of 8.

6.3.3.3.6 DeblockChroma_I

Prototype

```
OMXResult omxVCM4P10_DeblockChroma_I(OMX_U8* pSrcDst, OMX_S32 srcdstStep,
   OMX_U8* pAlpha, OMX_U8* pBeta, OMX_U8* pThresholds, OMX_U8 *pBS);
```

Description

Performs deblocking filtering on all edges of the chroma macroblock (16x16).

Input Arguments

- `pSrcDst` - Pointer to the input coefficients; must be aligned on an 8-byte boundary.
- `srcdstStep` - Step of the arrays; must be a multiple of 8.
- `pAlpha` - pointer to a 2x2 array of alpha thresholds, organized as follows: {external vertical edge, internal vertical edge, external horizontal edge, internal horizontal edge }
- `pBeta` - pointer to a 2x2 array of Beta Thresholds, organized as follows: { external vertical edge, internal vertical edge, external horizontal edge, internal horizontal edge }
- `pThresholds` - Array of size 8x2 of Thresholds (TC0) (values for the left or above edge of each 4x2 or 2x4 block, arranged in vertical block order and then in horizontal block order); must be aligned on a 4-byte boundary.
- `pBS` - Array of size 16x2 of BS parameters (arranged in scan block order for vertical edges and then horizontal edges); must be aligned on a 4-byte boundary.

Output Arguments

- `pSrcDst` - Pointer to the initial and resultant coefficients

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — Either of the pointers in `pSrcDst`, `pAlpha`, `pBeta`, `pTresholds`, or `pBS` is NULL.
  - — `pSrcDst` is not 8-byte aligned.

—`srcdstStep` is not a multiple of 8.

6.3.4 Decoder Functions

This section describes functions that could be used to construct a baseline profile H.264 decoder.

6.3.4.1 CAVLC Decoding Functions

6.3.4.1.1 DecodeChromaDcCoeffsToPairCAVLC

Prototype

```
OMXResult omxVCM4P10_DecodeChromaDcCoeffsToPairCAVLC(const OMX_U8**
   ppBitStream,OMX_S32* pOffset, OMX_U8* pNumCoeff, OMX_U8**ppPosCoefbuf);
```

Description

Performs CAVLC decoding and inverse raster scan for 2x2 block of ChromaDCLevel. The decoded coefficients in packed position-coefficient buffer are stored in increasing raster scan order, namely position order.

Input Arguments

- `ppBitStream` - Double pointer to current byte in bit stream buffer
- `pOffset` - Pointer to current bit position in the byte pointed to by `*ppBitStream`

Output Arguments

- `ppBitStream - *ppBitStream` is updated after each block is decoded
- `pOffset - *pOffset` is updated after each block is decoded
- `pNumCoeff -` Pointer to the number of nonzero coefficients in this block
- `ppPosCoefBuf -` Double pointer to destination residual coefficient-position pair buffer

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `ppBitStream` or `pOffset` is NULL.
- `ppPosCoefBuf` or `pNumCoeff` is NULL.
- an illegal code is encountered in the bitstream

---

2 ***Note:*** *pOffset is valid in the range [0-7].*
*Bit Position in one byte: *pOffset*
|Most Least|
|1 2 3 4 5 6 7|
ppPosCoefBuf will not be updated if there are no non-zero coefficients in the currently decoded block. In this case, users are expected to keep this information to correctly bypass the transform/dequantisation of these empty blocks later.

6.3.4.1.2 DecodeCoeffsToPairCAVLC

Prototype

```
OMXResult omxVCM4P10_DecodeCoeffsToPairCAVLC(const OMX_U8** ppBitStream,
   OMX_S32* pOffset, OMX_U8* pNumCoeff, OMX_U8**ppPosCoefbuf, OMX_INT
   sVLCSelect, OMX_INT sMaxNumCoeff);
```

Description

Performs CAVLC decoding and inverse zigzag scan for 4x4 block of `Intra16x16DCLevel`, `Intra16x16ACLevel`, `LumaLevel`, and `ChromaACLevel`. Inverse field scan is not supported. The decoded coefficients in packed position-coefficient buffer are stored in increasing zigzag order instead of position order.

Input arguments

- `ppBitStream -` Double pointer to current byte in bit stream buffer
- `pOffset -` Pointer to current bit position in the byte pointed to by `*ppBitStream`
- `sMaxNumCoeff -` Maximum the number of non-zero coefficients in current block

- `sVLCSelect` - VLC table selector, obtained from number of non-zero ACcoefficients of above and left 4x4 blocks. It is equivalent to the variable `nC` described in H.264 standard table 9-5, except its value can't be less than zero.

Output Arguments

- `ppBitStream` - `*ppBitStream` is updated after each block is decoded
- `pOffset` - `*pOffset` is updated after each block is decoded
- `pNumCoeff` - Pointer to the number of nonzero coefficients in this block
- `ppPosCoefBuf` - Double pointer to destination residual coefficient-position pair buffer

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `ppBitStream` or `pOffset` is NULL.
- `ppPosCoefBuf` or `pNumCoeff` is NULL.
- `sMaxNumCoeff` is not equal to either 15 or 16.
- `sVLCSelect` is less than 0.
- an illegal code is encountered in the bitstream

6.3.4.2 Inverse Quantization/Transform/Add Residual Functions

6.3.4.2.1 TransformDequantLumaDCFromPair_C1

Prototype

```
OMXResult omxVCM4P10_TransformDequantLumaDCFromPair_C1(OMX_U8 **ppSrc,
   OMX_S16* pDst, OMX_INT QP);
```

Description

Reconstructs the 4x4 LumaDC block from the coefficient-position pair buffer, performs integer inverse, and dequantization for 4x4 LumaDC coefficients, and updates the pair buffer pointer to the next non-empty block.

Input Arguments

- `ppSrc` - Double pointer to residual coefficient-position pair buffer output by CALVC decoding; must be aligned on an 8-byte boundary.
- `QP` - Quantization parameter `QpY`

Output Arguments

- `ppSrc` - `*ppSrc` is updated to the start of next non empty block
- `pDst` - Pointer to the reconstructed 4x4 LumaDC coefficients buffer; must be aligned on a 16-byte boundary.

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `ppSrc` or `pDst` is NULL.
- `pDst` is not 8 byte aligned.
- `QP` is not in the range of [0-51].

6.3.4.2.2 TransformDequantChromaDCFromPair_C1

Prototype

```
OMXResult omxVCM4P10_TransformDequantChromaDCFromPair_C1(OMX_U8 **ppSrc,
    OMX_S16* pDst, OMX_INT QP);
```

Description

Reconstruct the 2x2 ChromaDC block from coefficient-position pair buffer, perform integer inverse transformation, and dequantization for 2x2 chroma DC coefficients, and update the pair buffer pointer to next non-empty block.

Input arguments

- `ppSrc` - Double pointer to residual coefficient-position pair buffer output by CALVC decoding
- `QP` - Quantization parameter `QpC`

Output Arguments

- `ppSrc` - *ppSrc is updated to the start of next non empty block
- `pDst` - Pointer to the reconstructed 2x2 ChromaDC coefficients buffer

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If any of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `ppSrc` or `pDst` is NULL.
- `pDst` is not 8 byte aligned.
- `QP` is not in the range of [0-51].

6.3.4.2.3 DequantTransformResidualFromPairAndAdd_C1

Prototype

```
OMXResult omxVCM4P10_DequantTransformResidualFromPairAndAdd_C1(OMX_U8
    **ppSrc, const OMX_U8* pPred, OMX_S16* pDC, OMX_U8* pDst, OMX_INT
    predStep, OMX_INT dstStep, OMX_INT QP, OMX_INT AC);
```

Description

Reconstruct the 4x4 residual block from coefficient-position pair buffer, perform dequantisation and integer inverse transformation for 4x4 block of residuals with previous intra prediction or motion compensation data, and update the pair buffer pointer to next non-empty block. If pDC == NULL, there're 16 non-zero AC coefficients at most in the packed buffer starting from 4x4 block position 0; If pDC != NULL, there're 15 non-zero AC coefficients at most in the packet buffer starting from 4x4 block position 1.

Input Arguments

- `ppSrc` - Double pointer to residual coefficient-position pair buffer output by CALVC decoding
- `pPred` - Pointer to the reference 4x4 block
- `predStep` - Reference frame step in byte
- `dstStep` - Destination frame step in byte
- `pDC` - Pointer to the DC coefficient of this block, NULL if it doesn't exist
- `QP` - `QP` Quantization parameter. It should be `QpC` in chroma 4x4 block decoding, otherwise it should be `QpY`.
- `AC` - Flag indicating if at least one non-zero AC coefficient exists

Output Arguments

- `pDst` - pointer to the reconstructed 4x4 block data

Returns

If the function runs without error, it returns `OMX_StsNoErr`.

If one of the following cases occurs, the function returns `OMX_StsBadArgErr`:

- `pPred` or `pDst` is NULL.
- `pPred` or `pDst` is not 4-byte aligned.
- `predStep` or `dstStep` is not a multiple of 4.
- `AC !=0`, but `Qp` is not in the range of [0-51] or `ppSrc == NULL`.
- `AC ==0` && `pDC` ==NULL.

6.3.5 Encoder Functions

This section describes functions that could be used to construct a baseline profile H.264 encoder. Both high-level and low-level motion estimation functions are defined. Helper functions are defined to support initialization of vendor-specific motion estimation specification structures.

6.3.5.1 Motion Estimation Helper Functions

6.3.5.1.1 MEGetBufSize

Prototype

```
OMXResult omxVCM4P10_MEGetBufSize(OMXVCM4P10MEMode MEmode, OMXVCM4P10MBMode
    *pMBOptions, OMX_U32 *pSize)
```

Description

Computes the size, in bytes, of the vendor-specific specification structure for the omxVCM4P10 motion estimation functions `BlockMatch_Integer` and `MotionEstimationMB`.

Input Arguments

- MEmode – motion estimation mode; available modes are defined by the enumerated type `OMXVCM4P10MEMode`
- pMBOptions –macroblock mode options; available modes are defined by the enumerated type OMXVCM4P10MBMode

Output Arguments

- `pSize` – pointer to the number of bytes required for the specification structure

Returns

- Standard OMXResult result. See enumeration for possible result codes.

6.3.5.1.2 MEInit

Prototype

```
OMXResult omxVCM4P10_MEInit (OMXVCM4P10MEMode MEmode, OMXVCM4P10MBMode*
   pMBOptions, void *pMESpec);
```

Description

Initializes the vendor-specific specification structure required for the omxVCM4P10 motion estimation functions `BlockMatch_Integer` and `MotionEstimationMB`. Memory for the specification structure `*pMESpec` must be allocated prior to calling the function, and should be aligned on a 4-byte boundary. The number of bytes required for the specification structure can be determined using the function `omxVCM4P10_MEGetBufSize`.

Input Arguments

- MEmode – motion estimation mode; available modes are defined by the enumerated type `OMXVCM4P10MEMode`
- pMBOptions –macroblock mode options; available modes are defined by the enumerated type OMXVCM4P10MBMode
- `pMESpec` – pointer to the uninitialized ME specification structure

Output Arguments

- `pMESpec` – pointer to the initialized ME specification structure

Returns

- Standard OMXResult result. See enumeration for possible result codes.

6.3.5.2 Motion Estimation Functions, Low-Level

6.3.5.2.1 BlockMatch_Integer

Prototype

```
OMXResult omxVCM4P10_BlockMatch_Integer (OMX_U8* pSrcOrgY, OMX_S32
   nSrcOrgStep, OMX_U8* pSrcRefY, OMX_S32 nSrcRefStep, OMX_U32 nMBPosX,
   OMX_U32 nMBPosY, OMX_U8 iBlockWidth, OMX_U8 iBlockHeight, OMX_U32
   nSearchRange, OMX_U32 nLamda, OMXMotionVector* pMVPred, OMXMotionVector*
   pMVCandidate, OMXMotionVector* pBestMV, OMX_S32* pBestCost void
   *pMESpec)
```

Description

Performs integer block match. Returns best MV and associated cost.

Input Arguments

- pSrcOrgY – Pointer to the top-left corner of the current MB.
  If iBlockWidth==4, 4-byte alignment required.
  If iBlockWidth==8, 8-byte alignment required.
  If iBlockWidth==16, 16-byte alignment required.
- pSrcRefY – Pointer to the top-left corner of the co-located MB in the reference picture
- nSrcOrgStep – Stride of the original picture plane, expressed in terms of integer pixels; must be a multiple of iBlockWidth.
- nSrcRefStep – Stride of the reference picture plane, expressed in terms of integer pixels
- nMBPosX – X coordinate of the top-left corner of the current MB, expressed in terms of integer pixels
- nMBPosY – Y coordinate of the top-left corner of the current MB, expressed in terms of integer pixels
- iBlockWidth – Width of the current block, expressed in terms of integer pixels
- iBlockHeight – Height of the current block, expressed in terms of integer pixels
- nSearchRange – Motion search range, expressed in terms of integer pixels
- nLamda – Lamda factor; used to compute motion cost
- pMVPred – Predicted MV; used to compute motion cost, expressed in terms of 1/4-pel units
- pMVCandidate – Candidate MV; used to initialize the motion search, expressed in terms of integer pixels
- pMESpec – pointer to the ME specification structure

Output Arguments

- pBestMV – Best MV resulting from integer search, expressed in terms of 1/4-pel units
- pBestCost – Motion cost associated with the best MV; computed as SAD+Lamda*BitsUsedByMV

Returns

- Standard OMXResult result. See enumeration for possible result codes.

6.3.5.2.2 BlockMatch_Half

Prototype

```
OMXResult omxVCM4P10_BlockMatch_Half(OMX_U8* pSrcOrgY, OMX_S32 nSrcOrgStep,
    OMX_U8* pSrcRefY, OMX_S32 nSrcRefStep, OMX_U8 iBlockWidth, OMX_U8
    iBlockHeight, OMX_U32 nLamda, OMXMotionVector* pMVPred, OMXMotionVector*
    pSrcDstBestMV, OMX_S32* pBestCost)
```

Description

Performs a half-pel block match using results from a prior integer search. Returns the best MV and associated cost. This function estimates the half-pixel motion vector by interpolating the integer resolution motion vector referenced by the input parameter `pSrcDstBestMV`, i.e., the initial integer MV is generated externally. The function `omxVCM4P10_BlockMatch_Integer` may be used for integer motion estimation.

Input Arguments

- pSrcOrgY – Pointer to the current position in original picture plane.

  If iBlockWidth==4, 4-byte alignment required.

  If iBlockWidth==8, 8-byte alignment required.

  If iBlockWidth==16, 16-byte alignment required.
- pSrcRefY – Pointer to the top-left corner of the co-located MB in the reference picture
- nSrcOrgStep – Stride of the original picture plane in terms of full pixels; must be a multiple of iBlockWidth.
- nSrcRefStep – Stride of the reference picture plane in terms of full pixels
- iBlockWidth – Width of the current block in terms of full pixels
- iBlockHeight – Height of the current block in terms of full pixels
- nLamda – Lamda factor, used to compute motion cost
- pMVPred – Predicted MV, represented in terms of 1/4-pel units; used to compute motion cost
- pSrcDstBestMV – The best MV resulting from a prior integer search, represented in terms of 1/4-pel units

Output Arguments

- pSrcDstBestMV – Best MV resulting from the half-pel search, expressed in terms of 1/4-pel units
- pBestCost – Motion cost associated with the best MV; computed as SAD+Lamda*BitsUsedByMV

Returns

- Standard OMXResult result. See enumeration for possible result codes.

6.3.5.2.3 BlockMatch_Quarter

Prototype

```
OMXResult omxVCM4P10_BlockMatch_Quarter(OMX_U8* pSrcOrgY, OMX_S32
    nSrcOrgStep, OMX_U8* pSrcRefY, OMX_S32 nSrcRefStep, OMX_U8 iBlockWidth,
    OMX_U8 iBlockHeight, OMX_U32 nLamda, OMXMotionVector* pMVPred,
    OMXMotionVector* pSrcDstBestMV, OMX_S32* pBestCost)
```

Description

Performs a quarter-pel block match using results from a prior half-pel search. Returns the best MV and associated cost. This function estimates the quarter-pixel motion vector by interpolating the half-pel resolution motion vector referenced by the input parameter `pSrcDstBestMV`, i.e., the initial half-pel MV is generated externally. The function `omxVCM4P10_BlockMatch_Half` may be used for half-pel motion estimation.

Input Arguments

- pSrcOrgY – Pointer to the current position in original picture plane.

  If iBlockWidth==4, 4-byte alignment required.

  If iBlockWidth==8, 8-byte alignment required.

  If iBlockWidth==16, 16-byte alignment required.
- pSrcRefY – Pointer to the top-left corner of the co-located MB in the reference picture
- nSrcOrgStep – Stride of the original picture plane in terms of full pixels; must be a multiple of iBlockWidth.
- nSrcRefStep – Stride of the reference picture plane in terms of full pixels
- iBlockWidth – Width of the current block in terms of full pixels
- iBlockHeight – Height of the current block in terms of full pixels
- nLamda – Lamda factor, used to compute motion cost
- pMVPred – Predicted MV, represented in terms of 1/4-pel units; used to compute motion cost
- pSrcDstBestMV – The best MV resulting from a prior half-pel search, represented in terms of 1/4-pel units

Output Arguments

- pSrcDstBestMV – Best MV resulting from the quarter-pel search, expressed in terms of 1/4-pel units
- pBestCost – Motion cost associated with the best MV; computed as SAD+Lamda*BitsUsedByMV

Returns

- Standard OMXResult result. See enumeration for possible result codes.

6.3.5.3 Motion Estimation Functions, High-Level

6.3.5.3.1 MotionEstimationMB

Prototype

```
OMXResult omxVCM4P10_MotionEstimationMB(OMX_U8* pSrcOrgY, OMX_S32
    nSrcOrgStep, OMX_U8* pSrcRefY, OMX_S32 nSrcRefStep, OMX_U8* pSrcRecY,
    OMX_S32 nSrcRefStep, OMX_U32 nMBPosX, OMX_U32 nMBPosY, void* pMESpec,
    OMXVCM4P10MBMode* pMBOptions, OMXVCM4P10MBInfoPtr *pMBInter,
    OMXVCM4P10MBInfoPtr *pMBIntra, OMXVCM4P10MBInfo *pMBCurr)
```

Description

Performs MB-level motion estimation and selects best motion estimation strategy from the set of modes supported in baseline profile ISO/IEC 14496-10.

Input Arguments

- pSrcOrgY – Pointer to the current position in original picture plane.
  - If iBlockWidth==4, 4-byte alignment required.
  - If iBlockWidth==8, 8-byte alignment required.
  - If iBlockWidth==16, 16-byte alignment required.
- pSrcRefY – Pointer to the top-left corner of the co-located MB in the reference picture
- pSrcRecY – Pointer to the top-left corner of the co-located MB in the reconstructed picture
- nSrcOrgStep – Stride of the original picture plane in terms of full pixels; must be a multiple of iBlockWidth.
- nSrcRefStep – Stride of the reference picture plane in terms of full pixels
- nSrcRecStep – Stride of the reconstructed picture plane in terms of full pixels
- nMBPosX – X coordinate of the top-left corner of the current MB; full-pixel units
- nMBPosY – Y coordinate of the top-left corner of the current MB; full-pixel units
- pMBOptions – Motion estimation parameters
- pMESpec – Pointer to the motion estimation specification structure; must have been allocated and initialized prior to calling this function
- pMBInter – Array, of dimension four, containing pointers to information associated with four adjacent type INTER MBs (Left, Top, Top-Left, Top-Right). Any pointer in the array may be set equal to NULL if the corresponding MB doesn't exist or is not of type INTER.
  - — pMBInter[0] – Pointer to left MB information
  - — pMBInter[1] – Pointer to top MB information
  - — pMBInter[2] – Pointer to top-left MB information
  - — pMBInter[3] – Pointer to top-right MB information.
- pMBIntra – Array, of dimension four, containing pointers to information associated with four adjacent type INTRA MBs (Left, Top, Top-Left, Top-Right). Any pointer in the array may be set equal to NULL if the corresponding MB doesn't exist or is not of type INTRA.
  - — pMBInter[0] – Pointer to left MB information
  - — pMBInter[1] – Pointer to top MB information

— pMBInter[2] – Pointer to top-left MB information

— pMBInter[3] – Pointer to top-right MB information

- pMBCurr – Pointer to information structure for the current MB.

Output Arguments

- pMBCurr – Pointer to updated information structure for the current MB after MB-level motion estimation has been completed. The following fields are updated by the ME function. Together this information quantifies the MB-level ME search results:

  — MbType,

  — subMBType[4]

  — pMV0[4][4]

  — pMVPred[4][4]

  — pRefL0Idx[4]

  — Intra16x16PredMode

  — pIntra4x4PredMode[4][4]

Returns

- Standard OMXResult result. See enumeration for possible result codes.

6.3.5.4 SAD/SATD Functions

6.3.5.4.1 SAD_4x

Prototype

```
OMXResult omxVCM4P10_SAD_4x (OMX_U8* pSrcOrg, OMX_U32 iStepOrg OMX_U8*
    pSrcRef, OMX_U32 iStepRef, OMX_S32* pDstSAD, OMX_U32 iHeight);
```

Description

This function calculates the SAD for 4x8 and 4x4 blocks.

Input Parameters

- `pSrcOrg` - Pointer to the original block; must be aligned on a 4-byte boundary.
- `iStepOrg` - Step of the original block buffer
- `pSrcRef` - Pointer to the reference block
- `iStepRef` - Step of the reference block buffer
- `iHeight` - Height of the block

Output Parameters

- `pDstSAD` - Pointer of result SAD

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `iHeight` is not equal to either 4 or 8.

6.3.5.4.2 SADQuar_4x

Prototype

```
OMXResult omxVCM4P10_SADQuar_4x (OMX_U8* pSrc, OMX_U8* pSrcRef0, OMX_U8*
    pSrcRef1, OMX_U32 iSrcStep, OMX_U32 iRefStep0, OMX_U32 iRefStep1,
    OMX_U32* pDstSAD, OMX_U32 iHeight);
```

Description

This function calculates the SAD between one block (pSrc) and the average of the other two (pSrcRef0 and pSrcRef1) for 4x8 or 4x4 blocks

Input Parameters

- `pSrc` - Pointer to the original block; must be aligned on a 4-byte boundary.
- `pSrcRef0` - Pointer to reference block 0
- `pSrcRef1` - Pointer to reference block 1
- `iSrcStep` - Step of the original block buffer
- `iRefStep0` - Step of reference block 0
- `iRefStep1` - Step of reference block 1
- `iHeight` - Height of the block

Output Parameters

- `pDstSAD` - Pointer of result SAD

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `iHeight` is not equal to either 4 or 8.

6.3.5.4.3 SADQuar_8x

Prototype

```
OMXResult omxVCM4P10_SADQuar_8x (OMX_U8* pSrc, OMX_U8* pSrcRef0, OMX_U8*
    pSrcRef1, OMX_U32 iSrcStep, OMX_U32 iRefStep0, OMX_U32 iRefStep1,
    OMX_U32* pDstSAD, OMX_U32 iHeight);
```

Description

This function calculates the SAD between one block (`pSrc`) and the average of the other two (`pSrcRef0` and `pSrcRef1`) for 8x16, 8x8, or 8x4 blocks.

Input Parameters

- `pSrc` - Pointer to the original block; must be aligned on an 8-byte boundary.
- `pSrcRef0` - Pointer to reference block 0
- `pSrcRef1` - Pointer to reference block 1
- `iSrcStep` - Step of the original block buffer
- `iRefStep0` - Step of reference block 0
- `iRefStep1` - Step of reference block 1
- `iHeight` - Height of the block

Output Parameters

- `pDstSAD` - Pointer of result SAD

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `iHeight` is not equal to either 4, 8, or 16.

6.3.5.4.4 SADQuar_16x

Prototype

```
OMXResult omxVCM4P10_SADQuar_16x (OMX_U8* pSrc, OMX_U8* pSrcRef0, OMX_U8*
   pSrcRef1, OMX_U32 iSrcStep, OMX_U32 iRefStep0, OMX_U32 iRefStep1,
   OMX_U32* pDstSAD, OMX_U32 iHeight);
```

Description

This function calculates the SAD between one block (`pSrc`) and the average of the other two (`pSrcRef0` and `pSrcRef1`) for 16x16 or 16x8 blocks. Rounding is applied according to the convention (a+b+1)>>2.

Input Parameters

- `pSrc` - Pointer to the original block; must be aligned on a 16-byte boundary.
- `pSrcRef0` - Pointer to reference block 0
- `pSrcRef1` - Pointer to reference block 1
- `iSrcStep` - Step of the original block buffer
- `iRefStep0` - Step of reference block 0
- `iRefStep1` - Step of reference block 1
- `iHeight` - Height of the block

Output Parameters

- `pDstSAD` - Pointer of result SAD

Returns

The function returns `OMX_StsNoErr` if it runs without error.

The function returns `OMX_StsBadArgErr` if one or more of the following is true:

- `iHeight` is not equal to either 8 or 16.

6.3.5.4.5 SATD_4x4

Prototype

```
OMXResult omxVCM4P10_SATD_4x4 (OMX_U8* pSrcOrg, OMX_U32 iStepOrg, OMX_U8*
    pSrcRef, OMX_U32 iStepRef, OMX_U32* pDstSAD);
```

Description

This function calculates the sum of absolute transform differences (SATD) for a 4x4 block by applying a Hadamard transform to the difference block and then calculating the sum of absolute coefficient values.

Input Parameters

- `pSrcOrg` - Pointer to the original block; must be aligned on a 4-byte boundary.
- `iStepOrg` - Step of the original block buffer; must be a multiple of 4.
- `pSrcRef` - Pointer to the reference block; must be aligned on a 4-byte boundary.
- `iStepRef` - Step of the reference block buffer; must be a multiple of 4.

Output Parameters

- `pDstSAD` - pointer to the resulting SAD.

6.3.5.5 Interpolation Functions

6.3.5.5.1 InterpolateHalfHor_Luma

Prototype

```
OMXResult omxVCM4P10_InterpolateHalfHor_Luma(OMX_U8* pSrc, OMX_U32 iSrcStep,
    OMX_U8* pDstLeft, OMX_U8* pDstRight, OMX_U32 iDstStep, OMX_U32 iWidth,
    OMX_U32 iHeight);
```

Description

This function performs interpolation for two horizontal 1/2-pel positions (-1/2,0) and (1/2, 0) - around a full-pel position.

Input Parameters

- `pSrc` - Pointer to the top-left corner of the block used to interpolate in the reconstruction frame plane.
- `iSrcStep` - Step of the source buffer.
- `iDstStep` - Step of the destination(interpolation) buffer; must be a multiple of iWidth.
- `iWidth- Width of the current block`
- `iHeight` - Height of the current block

Output Parameters

- `pDstLeft` - Pointer to the interpolation buffer of the left -pel position (-1/2, 0)
  If iWidth==4, 4-byte alignment required.
  If iWidth==8, 8-byte alignment required.
  If iWidth==16, 16-byte alignment required.
- `pDstRight` - Pointer to the interpolation buffer of the right -pel position (1/2, 0)
  If iWidth==4, 4-byte alignment required.
  If iWidth==8, 8-byte alignment required.
  If iWidth==16, 16-byte alignment required.

6.3.5.5.2 InterpolateHalfVer_Luma

Prototype

```
OMXResult omxVCM4P10_InterpolateHalfVer_Luma(OMX_U8* pSrc, OMX_U32 iSrcStep,
   OMX_U8* pDstUp, OMX_U8* pDstDown, OMX_U32 iDstStep, OMX_U32 iWidth,
   OMX_U32 iHeight);
```

Description

This function performs interpolation for two vertical 1/2-pel positions - (0, -1/2) and (0, 1/2) - around a full-pel position.

Input Parameters

- `pSrc` - Pointer to top-left corner of block used to interpolate in the reconstructed frame plane
- `iSrcStep` - Step of the source buffer.
- `iDstStep` - Step of the destination (interpolation) buffer; must be a multiple of `iWidth`.
- `iWidth` - Width of the current block
- `iHeight` - Height of the current block

Output Parameters

- `pDstUp` - Pointer to the interpolation buffer of the -pel position above the current full-pel position (0, -1/2)
  If iWidth==4, 4-byte alignment required.
  If iWidth==8, 8-byte alignment required.
  If iWidth==16, 16-byte alignment required.
- `pDstDown` - Pointer to the interpolation buffer of the -pel position below the current full-pel position (0, 1/2)
  If iWidth==4, 4-byte alignment required.
  If iWidth==8, 8-byte alignment required.
  If iWidth==16, 16-byte alignment required.

6.3.5.5.3 Average_4x

Prototype

```
OMXResult omxVCM4P10_Average_4x (OMX_U8* pPred0, OMX_U8* pPred1, OMX_U32
   iPredStep0, OMX_U32 iPredStep1, OMX_U8* pDstPred, OMX_U32 iDstStep,
   OMX_U32 iHeight);
```

Description

This function calculates the average of two 4x4 or 4x8 blocks and stores the result.

Input Parameters

- `pPred0` - Pointer to the top-left corner of reference block 0
- `pPred1` - Pointer to the top-left corner of reference block 1
- `iPredStep0` - Step of reference block 0
- `iPredStep1` - Step of reference block 1
- `iDstStep` - Step of the destination buffer
- `iHeight` - Height of the blocks

Output Parameters

- `pDstPred` - Pointer to the destination buffer. 4-byte alignment required.

6.3.5.6 Transform and Quantization Functions

6.3.5.6.1 TransformQuant_ChromaDC

Prototype

```
OMXResult omxVCM4P10_TransformQuant_ChromaDC(OMX_S16* pSrcDst, OMX_U32 iQP
   OMX_U8 bIntra);
```

Description

This function performs 2x2 hadamard transform of chroma DC coefficients and then quantizes the coefficients.

Input Parameters

- `pSrcDst` - Pointer to the 2x2 array of chroma DC coefficients. 8-byte alignment required.
- `iQP` - Quantization parameter.
- `bIntra` - Indicate whether this is an INTRA block. 1-INTRA, 0-INTER

Output Parameters

- `pSrcDst` - Pointer to transformed and quantized coefficients. 8-byte alignment required.

6.3.5.6.2 TransformQuant_LumaDC

Prototype

```
OMXResult omxVCM4P10_TransformQuant_LumaDC(OMX_S16* pSrcDst, OMX_U32 iQP);
```

Description

This function performs a 4x4 hadamard transform of luma DC coefficients and then quantizes the coefficients.

Input Parameters

- `pSrcDst` - Pointer to the 4x4 array of luma DC coefficients. 16-byte alignment required.
- `iQP` - Quantization parameter.

Output Parameters

- `pSrcDst` - Pointer to transformed and quantized coefficients. 16-byte alignment required.

6.3.5.6.3 InvTransformDequant_LumaDC

Prototype

```
OMXResult omxVCM4P10_InvTransformDequant_LumaDC(OMX_S16* pSrc,OMX_S16* pDst,
   OMX_U32 iQP);
```

Description

This function performs inverse 4x4 hadamard transform and then dequantizes the coefficients.

Input Parameters

- `pSrc` - Pointer to the 4x4 array of the 4x4 hadamard transformed and quantized coefficients. 16-byte alignment required.
- `iQP` - Quantization parameter.

Output Parameters

- `pDst` - Pointer to inverse-transformed and dequantized coefficients. 16-byte alignment required.

6.3.5.6.4 InvTransformDequant_ChromaDC

Prototype

```
OMXResult omxVCM4P10_InvTransformDequant_ChromaDC(OMX_S16* pSrc, OMX_S16*
   pDst, OMX_U32 iQP);
```

Description

This function performs inverse 2x2 hadamard transform and then dequantizes the coefficients.

Input Parameters

- `pSrc` - Pointer to the 2x2 array of the 2x2 hadamard transformed and quantized coefficients. 8-byte alignment required.
- `iQP` - Quantization parameter.

Output Parameters

- `pDst` - Pointer to inverse-transformed and dequantized coefficients. 8-byte alignment required.

6.3.5.7 Transform and Compensation Functions

6.3.5.7.1 InvTransformResidualAndAdd

Prototype

```
OMXResult omxVCM4P10_InvTransformResidualAndAdd (OMX_U8* pSrcPred, OMX_S16*
    pDequantCoeff, OMX_U8* pDstRecon, OMX_U32 iSrcPredStep, OMX_U32
    iDstReconStep, OMX_U8 bAC);
```

Description

This function performs inverse an 4x4 integer transformation to produce the difference signal and then adds the difference to the prediction to get the reconstructed signal.

Input Parameters

- `pSrcPred` - Pointer to prediction signal. 4-byte alignment required.
- `pDequantCoeff` - Pointer to the transformed coefficients. 8-byte alignment required.
- `iSrcPredStep` - Step of the prediction buffer.
- `iDstReconStep` - Step of the destination reconstruction buffer
- `bAC` - Indicate whether there is AC coefficients in the coefficients matrix.

Output Parameters

- `pDstRecon` - Pointer to the destination reconstruction buffer. 4-byte alignment required.

6.3.5.8 Compensation, Transform, and Quantization Functions

6.3.5.8.1 SubAndTransformQDQResidual

Prototype

```
OMXResult omxVCM4P10_SubAndTransformQDQResidual (OMX_U8* pSrcOrg,
    OMX_U8*pSrcPred, OMX_U32 iSrcOrgStep, OMX_U32 iSrcPredStep, OMX_S16*
    pDstQuantCoeff, OMX_S16* pDstDeQuantCoeff, OMX_S16* pDCCoeff, OMX_S8*
    pNumCoeff, OMX_U32 nThreshSAD, OMX_U32 iQP, OMX_U8 bIntra);
```

Description

This function subtracts the prediction signal from the original signal to produce the difference signal and then performs a 4x4 integer transform and quantization. The quantized transformed coefficients are stored as `pDstQuantCoeff`. This function can also output dequantized coefficients or unquantized DC coefficients optionally by setting the pointers `pDstDeQuantCoeff, pDCCoeff`.

Input Parameters

- `pSrcOrg` - Pointer to original signal. 4-byte alignment required.
- `pSrcPred` - Pointer to prediction signal. 4-byte alignment required.
- `iSrcOrgStep` - Step of the original signal buffer; must be a multiple of 4.
- `iSrcPredStep` - Step of the prediction signal buffer; must be a multiple of 4.
- `pNumCoeff` - Number of non-zero coefficients after quantization. If this parameter is not required, it is set to NULL.
- `nThreshSAD` - Zero-block early detection threshold. If this parameter is not required, it is set to 0.
- `iQP` - Quantization parameter.
- `bIntra` - Indicates whether this is an INTRA block, either 1-INTRA or 0-INTER

Output Parameters

- `pDstQuantCoeff` - Pointer to the quantized transformed coefficients. 8-byte alignment required.
- `pDstDeQuantCoeff` - Pointer to the dequantized transformed coefficients if this parameter is not equal to NULL. 8-byte alignment required.
- `pDCCoeff` - Pointer to the unquantized DC coefficient if this parameter is not equal to NULL.

6.3.5.9 VLC Functions

6.3.5.9.1 GetVLEInfo

Prototype

```
OMXResult omxVCM4P10_GetVLEInfo (OMX_S16* pSrcCoeff, OMX_U8* pScanMatrix,
   OMX_U8 bAC, OMX_U32 MaxNumCoef, OMXVCM4P10RLEInfo* pDstRLEInfo);
```

Description

This function extracts run-length encoding (RLE) information from the coefficient matrix. The results are returned in an `OMXVCM4P10RLEInfo` structure.

Input Parameters

- `pSrcCoeff` - pointer to the transform coefficient matrix. 8-byte alignment required.
- `pScanMatrix` - pointer to the scan order definition matrix
- `bAC` - indicates presence of a DC coefficient; 0 = DC coefficient present, 1= DC coefficient absent.
- `MaxNumCoef` - maximum number of coefficients.

Output Parameters

- `pDstRLEInfo` - pointer to structure that stores information for run-length coding.

7.0 Concurrency Mechanisms

7

The OpenMAX DL API allows portability over a large range of hardware architectures. Enabled hardware architectures vary from single processor MCUs or DSPs to cabled targeted hardware accelerators. It is recognized that this form of API will not provide the most efficient code on all these architectures. With this in mind, the OpenMAX standard supports two asynchronous concurrent execution methodologies. The first methodology is call the Asynchronous Development Layer (aDL), which focuses on augmenting existing DL APIs with additional concurrent execution and primitive grouping interfaces. The second methodology is the Integrated Development Layer (iDL), that maps the Integration Layer interface and behavior onto groupings of DL primitives. Both of these methodologies are defined in header files separate from the standard Development Layer interface.

7.1 Asynchronous DL (aDL)

Overview

The Asynchronous Development Layer (aDL) API augments the existing DL interfaces to provide a clear path of migration from purely static execution environments to platforms that are enabled by asynchronous and parallel capabilities while remaining backward compatible and fully portable (with regards to source code). The aDL defines a common set of changes to DL APIs that allow the user to signify which primitives are grouped together for execution and how data flows between them. In addition, a set of functions and state are defined to enable control of the group execution.

Generally, developers will use aDL as a tool to target asynchronous behavior within existing DL based codecs. The API is designed to allow existing code to be ported with minimum effort to take advantage of asynchronous facilities presented by new ISAs, co-processors, tightly coupled hardware accelerators, and separate processing elements.

Upgrading DL API to aDL API

The following changes are applied to all DL APIs for inclusion into the aDL library.

1. The DL APIs name remains mostly unchanged
   a. This maintains an obvious and evident link to the DL API
   b. The 'omx' prefix is changed to 'omxa' to distinguish between DLs and aDLs
2. One parameter is added to the end of the parameter list and one to the beginning
   a. The return type is added to the beginning of the list to allow it to be used in the aDL chain.
   b. A handle is added to the end of the list to distinguish between chains
3. All parameters are replaced with their aDL type counterparts.
4. Calls to aDL functions associate themselves with the command buffer referenced by the handle.
   a. Calls to individual aDL primitives do not signify an execution of that primitive.
   b. Calls to aDL primitives add their arguments to the command buffer. The behavior of the primitives in execution is sensitive to order in which primitives are added.
   c. Each handle refers to a separate command buffer.

Example:

```
/* DL API */
OMXResult omxVCM4P2_QuantInter_I(
  OMX_S16 *pSrcDst,
  OMX_U8  QP,
  const OMX_U8 *pQMatrix);

/* aDL API */
void omxaVCM4P2_QuantInter_I(
 const OMX_ADLRESULTP  pResult,
 OMX_ADLS16 *pSrcDst,
 OMX_ADLU8 QP,
 const OMX_ADLU8 *pQMatrix);
 const OMX_PTR   handle );
```

aDL Control APIs

The primitives in the aDL API can be linked together to form chains of increased functionality. The length of this chain can be from 1 aDL API to many aDL APIs. The sequencing of the chain of APIs is based on the order of insertion into the command buffer and the flow of data between the primitives (defined by function parameters and aDL control interfaces). In order to provide a full featured concurrent execution interface, the aDL must utilize a state machine to manage the execution of the command buffer. The APIs used to control and monitor this machine are explained below.

omxaDL_Control

Name

omxaDL_Control	Send a command to the aDL machine (via an asynchronous handle) that references a set of non-blocking DL primitive calls.

C Prototype

```
OMXResult omxaDL_Control( OMX_PTR* pHandle, OMX_INT Control );
```

Parameters

pHandle	Void double pointer identifier for asynchronous context.
Control	Command identifier for the asynchronous primitives. See possible values below.

Description

Handles that point to a variable set to NULL are implicitly initialized (as per OMX_ADL_CMD_CREATE) to new state unless otherwise specified.

Table 2 - aDL Control Values

Control Value	*Effect*
OMX_ADL_CMD_BLOCK	*Blocks program flow until the data from all primitives associated with the handle is valid. This command does not change the aDL state. This command will return errors encountered while executing the command buffer.*
OMX_ADL_STATE_CAUSAL	*Tells the asynchronous state machine that the primitives associated with the handle have a strongly causal relationship. All primitives are assumed to be executed in the order they were added to the command buffer. The function will return an error (OMX_StsAdlInvalidCmdErr) if the state is in Executing.* *Explicitly setting the state to OMX_ADL_STATE_CAUSAL will also lock the command buffer as per*

	OMX_ADL_STATE_LOCK.
OMX_ADL_STATE_NONCAUSAL	*Tells the asynchronous state machine that the primitives associated with the handle have a noncausal relationship defined only by their data dependencies. The primitives are NOT assumed to be executed in order. This is the default operation.* *The function will return an error (OMX_StsAdlInvalidCmdErr) if the state is in OMX_ADL_STATE_EXECUTE.* *Explicitly setting the state to OMX_ADL_STATE_NONCAUSAL will also lock the command buffer as per OMX_ADL_STATE_LOCK.*
OMX_ADL_STATE_DEBUG	*Enables the debug mode of the aDL engine. When in this mode, the engine will do extra error checking as detailed in the errors section. An aDL state machine may fail to enter the DEBUG state if it is not supported. Such a failure must be reported via the OMXResult of the omxaDL_CONTROL API (OMX_StsAdlNoDebugErr).*
OMX_ADL_STATE_NODEBUG	*Disables the debug state*
OMX_ADL_CMD_CREATE	*Initializes a new handle. Depending on the implementation, this may allocate memory to associate with the handle pointer. The state associated with a new handle is implicitly OMX_ADL_STATE_STOP and OMX_ADL_STATE_UNLOCK.*
OMX_ADL_CMD_FREE	*Frees all data associated with a handle and set the handle value to NULL. Freeing the handle implies a machine stop (OMX_ADL_STATE_STOP) and implies that all currently executing aDL function outputs are not valid.*
OMX_ADL_STATE_STOP	*This attempts to stop the execution of an aDL group. STOP is not guaranteed to end execution and data is not guaranteed to be valid when STOP is called.*
OMX_ADL_CMD_STATUS	*Returns a value of '1' if the data*

	represented is valid (e.g. NOT in OMX_ADL_STATE_EXECUTE). This command does not change the aDL state. This command will return a '1' if the state machine is stopped and does gaurentee the validity of the function outputs if the machine was stopped during execution. If errors are encountered while executing the command buffer the error will be returned once execution has stopped instead of '1'.
OMX_ADL_STATE_EXECUTE	*This initiates the execution of the asynchronous state machine. All state set during execution will NOT effect currently running primitives and will generate an OMX_StsAdlInvalidCmdErr error.* *Explicitly setting the state to OMX_ADL_STATE_EXECUTE will also lock the command buffer as per OMX_ADL_STATE_LOCK.* *NOTE: Some synchronous (i.e. non-threaded) implementations may choose to block until execution is complete when this state is set.*
OMX_CMD_RESET_BUFFERS	*All auto-incrementing buffers (as designated in omxaDL_RegisterIndex) are set to their original address values.*
OMX_ADL_STATE_LOCK	*Explicitly locks the command buffer. Attempting to add additional primitives to a locked buffer will result in a OMX_StsAdlLockErr error and no modification to the state or command buffer.*
OMX_ADL_STATE_UNLOCK	*Unlocks the command buffer so that more commands may be added. Attempting to unlock while in OMX_ADL_STATE_EXECUTE will return an error (OMX_StsAdlInvalidCmdErr).*

Return Value

```
OMX_RESULT – Conveys error values if less than '1' and if the state
machine is in a non-executing state.  '1' represents a non-execution
state if the STATUS command is used.  The value is '0' otherwise.
```

omxaDL_RegisterIndex

Name

omxaDL_RegisterIndex — Associate a data buffer and configuration options with a parameter index with an aDL context.

C Prototype

```
OMXResult omxaDL_RegisterIndex( OMX_PTR pHandle, OMX_PTR pBuffer,
            OMX_INT index, OMX_PTR pIncrement ,OMX_U32 flags );
```

Parameters

pHandle	Void pointer identifier for asynchronous context.
pBuffer	Data buffer associated with the aDL parameter index.
index	Parameter index for exchanging data between primitives.
pIncrement	Address offset that is added to the pBuffer address after each aDL execution.
flags	Configurable options associated with a parameter index. These options (see Table 3) are signified by binary switches in the flags variable. The value of the flags variable can be generated by ORing desired option macros together.

Description

This function controls the association of parameter indices and buffers for aDL chain inputs and outputs.

Table 3 - RegisterIndex Flag Settings

Flag	*Description*
OMX_ADL_FLAG_INPUT	*'1' Signifies the data in the buffer is valid for input into an aDL chain parameter. Subsequent calls to register an Input buffer with the same index are ignored unless set as Immediate.*
OMX_ADL_FLAG_OUTPUT	*'1' Signifies the buffer*

	must be valid at the end of an aDL chain execution. '0' Signifies do not care. All buffers set as the Output for an index will contain the same data unless set as Immediate.
OMX_ADL_FLAG_IMMEDIATE	*'1' Signifies that buffer is directly relevant to its position in the chain. When calling with respect to an input on an existing index (signified via the input flag), the new buffer overrides the existing buffer associated with that index from that point on in the chain. When calling with respect to an output on an existing index (signified via the output flag), the position relative data should be copied from the existing buffer associated with the index into the new buffer in the omxaDL_RegisterIndex call.*
OMX_ADL_FLAG_VOLATILE	*'1' Signifies the buffer value may change during aDL chain executions. '0' Signifies it will never change. Implementations will only access volatile buffers once (in order) per input per execution.*
OMX_ADL_FLAG_FIXEDDATA	*'1' Signifies the data in a buffer will never change.*
OMX_ADL_FLAG_READONLY	*'1' Signifies the buffer may not be written into. Setting a buffer as both an output and read only will cause RegisterIndex to fail with an*

	OMX_StsAdlInvalidCmdErr error.

Return Value

`OMX_RESULT – Values less than ‘0’ signify an error.`

Parameter Controls

The structures of the aDL data types contain additional information that allows the user to control the flow of data between the primitives. All DL types are converted to aDL types as follows:

typedef struct OMX_ADL<DL_TYPE_NAME><P if a pointer>
{
  <DL_TYPE >   <* if pointer>data or pData;
  OMX_S32index;
} OMX_ADL<DL_TYPE_NAME><P if a pointer>;

If the name does not start with “OMX_”:

typedef struct OMXADL <DL_TYPE_NAME><P if a pointer>
{
  <DL_TYPE >   <* if pointer>data or pData;
  OMX_S32index;
} OMXADL <DL_TYPE_NAME><P if a pointer>;

Example:

```
typedef struct OMX_ADLS16P
{
 OMX_S16 *pData;
 OMX_S32 index;
} OMX_ADLS16;

typedef struct OMX_ADLRESULTP
{
 OMX_RESULT *pData;
 OMX_S32 index;
} OMX_ADLS16;

typedef struct OMX_ADLINT
{
 OMX_INT data;
 OMX_S32 index;
} OMX_ADLS16;

typedef struct OMXADLRect
{
 OMXRect data;
```

```
    OMX_S32 index;
   } OMXADLRect;
```

The data/pData field is defined as the data normally associated with a parameter. If the data is a pointer and is set to NULL, the implementation assumes the data is not needed by the user (in the case of an output), provided by an earlier primitive in the chain (in the case of an input), or provided by means of omxaDL_RegisterIndex(). If not provided earlier in the chain, it is assumed that the NULL value is the intended value for the primitive's argument.

The 'index' field designates which other parameter in the chain that the variable is linked to. Providing the same index to an output parameter of one aDL primitive and to an input parameter of a subsequent aDL primitive, will notify the implementation that the data should be passed between the two. Two parameters in the same function call must not have the same index, unless the index is of value '0'. If '0', the implementation assumes the value is singular and not needed by the rest of the chain. All function output arguments with an index of '0' are not guaranteed to be valid after execution (data/pData may not be used to store the results and pData can be set to NULL to explicitly signify the output data is not needed). For all other values, the data/pData field is ignored if the index is present earlier in the chain. Index values used once and only once on an output argument of a function are implicitly outputs of the chain (as if flagged as an output by a omxaDL_RegisterIndex() call). Lastly, any linked set of variables must be of the same type with the exception of dereferenced variables. If a '<TYPE_NAME>P' variable is used before a '<TYPE_NAME>' variable with the same index, the second variable is implicitly dereferenced.

Errors

All implementations are required to return the following errors regardless of debugging state.

Error	*Description*
OMX_StsAdlLockErr	*Failed command insertion due to the command buffer being locked.*
OMX_StsAdlFailedLockErr	*Failed state transition due to the inability of the implementation to lock the command buffer. This could be from a failed compilation, invalid chain, invalid data, etc.*
OMX_StsAdlInvalidCmdErr	*Failed command due to incompatibility with the current state.*
OMX_StsAdlResourceErr	*Failed state transition due to the inability of the implementation to secure appropriate resources or because the resources were lost while executing.*
OMX_StsAdlNoDebugErr	*Returned when setting the state to debug mode if the implementation does not support it.*
OMX_StsAdlExecErr	*A fatal error was encountered while executing.*
OMX_StsAdlInvalidHandle	*The handle passed into the call is invalid or corrupt.*
OMX_StsAdlErr	*Fatal error catchall*

Implementations that support the debug state must return the following errors when in that state. These errors may be optionally returned in the 'nodebug' state.

Debug Message	*Description*
OMX_StsAdlInvalidDataErr	*Data passed into a primitive is not in the correct format*
OMX_StsAdlInvalidPathErr	*Reported in I/O mismatches and recursive paths*
OMX_StsAdlInvalidParamErr	*Catch all error for other parameter related issues.*
OMX_StsAdlMemResourceErr	*Unable to secure memory resources or lost during execution.*
OMX_StsAdlExecResourceErr	*Unable to secure execution resources or lost during execution.*
OMX_StsAdlLostConnectionErr	*Special loss of resources or state due to losing connection to those resources (possibly caused by loss power).*

Example of Utilization

The aDL is a context based API that processes DL primitives in a user defined order. Once initialized, a handle references an empty group of primitives and their associated state. By calling an aDL function with the handle as an argument, the user pushes the associated primitive into the handle's command buffer. The order of execution is defined by the order of insertion and data dependencies. The data paths themselves are established by use of the omxaDL_RegisterIndex() function or implicit use of indices in function parameters. The following pieces of example code strive to illustrate these concepts for the user.

aDL Code Examples

Simple Example

```
OMX_PTR pHandle_chain1 = NULL;
OMX_BOOL bDoneRunning = FALSE;

/* Build a new chain */
if( omxaDL_Control( &pHandle_chain1, OMX_ADL_STATE_CREATE) >=0)
{
   omxaXYZ_API_A(.., pHandle_chain1); /* Push API_A into the
command buffer*/
   omxaXYZ_API_B(.., pHandle_chain1); /* Push API_B into the
command buffer*/
   omxaXYZ_API_C(.., pHandle_chain1); /* Push API_C into the
command buffer*/
   omxaXYZ_API_D(.., pHandle_chain1); /* Push API_D into the
command buffer*/
}
else
{
   /* ERROR */
   exit();
}

/* This line explicitly locks the FIFO */
/* This is not absolutely necessary as it is inferred in the
execution, but an explicit call allows the implementation to
optimize the command buffer before execution. */
omxaDL_Control( &pHandle_chain1, OMX_ADL_STATE_LOCK);

/* Run the chain */
omxaDL_Control( &pHandle_chain1, OMX_ADL_STATE_EXECUTE);


/* Run other code until the data is ready */
do
```

```
{
   /* Other Code*/

   if( omxaDL_status( pHandle_chain1, &bDoneRunning) < 0)
   {
      /* ERROR */
      exit();
   }
} while(!bDoneRunning);

/* Or just block until the data is ready */
if( omxaDL_Control( &pHandle_chain1, OMX_ADL_STATE_BLOCK) < 0)
{
   /* ERROR */
   exit();
}

/* When done with the primitive block … free the aDL state
resources*/
omxaDL_control( &pHandle_chain1, OMX_ADL_STATE_FREE);
pHandle_chain1 = NULL;
```

RegisterIndex Setup Example

```
omxaVCM4P2_DCT8x8blk(
  OMX_ADLRESULTP pResult,
  OMX_ADLS16P *pSrc,
  OMX_ADLS16P *pDst
  OMX_PTR handle);

omxaVCM4P2_QuantInter_I(
  OMX_ADLRESULTP pResult,
  OMX_ADLS16P *pSrcDst,
  OMX_ADLU8 QP,
  const OMX_ADLU8P *pQMatrix,
  OMX_PTR handle);


#define INTDONTCARE(x) (OMX_ADLINT){ x, 0}
#define U8DONTCARE(x) (OMX_ADLU8){ x, 0}
#define U8PDONTCARE(x) (OMX_ADLU8P){ x, 0}
#define RESULTDONTCARE (OMX_ADLRESULTP){NULL,0}
#define S16PINDEX(x) (OMX_ADLS16P){NULL,x}

{
/* Setup the source buffer for the first function in the chain
(DCT) and increment the buffer address with each execution. */
  omxaDL_RegisterIndex(pHandle_chain1, (OMX_PTR)pDCTSrcBuf, 1,
0x000000FF , OMX_ADL_FLAG_INPUT );
```

```
/* Setup the destination buffer for the last function in the
chain (Quant) and increment the buffer address with each
execution. */
  omxaDL_RegisterIndex(pHandle_chain1, (OMX_PTR)pQuantDstBuf, 2,
0x000000FF , OMX_ADL_FLAG_OUTPUT );

/* Add DCT to the command buffer with two data streams (one
input and one output) */
omxaVCM4P2_DCT8x8blk( RESULTDONTCARE, S16PINDEX( 1 ), S16PINDEX( 2
), pHandle_chain1);

/* Copy the DCT result out into a separate buffer before it is
utilized by the inplace quant function. */
  omxaDL_RegisterIndex(pHandle_chain1,
(OMX_PTR)pDCTOutputCopyBuf, 2, 0x000000FF , OMX_ADL_FLAG_OUTPUT
| OMX_ADL_FLAG_IMMEDIATE );

/* Add Quant to the command buffer with one inplace stream*/
omxaVCM4P2_QuantInter_I( RESULTDONTCARE, S16PINDEX( 2 ),
U8DONTCARE(QP), U8PDONTCARE(pQMatrix), pHandle_chain1);

/* Perform execution and codec work */
}
```

Concurrent Chains Example

```
/* aDL sample code for running ME in parallel with quantization,
   transform, inverse transform and inverse quantisation */

OMX_PTR pHandle_chain_ME = NULL;
OMX_PTR pHandle_chain_TQIQIT = NULL;
OMX_BOOL bDoneRunning = FALSE;

OMX_ADLU8P MESrc = U8PDONTCARE(pSrcRefBuf);
OMXADLRectP MERefRect = RECTPDONTCARE(pRefRect);
OMX_ADLU8P MESrcCurrBuf = U8PDONTCARE(pSrcCurrBuf);
OMXADLVCM4P2CoordinateP MECurrPointPos =
COORDPDONTCARE(pCurrPointPos);
OMXADLVCMotionVectorP MESrcPreMV = MVPDONTCARE(pSrcPreMV);
OMX_ADLINT MESrcPreSAD = INTPDONTCARE(pSrcPreSAD);
OMX_ADLPTR MEState = PTRDONTCARE(pState);
OMXADLVCM4P2MacroblockType MEDstMBType = {pDstMBType , 1};
OMXADLVCMotionVector MEDstMV = {pDstMV , 2};
OMX_ADLINT MEDstSAD = {pDstSAD , 3};
OMX_ADLRESULTP ResultDontCare = {NULL , 0};
```

```
/* Set up of ME chain in parallel */
if( omxaDL_control( &pHandle_chain_ME, OMX_ADL_STATE_CREATE)
>=0)
{
   omxaVCM4P2_MotionEstimationMB (ResultDontCare, &MESrc,
INTDONTCARE(RefWidth), &MERefRect, &MESrcCurrBuf,
&MECurrPointPos, &MESrcPreMV, &MESrcPreSAD, INTDONTCARE(rndVal),
INTDONTCARE(searchRange), &MEState, &MEDstMBType, &MEDstMV,
MEDstSAD, pHandle_chain_ME);
}
else
{
   /* Error */
}

/* Set up of Transform and Quantisation, Inverse Transform
 and Inverse Quantisation chain */
OMX_ADLS16P TRBSrc = S16PDONTCARE(pSrc);
OMX_ADLS16P TRBDst = {pDst, 1};
OMX_ADLS16P TRBRec = {pRec, 5};
OMX_ADLU8P  TRBMatrix = U8PDONTCARE(pQMatrix);

OMX_ADLS16P QISrc = {NULL, 1};
OMX_ADLU8P  QIMatrix = U8PDONTCARE(pQMatrix);

OMX_ADLS16P IDCTSrc = {NULL , 1};
OMX_ADLS16P IDCTDst = {pEndDst , 2};

if( omxaDL_control( &pHandle_chain_TQIQIT,
OMX_ADL_STATE_INITIALIZE) >=0)
{
   omxaVCM4P2_TransRecBlockCeof_inter(ResultDontCare, &TRBSrc,
&TRBDst, &TRBRec), U8DONTCARE(QP), &TRBMatrix,
pHandle_chain_TQIQIT);
   omxaVCM4P2_QuantInvInter_S16_I(ResultDontCare, &QISrc,
INTDONTCARE(QP), &QIMatrix, pHandle_chain_TQIQIT);
   omxaVCM4P2_IDCT8x8blk(ResultDontCare, &IDCTSrc, &IDCTDst,
pHandle_chain_TQIQIT);
}
else
{
  /* ERROR */
}


/* Run the two chains */
omxaDL_control( &pHandle_chain_ME, OMX_ADL_STATE_EXECUTE);
omxaDL_control( &pHandle_chain_TQIQIT, OMX_ADL_STATE_EXECUTE);

/* Wait for the two chain to finish */
omxaDL_control( &pHandle_chain_ME, OMX_ADL_STATE_BLOCK);
```

```
    omxaDL_control( &pHandle_chain_TQIQIT, OMX_ADL_STATE_BLOCK);

    /* Free up the aDL resources*/
    omxaDL_control( &pHandle_chain_ME, OMX_ADL_STATE_FREE);
    omxaDL_control( &pHandle_chain_TQIQIT, OMX_ADL_STATE_FREE);
    pHandle_chain_ME = NULL;
    pHandle_chain_TQIQIT = NULL;
```

7.2 Integrated DL (iDL)

Overview

The Integrated Development Layer (iDL) API enables DL functionality in a concurrent execution environment by merging the DL API with the higher level Integration Layer (IL) API. This enables users to take advantage of a wider range of architectures while using the known asynchronous interface of the IL state machine. The iDL API defines a set of rules for converting the DL primitives into IL interface structures. Those structures are then added to the existing set of IL structures and controlled via the existing IL APIs.

Generally, developers will use iDL as a tool to target specific asynchronous behavior provided by a platform vendor. This may be used to benefit existing codecs, but is most optimally used when considered in the codec design. The behavior and ordering of DL functionality is specific to an iDL component for better resource utilization and performance optimizations within the system (coupled hardware, etc.) Developers will need to determine how the optimization and portability trade-offs inherent in the platform's iDL based components will impact their codec.

Upgrading a DL codec to an iDL codec

5. The DL APIs name is used only in connection with an IL port configuration structure.
    a. This maintains an obvious and evident link to the DL API
    b. The DL configuration structure maintains the same parameter set as the DL API although data input and output is not used in iDL based components. Data for the primitives are passed through other component ports as defined by the component documentation.
6. All DL function parameters are replaced with their iDL type counterparts.
7. Multiple iDL calls can be externally configured through a single IL input port. The internal iDL chaining topology is described in the associated component documentation.
    a. Data buffers fed into the first iDL in an IL component may be structured as specified by standard IL configuration structures but may be internally broken down and processed with DL specified granularities.
    b. Codecs may be broken into one or multiple iDL based IL components.

Example:

```
/* DL API */
OMXResult omxVCM4P2_QuantInter_I(
  OMX_S16 *pSrcDst,
  OMX_U8 QP,
  const OMX_U8 *pQMatrix);

/* iDL Configuration Structure */
typedef struct OMX_iDL_PARAM_VCM4P2_QuantInter_I {
  OMX_S16 *pSrcDst,
  OMX_U8 QP,
  OMX_U8 *pQMatrix
 }OMX_iDL_PARAM_ VCM4P2_QuantInter_I;
```

iDL Concurrent Execution

The iDL functions can be linked together within an IL component to form a chain of iDL configurable functions. The length of this chain can be from 1 iDL function to many iDL functions. The sequencing of the chain of functions is based on IL component documentation. Concurrency can be enabled within an IL component or by splitting a codec into multiple IL components based on concurrency requirements and connecting them. iDL relies on the IL API for execution control. If pre-buffering is required for a group of iDLs, this group can be split into a separate IL component.

Errors

iDL based components utilize IL error codes.

Example of Utilization

The iDL port configuration structures enable DL based codec configurations to be ported to IL based codecs where the use of IL configuration structures and codecs is clearly described in the OpenMAX IL 1.0 specification. The following code exemplifies how an IL component containing a single DL function would be initialized and operated. For a component containing multiple DL functions, the connectivity between DL functions must also be specified with the component documentation. Dynamic reconfiguration of DL functions is not specified for 1.0.

```
/*
This Sample code does the following:
1. Loads the component
2. Initializes a new port configuration structure on the
component's input port
3. Sets the quantization parameter for the internal DL function
and ignores the other data pointers
4. Puts the comomponent into the IDLE state
5. Puts the component in to the Execute state
6. Send the component a normal data buffer using Empty This
buffer

It should be noted that for this example, the following would be
included with the component documentation:

This component comprises a single DL function
(VCM4P2_QuantInter_I) the pSrcDst data for this function is
excracted from data received from the IL input port and the
reluting data  is written to the IL data output port.  The
pQMatrix is restructed to NULL for this particular component.
Only the QP parameter may be set and get.

*/

/* assuming we have the following struct
typedef struct OMX_iDL_PARAM_VCM4P2_QuantInter_I {
  OMX_S16 *pSrcDst,
  OMX_U8 QP,
  OMX_U8 *pQMatrix
 }OMX_iDL_PARAM_ VCM4P2_QuantInter_I;

defined as following struct in OMX_Video.h
OMX_VIDEO_PARAM_VCM4P2

and following index in OMX_Index.h
OMX_IndexParamVideoVCM4P2

*/
```

```
sample_client_code(){

    OMX_VIDEO_PARAM_VCM4P2 sVCM;
    OMX_BUFFERHEADERTYPE *pBufferHdr = NULL;

    // load component
    OMX_Init();
    OMX_GetHandle(&hComp, cComponentName, pWrappedAppData,
pWrappedCallbacks);

    // get all video ports
    OMX_GetParameter(hComp, OMX_IndexParamVideoInit,
(OMX_PTR)&sPortParam);

    // set the image width
    OMX_GetParameter(hComp, OMX_IndexParamVideoVCM4P2,
(OMX_PTR)&sVCM);
    sVCM.QP = QPvalue;
    OMX_SetParameter(hComp, OMX_IndexParamVideoVCM4P2,
(OMX_PTR)&sVCM);

    // command to idle state
    OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateIdle,
0);

    // allocate buffers
 for(i = sPortParam.nStartPortNumber; i <
sPortParam.nStartPortNumber + sPortParam.nPorts; i++)
{
    sPortDef.nPortIndex = i;
   OMX_GetParameter(hComp, OMX_IndexParamPortDefinition,
(OMX_PTR)&sPortDef);

for (j = 0x0; j < sPortDef.nBufferCountActual; j++)
       {
       OMX_AllocateBuffer(hComp, &pBufferHdr,
sPortDef.nPortIndex, 0, sPortDef.nBufferSize);
        PSEUDO_Add_Buffer_To_Internal_List(pBufferHdr);
        }
 }

    PSEUDO_Wait_For_Transition_Event();

    // command to executing state
    OMX_SendCommand(hComp, OMX_CommandStateSet,
OMX_StateExecuting, 0);
    PSEUDO_Wait_For_Transition_Event();

    // process buffers
    PSEUDO_Do_For_All_Output_Buffers{
        PSEUDO_Read_Buffer_From_Internal_List(pBufferHdr);
```

```
            OMX_FillThisBuffer(hComp, pBufferHdr);
        }

        PSEUDO_Do_For_All_Input_Buffers{
            PSEUDO_Read_Buffer_From_Internal_List(pBufferHdr);
            OMX_EmptyThisBuffer(hComp, pBufferHdr);
        }

        PSEUDO_Cleanup();
    }
```

Optional Extensions (DLx)

A

Introduction

This appendix defines the function set that comprises the DL extension API ("DLx").

Purpose

The information contained in this appendix is provided for reference only. In certain cases it may provide a preview of functions that may appear in an unspecified future revision of the DL specification.

Scope

The information contained in this appendix is outside the scope of the DL specification. All definitions associated with the DLx API are subject to change at any time without notice.

Compliance

The DLx API defines an optional set of functions. Implementation of the DLx API is neither required nor recommended in the context of OpenMAX DL compliance.

Data Structures

OMXEncVLCParam

```
typedef struct OMXEncVLCParam
{
OMX_S16 Intraflag;
OMX_S16 IntraVlc;
OMX_S16 Blktype;
OMX_S16 CBP;
OMX_S16 Escape;
OMX_S16 Numblks;
} OMXEncVLCParam;
```

OMXDecVLDParam

```
typedef struct OMXDecVLDParam
{
OMX_S16 intraflag;
OMX_S16 intraVlc;
OMX_S16 blktype;
OMX_S16 CBP;
OMX_S16 escape;
OMX_S16 numblks;
} OMXDecVLDParam;
```

OMXCopyRefParam

```
typedef struct OMXCopyRefParam
{
OMX_S16 blk_width;
OMX_S16 Blk_height;
OMX_S16 total_ref_width;
OMX_S16 output_type;
OMX_S16 inter;
OMX_S16 y;
OMX_S16 Nh;
OMX_S16 Nv;
OMX_S16 msb;
OMX_S16 round_shift;
```

```
OMX_S16 round_type;
} OMXCopyRefParam;
```

OMXSadmultipleParam

```
typedef struct OMXSadmultipleParam
{
OMX_S16 Block_width;
OMX_S16 Block_height;
OMX_S16 Target_width;
OMX_S16 ref_width;
OMX_S16 step_horz;
OMX_S16 step_vert;
OMX_S16 nsteps_horz;
OMX_S16 nsteps_vert;
OMX_S16 minmax_clear;
OMX_S16 Minmax;
OMX_S16 id_mode;
OMX_S16 id_value;
OMX_S16 Thresh;
OMX_S16 center_block;
OMX_S16 skip_field;
} OMXSadmultipleParam;
```

OMXSadmultipleInterpParam

```
typedef struct OMXSadmultipleInterpParam
{
OMX_S16 block_width;
OMX_S16 block_height;
OMX_S16 target_width;
} OMXSadmultipleInterpParam;
```

OMXSoSmultipleParam

```
typedef struct OMXSoSmultipleParam
{
OMX_S16 Block_width;
OMX_S16 Block_height;
OMX_S16 Target_width;
OMX_S16 ref_width;
```

```
OMX_S16 step_horz;
OMX_S16 step_vert;
OMX_S16 nsteps_horz;
OMX_S16 nsteps_vert;
OMX_S16 minmax_clear;
OMX_S16 Minmax;
OMX_S16 id_mode;
OMX_S16 id_value;
OMX_S16 Thresh;
OMX_S16 center_block;
OMX_S16 skip_field;
} OMXSoSmultipleParam;
```

OMXSoSmultipleInterpParam

```
typedef struct OMXSoSmultipleInterpParam
{
OMX_S16 block_width;
OMX_S16 block_height;
OMX_S16 target_width;
} OMXSoSmultipleInterpParam;
```

Functions for the Video Coding MPEG-4 Subdomain (omxVCM4P2)

DCT8x8blks_DLx

Prototype

```
OMXResult omxVCM4P2_DCT8x8blks_DLx (OMX_S16 *pBlkIn, OMX_S16 *pBlkOut,
   OMX_INT knum_blks);
```

Description

Performs the 2D IDCT on multiple 8x8 data blocks.

Input Arguments

- `pBlkIn` – starting address of input matrix, 16-byte aligned.
- `knum_blks` – number of 8x8 blocks.

Output Arguments

- `pBlkOut` – starting address of output matrix, 16-byte aligned.

Returns

- `OMXResult`

IDCT8x8blks_DLx

Prototype

```
OMXResult omxVCM4P2_IDCT8x8blks_DLx (OMX_S16 *pBlkIn, OMX_S16 *pBlkOut,
   OMX_INT knum_blks);
```

Description

This function performs the 2D IDCT on 8x8 data blocks.

Input Arguments

- `pBlkIn` – starting address of input matrix, 16-byte aligned.
- `knum_blks` – number of 8x8 blocks.

Output Arguments

- `pBlkOut` – starting address of output matrix, 16-byte aligned.

Returns

- `OMXResult`

hpDiffMBY_DLx

Prototype

```
OMXResult omxVCM4P2_hpDiffMBY_DLx (OMX_U8 *pSrcBuf, OMX_U8 *pRefBuf,
   OMXMotVect *pSrcMV, OMX_INT rndCtrl, OMX_INT width, OMX_INT fourMVmode,
   OMX_S16 *pResBuf, OMXMotVect **pDstMV);
```

Description

This is a middle level function call that performs half-pixel motion estimation and difference computation for a frame with no picture extension.

Input Arguments

- `pSrcBuf` – Pointer to the current macroblock in the current frame buffer.
- `pRefBuf` – Pointer to the best-matched macroblock in the reference frame buffer.
- `pSrcMV` – Pointer to the best motion vector in full pixel units for the luminance vector from integer motion estimation.
- `rndCtrl` – Round control parameter used for half-pixel interpolation.
- `width` – Width of the luminance component of the frame.
- `fourMVmode` – Enable bit to indicate that there can be four MVs for the 16x16 macroblock.

Output Arguments

- `pResBuf` – Pointer to the residual/error values as result of mismatch between the best-matched pixels in the reference macroblock and the current macroblock. This is a linear array of 256 elements.

- `pDstMV` – Pointer to the best-matched motion vectors for all the four blocks in the luminance macroblock. This is an array of four elements, with each element representing a motion vector (in half-pixel units) for one block. If the macroblock has only one best matched motion vector, then the best-matched motion vector is replicated to all the members of the array.

Returns

- `OMXResult`

hpDiffBlkC_DLx

Prototype

```
OMXResult omxVCM4P2_hpDiffBlkC_DLx (OMX_U8 *pSrcBuf, OMX_U8 *pRefBuf,
   OMX_INT rndCtrl, OMXMotVect *pSrcMV, OMX_INT width, OMX_S16 *pResBuf);
```

Description

Performs half-pixel interpolation chrominance pixels (Cb) and calculates difference data. This is a middle level function call that computes the half-pixel refinement and the difference between the current block (Cb) and the computed half/integer pixels for a frame

---

2 ***Note:*** *This is a super-block function. The luminance motion vector is used to determine the type of interpolation pixel (horizontal, vertical, horizontal-vertical, or integer) to be calculated. The same function call can also be used for H.263 encoding.*

Input Arguments

- `pSrcBuf` – Pointer to the current macroblock in the current frame buffer.
- `pRefBuf` – Pointer to the best-matched macroblock in the reference frame buffer.
- `rndCtrl` – Round control parameter used for half-pixel interpolation.
- `width` – Width of the chrominance component of the frame.
- `pSrcMV` – Pointer to the best-matched motion vector (in half pixel units).

Output Arguments

- `pResBuf` – Pointer to the residual/error values as result of mismatch between the best-matched pixels in the reference macroblock and the current macroblock. This is a linear array of 64 elements.

Returns

- `OMXResult`

hpDiffBlkDCTY_DLx

Prototype

```
OMXResult omxVCM4P2_hpDiffBlkDCTY_DLx (OMX_U8 *pSrcBuf, OMX_U8 *pRefBuf,
    OMXMotVect *pSrcMV, OMX_INT rndCtrl, OMX_INT width, OMX_INT fourMVmode,
    OMX_S16 *pResBuf, OMXMotVect **pDstMV);
```

Description

Performs half-pixel interpolation, calculates difference data and DCT for the luminance macroblock in a frame with no picture extension.

This is a super-block function. This is a middle level function call that performs half-pixel motion estimation, computes difference values and performs DCT on each of the four luminance blocks.

---

2 ***Note:*** *This function may also be used for H.263 encode.*

Input Arguments

- `pSrcBuf` – Pointer to the current macroblock in the current frame buffer.
- `pRefBuf` – Pointer to the best-matched macroblock in the reference frame buffer.
- `pSrcMV` – Pointer to the best motion vector in full pixel units for the luminance vector from integer motion estimation.
- `rndCtrl` – Round control parameter used for half-pixel interpolation.
- `width` – Width of the chrominance component of the frame.
- `fourMVmode` – Enable bit to indicate that there can be four MVs for the 16x16 macroblock.

Output Arguments

- `pResBuf` – Pointer to the DCT macroblock array. This is a linear array of 256 elements.
- `pDstMV` – Pointer to the best-matched motion vectors for all four blocks in the luminance macroblock. This is an array of four elements, with each element representing a motion vector (in half-pixel units) of one block. If the macroblock has only one best-matched motion vector then the best-matched motion vector is replicated to all the members of the array.

Returns

- `OMXResult`

hpDiffBlkDCTC_DLx

Prototype

```
OMXResult omxVCM4P2_hpDiffBlkDCTC_DLx (OMX_U8 *pSrcBuf, OMX_U8 *pRefBuf,
    OMX_INT rndCtrl, OMX_INT width, OMX_INT fourMVmode, OMXMotVect *pSrcMV,
    OMX_S16 *pResBuf);
```

Description

Perform half-pixel interpolation, calculate difference and apply DCT on the resultant block data. This is a super-block function. This is a middle-level function call that performs half-pixel interpolation as required on the best matched chrominance (Cb/Cr), computes the difference between the current block and the computed half/integer pixels and applies DCT on the resultant for a frame.

2 ***Note:*** *The same function call can also be used for H.263 encoding.*

Input Arguments

- `pSrcBuf` – Pointer to the current macroblock in the current frame buffer.
- `pRefBuf` – Pointer to the best matched block in the chrominance reference frame buffer.
- `rndCtrl` – Round control parameter used for half-pixel interpolation.
- `width` – Width of the chrominance component of the frame.
- `fourMVmode` – Enable bit to indicate that there can be four MVs for the 16x16 macroblock.
- `pSrcMV` – Pointer to the best-matched motion vector (in half pixel units).

Output Arguments

- `pResBuf` – Pointer to the residual/error values as result of mismatch between the best-matched pixels in the reference macroblock and the current macroblock. This is a linear array of 64 elements.

Returns

- `OMXResult`

QuantACDCScanBlkIntra_DLx

Prototype

```
OMXResult omxVCM4P2_QuantACDCScanBlkIntra_DLx (OMX_S16 *pSrcBuf, OMX_INT
   dcScaler, OMX_INT quant, OMX_INT acFlag, OMX_S16 *pACDCArray, OMX_U8
   *numCoef, OMX_U8 *scanDirection, OMX_S16 *pResBuf);
```

Description

This is a middle level function call that performs quantization, AC/DC prediction, and scanning of one intra block of coefficients of data.

2 ***Note:*** *This is a super-block function. The same function call can also be used for H.263 encoding. In this case, the user should set the dcScaler and quant to the same value.*

Input Arguments

- `pSrcBuf` – Pointer to the block.

- `dcScaler` – Quantization parameter for DC coefficient.
- `quant` – Quantization parameter for AC coefficients.
- `acFlag` – Flag indication whether AC prediction is switched on.
- `pACDCArray` – Pointer to array containing AC/DC (I/O) coefficients of previous blocks.

Output Arguments

- `numCoef` – Pointer to the number of valid coefficients in the block.
- `scanDirection` – Pointer to the scan lookup table.
- `pResBuf` – Pointer to the linear array of scanned coefficients, where numCoef points to the number of valid elements.

Returns

- `OMXResult`

QuantScanBlkInter_DLx

Prototype

```
OMXResult omxVCM4P2_QuantScanBlkInter_DLx (OMX_S16 *pSrcBuf, OMX_INT
   quant,OMX_S16 *pResBuf);
```

Description

Perform quantization, AC/DC prediction and scan for a block of data. This is a middle-level function call that performs quantization, AC/DC prediction and scanning of a inter-block of coefficients.

2 ***Note:*** *This is a super-block function. This function may also be used in H.263 video coding.*

Input Arguments

- `pSrcBuf` – Pointer to the input block.
- `quant` – Quantization parameter for the block coefficients.

Output Arguments

- `pResBuf` – Pointer to the linear array of scanned coefficients, where numCoef points the number of valid elements.

Returns

- The number of valid coefficients in a block.

IQMBIntra_DLx

Prototype

```
OMXResult omxVCM4P2_IQMBIntra_DLx (OMX_S16 *pSrcBuf, OMX_INT dcScaler,
   OMX_INT quant, OMX_U8 *numCoef, OMX_U8 *pResBuf);
```

Description

This is a middle-level function call that performs inverse quantization for all six blocks (YcbCr) in the current macroblock.

---

2 ***Note:*** *This function can also be used in H.263 video coding. To use the function for H.263 baseline, set dcScaler to 8.*

Input Arguments

- `pSrcBuf` – Pointer to the inverse transformed data. This is a linear array containing all the 384 coefficients of the macroblock, whose coefficients are arranged in a serial fashion.
- `dcScaler` – Quantization parameter for DC coefficient.
- `quant` – Quantization parameter for AC coefficients.
- `numCoef` – Pointer to the number of valid coefficients of the blocks (6) in a macroblock.

Output Arguments

- `pResBuf` – Pointer to the location of resultant data.

Returns

- `OMXResult`

IQMBInter_DLx

Prototype

```
OMXResult OMX_INT omxVCM4P2_IQMBInter_DLx (OMX_S16 *pSrcBuf, OMX_INT quant,
   OMX_U8 *numCoef, OMX_S16 *pResBuf);
```

Description

Performs inverse quantization of the current luminance macroblock. This is a middle-level function call that performs inverse quantization of the current luminance macroblock.

---

2 ***Note:*** *This is a super-block function. This function can also be used in H.263 video coding.*

Input Arguments

- `pSrcBuf` – Pointer to IDCT block data. This is a linear array containing all the 256 coefficients of the macroblock, whose coefficients are arranged in a serial fashion.
- `quant` – Quantization parameter for AC coefficients.

Output Arguments

- `pResBuf` – Pointer to the location of the quantized data.

Returns

- `numCoef` – Number of valid coefficients of the blocks (6) in a macroblock.

IDCTMBIntra_DLx

Prototype

```
OMXResult omxVCM4P2_IDCTMBIntra_DLx (OMX_S16 *pSrcBufY, OMX_INT width,
   OMX_U8 *pResBufY, OMX_U8 *pResBufCb, OMX_U8 *pResBufCr);
```

Description

This is a middle level function call that performs inverse DCT for all the six blocks (YCcbCr) in the current macroblock. The function can also be considered as the reconstruction step for an intra macroblock in a P-VOP.

---

2 ***Note:*** *This is a super-block function. The coefficients in the input buffer are arranged in linear block-wise form; for example, coefficients of one block (8x8) of data are succeeded by the coefficients of the next block. This function can also be used in H.263 video coding.*

Input Arguments

- `pSrcBufY` – Pointer to the inverse transformed data. This is a linear array containing all the 384 coefficients of the macroblock, whose coefficients are arranged in a serial fashion.
- `width` – Width of the output Y frame component.

Output Arguments

- `pResBufY` – Pointer to the location of IDCT data for Y component.
- `pResBufCb` – Pointer to the location of IDCT data for Cb component.
- `pResBufCr` – Pointer to the location of IDCT data for Cr component.

Returns

- `OMXResult`

IQIDCTMBIntra_DLx

Prototype

```
OMXResult omxVCM4P2_IQIDCTMBIntra_DLx (OMX_S16 *pSrcBuf, OMX_INT dcScaler,
   OMX_INT quant, OMX_INT width, OMX_U8 *pResBufY, OMX_U8 *pResBufCb, OMX_U8
   *pResBufCr);
```

Description

This is a middle level function call that performs inverse discrete cosine transform and inverse quantization for all six blocks (YCbCr) in the current macroblock. This is a super-block function.

---

2 ***Note:*** *In the input buffer (pSrcBuf), the coefficients of each block are assumed to be pre-processed. For example, every block is assumed to have 64 coefficients arranged in the normal order with the invalid coefficients assigned to zero. This function can also be used in H.263 video coding.*

Input Arguments

- `pSrcBuf` – Pointer to the quantized data. This is a linear array containing all the 384 coefficients of the macroblock, whose coefficients are arranged in a serial fashion.
- `dcScaler` – Quantization parameter for DC coefficient.
- `quant` – Quantization parameter for AC coefficients.
- `width` – Width of the Y component of the frame.

Output Arguments

- `pResBufY` – Pointer to the location of the reconstructed buffer where the luminance data is to be written.
- `pResBufCb` – Pointer to the location of the reconstructed buffer where the chrominance data (Cb) is to be written.
- `pResBufCb` – Pointer to the location of the reconstructed buffer where the chrominance data (Cr) is to be written.

Returns

- `OMXResult`

IQIDCTMBReconYInter_DLx

Prototype

```
OMXResult omxVCM4P2_IQIDCTMBReconYInter_DLx (OMX_S16 *pSrcBuf, OMX_INT
   quant, OMX_U8 *pRefBuf, OMXMotVect **pSrcMV, OMX_INT rndCtrl, OMX_INT
   width, OMX_U8 *pResBuf);
```

Description

Performs inverse discrete cosine transform, inverse quantization, and reconstructs the current luminance macroblock. This is a middle level function call that performs inverse quantization for all the six blocks in the current macroblock.

2 ***Note:*** *This is a super-block function. This function can also be used in H.263 video coding.*

Input Arguments

- `pSrcBuf` – Pointer to the quantized data. This is a linear array containing all the 256 coefficients of the macroblock, whose coefficients are arranged in a serial fashion.
- `quant` – Quantization parameter for AC coefficients.
- `pRefBuf` – Pointer to the best-matched macroblock in the reference (frame) buffer.
- `rndCtrl` – Round control parameter used for half-pixelinterpolation.
- `pSrcMV` – Pointer to the half-pixel motion vector for all the four blocks.
- `width` – Width of the Y component of the frame.

Output Arguments

- `pResBuf` – Pointer to the location of the reference buffer where the luminance data is to be written.

Returns

- `OMXResult`

IQIDCTReconYInter_DLx

Prototype

```
OMXResult omxVCM4P2_IQIDCTReconYInter_DLx (OMX_S16 *pSrcBuf, OMX_INT quant,
   OMX_U8 *pRefBuf, OMXMotVect **pSrcMV, OMX_INT rndCtrl, OMX_INT width,
   OMX_U8 *pResBuf);
```

Description

Perform inverse discrete cosine transform, inverse quantization and reconstruct the current luminance macroblock.

2 ***Note:*** *This function performs inverse quantization for all six blocks in the current macroblock.*

Input Arguments

- `pSrcBuf` – Pointer to the quantized data. This is a linear array containing all the 64 coefficients of the macroblock, whose coefficients are arranged in a serial fashion.
- `quant` – Quantization parameter for AC coefficients.
- `pRefBuf` – Pointer to the best-matched macroblock in the reference (frame) buffer.
- `rndCtrl` – Round control parameter used for half-pixel interpolation.

- `pSrcMV` – Pointer to the half-pixel motion vector for the block.
- `width` – Width of the Y component of the frame.

Output Arguments

- `pResBuf` – Pointer to the location of the reference buffer where the luminance data is to be written.

Returns

- `OMXResult`

IQIDCTReconCInter_DLx

Prototype

```
OMXResult omxVCM4P2_IQIDCTReconCInter_DLx (OMX_S16 *pSrcBuf, OMX_INT quant,
    OMX_U8 *pRefBuf, OMXMotVect **pSrcMV, OMX_INT rndCtrl, OMX_INT width,
    OMX_U8 *pResBuf);
```

Description

This is a middle level function call that performs inverse discrete cosine transform, inverse quantization, and reconstruction of the current Cb/Cr block.

---

2 ***Note:*** *This is a super-block function. Reconstruction involves computing half-pixels for each of the blocks and adding them to the corresponding inverse-quantized block data. In the input buffer (pSrcBuf), the coefficients of the block are assumed to be pre-processed, for example, to have 64 coefficients arranged in the normal order with the invalid coefficients assigned to zero. This function can also be used in H.263 video coding.*

Input Arguments

- `pSrcBuf` – Pointer to the quantized data. This is a linear array containing all the 64 coefficients of the current Cb/Cr block.
- `quant` – Quantization parameter for all the coefficients.
- `pRefBuf` – Pointer to the best-matched macroblock in the reference (frame) buffer.
- `pSrcMV` – Pointer to the half-pixel motion vector for all the four blocks.
- `rndCtrl` – Round control parameter used for half-pixel interpolation.
- `width` – Width of the chrominance component of the frame.

Output Arguments

- `pResBuf` – Pointer to the location of the reference buffer where the luminance data is to be written.

Returns

- `OMXResult`

DecVLDCoeff_DLx

Prototype

```
OMXResult omxVCM4P2_DecVLDCoeff_DLx (omxUTBD_sState *pBitstreamInput,
   OMX_S16 *pCoeffBlkOutput, OMXDecVLDParam *sparams);
```

Description

Reads the bit stream and decodes the value using VLD table.

Input Arguments

- `pBitstreamInput` – Pointer to input bit stream.
- `pCoeffBlkOutput` – Pointer to output.
- `sparams` – Pointer to Parameters: for OMXDecVLDParam.

Sparams:

- `*intraflag` – 0?Non intra macro block
- *1?Intra macro block
- *intraVlc – Intra Vlc flag
- *blktype – Luma / Chroma Flag
- *CBPCBP
- *escape – MPEG4 Escape coding
- *0?escape3 only
- *1?escape 1,2,3
- *numblks – Number of blocks - can vary from 1 to 6

Returns

- `OMXResult`

EncVLCCoeff_DLx

Prototype

```
OMXResult omxVCM4P2_EncVLCCoeff_DLx (OMX_S16 *pInputBlk, omxUTBD_sState*
   pBitstreamOutput, OMXEncVLCParam *sparams);
```

Description

Encodes a VLC coefficient to the bit stream using VLC table.

Input Arguments

- `pInputBlk` – Pointer to input block.
- `sparams` – Pointer to parameters for OMXEncVLCParam.

Output Arguments

- `pBitstreamOutput` – Pointer to bit stream.

Returns

- `OMXResult`

MotionCompensateBlks_DLx

Prototype

```
OMXResult omxVCM4P2_MotionCompensateBlks_DLx(OMX_U8 *refblk_ptr, OMX_U8
    *pred_blk_ptr, OMXCopyRefParam *params);
```

Description

This motion compensation API performs type A compensation. It can be used for luma and chroma motion compensation. It supports single/four motion vector, both types of rounding control and can work on interleaved and separated input data.

Input Arguments

- `Refblk_ptr` – Starting address of reference block.
- `sparams` – Pointer to Parameters: for `OMXCopyRefParam`.

Sparams:

- `blk_width` – Width of each output block.
- `Blk_height` – Height of each output block.
- `total_ref_width` – width of the reference block.
- `inter` – Indicates whether input is in interleaved YCrYCb mode, interleaved CbCr mode, or in separated only Y’s, only Cr’s, or only Cb mode. In all the modes the input is assumed to be in byte-packed fashion (For example, two values in a word). For inter = 0, width should be in bytes. For inter = 1 or 2, width should be in words.
    — *0?separated
    — *1?interleaved YCbYCr
    — *2- interleaved CbCr
- `output_type` – Signed short/unsigned byte.
- `y` – Indicates whether luma or chroma motion compensation has to be performed.
    — *0?chroma
    — *1?luma
- For chroma motion comp both Cb and Cr blocks will be written out in sequential order.
- Parameters:Nh – Number of blocks to be motion compensated in horizontal direction.
- Parameters:Nv – Number of blocks to be motion compensated in vertical direction.
- Parameters:msb – Indicates whether input data starts from the most significant byte (MSB) of the first word pointed by refblk_ptr or the least significant byte (LSB) of the same.
    — *0?LSB
    — *1 - MSB
- For luma compensation in interleaved cbYcr mode we set msb = 1 as Y is in MSB and inversely for chroma.
- For CbCrCbCr mode we set msb = 0 for Cb and vice-versa for Cr.

- Parameters:round_shift – Number of bits to shift down before output. For mcomp_A to 0.
- Parameters:round_type – Shifting type, whether rounding or truncation has to be performed while shifting.
  - *0?rounding
  - *1?truncation
- Set to 0 for rnd_control = 0
- *For mcomp_D rnd_control=1, do as suggested earlier.

Output Arguments

- `pred_blk_ptr` – Starting address of output block.

Returns

- `OMXResult`

Sadmultipleinterp_interleavedYCbYCr_DLx

Prototype

```
OMXResult omxVCM4P2_Sadmultipleinterp_interleavedYCbYCr_DLx(OMX_U8
    *pTargetBlk, OMX_U8 *pRefBlk, OMX_S16 *pOutput,
    OMXSadmultipleInterpParam *sparams);
```

Description

Performs halfpel motion estimation using interpolated data from M4P2_Allinterp. The target data has to be read from every other byte since it is in interleaved YCbYCr format.

Input Arguments

- `pTargetBlk` – Pointer to target array (linear). 8-byte aligned.
- `pRefBlk` – Pointer to reference array.1 byte aligned.
- `sparams` – Pointer to Parameters: for Sadmultiple.
- `OMXSadmultipleInterpParam` elements:
  - `block_width` – width of matching block.
  - `block_height` – height of matching block.
  - `target_width` – width of target array.

Output Arguments

- `pOutput` – Pointer to output array.

Returns

- `OMXResult`

SoSmultipleinterp_interleavedYCbYCr_DLx

Prototype

```
OMXResult omxVCM4P2_SoSmultipleinterp_interleavedYCbYCr_DLx(OMX_U8
    *pTargetBlk, OMX_U8 *pRefBlk, OMX_S16 *pOutput,
    OMXSoSmultipleInterpParam *sparams);
```

Description

Performs halfpel motion estimation based on sum of squares using interpolated data from `M4P2_Allinterp`. The target data has to be read from every other byte since it is in interleaved YCbYCr format.

Input Arguments

- `pTargetBlk` – Pointer to target array (linear). 8-byte aligned.
- `pRefBlk` – Pointer to reference array.1-byte aligned.
- `sparams` – Pointer to Parameters for SoSmultipleInterp.

`OMXSoSmultipleInterpParam` elements:

- `block_width` – width of matching block.
- `block_height` – height of matching block.
- `target_width` – width of target array.

Output Arguments

- `pOutput` – Pointer to output array.

Returns

- `OMXResult`

Allinterp_interleavedYCbYCr_DLx

Prototype

```
OMXResult omxVCM4P2_Allinterp_interleavedYCbYCr_DLx(OMX_U8 *pRefBlk, OMX_U8
    *pOutputBlk, OMXAllinterpParam *sparams);
```

Description

This function performs interpolation of luminance reference data. The output consists of three contiguous arrays corresponding to the three types of interpolated elements (type3?bi directional interpolation, type 1?horizontal interpolation, type 2?vertical interpolation). The target data has to be read from every other bytes since it is in interleaved YCbYCr fashion.

Input Arguments

- `pRefBlk` – Starting address of input block,1-byte aligned.
- `sparams` – Pointer to parameters for OMXAllinterp.

Sparams:

- `Block_width` – Width of each matching block.

- `Block_height` – Height of each matching block.
- `ref_width` – Reference buffer width.
- `Round_on_off` – Flag to turn off rounding.
  - 0 = rounding on.
  - 1 = rounding off.

Output Arguments

- `pOutputBlk` – Starting address of output block,8-byte aligned.

Returns

- `OMXResult`

allinterpblock_interleavedYCbYCr_DLx

Prototype

```
OMXResult omxVCM4P2_allinterpblock_interleavedYCbYCr_DLx(OMX_U8 *pRefBlk,
    OMX_U8 *pOutputBlk, OMXAllinterpBlockParam *sparams);
```

Description

This function performs interpolation of luminance reference data. The output is either interleaved with input or stored as three contiguous arrays corresponding to the three types of interpolated elements (bidirectional interpolation, horizontal interpolation, vertical interpolation). Reference data has to be read from every other byte since it is in interleaved YCbYCr fashion.

Input Arguments

- `pRefBlk` – Starting address of input block.1-byte aligned.
- `sparams` – Pointer to parameter for OMXAllinterpBlock.

Sparams:

- `Blk_width` – Width of each matching block.
- `Blk_height` – Height of each matching block.
- `ref_width` – Reference buffer width.
- `out_inter` – Indicates whether data is written interleaved with full pel data or as half-pel data only.
  - 1 = Halfpel Data interleaved with full pel data.
  - 0 = Hlafpel Data stored in three blocks corresponding to types b,c,d.
- `Round_on_off` – Flag to turn off rounding.
  - 0 = rounding on.
  - 1 = rounding off.

Output Arguments

- `pOutputBlk` – Starting address of output block.8-byte aligned.

Returns

- `OMXResult`

Sadmultiple_DLx

Prototype

```
OMXResult omxVCM4P2_Sadmultiple_DLx(OMX_U8 *pTargetBlk, OMX_U8 *pRefBlk,
   OMX_S16 *pOutput, OMXSadmultipleParam *sParams);
```

Description

Computes absolute differences between target block and blocks in reference array. The reference data and target data are each assumed to be a luminance-only block.

Input Arguments

- `pTargetBlk` – Pointer to target array (linear), 8-byte aligned.
- `pRefBlk` – Pointer to reference array.1-byte aligned.
- `sParams` – Pointer to Parameters for Sadmultiple.

`OMXSadmultipleParam` elements:

- `Block_width` – width of matching block.
- `Block_height` – height of matching block.
- `Target_width` – width of target array.
- `ref_width` – width of reference array.
- `step_horz` – Horizontal offset between matchings.
- `step_vert` – vertical offset between matchings.
- `nsteps_horz` – number of steps horizontally.
- `nsteps_vert` – number of steps vertically.
- `minmax_clear`.
  - — 0: retain previous min/max value.
  - — 1: clear previous min/max value.
- `Minmax`
  - — 0: Minimum SAD is calculated.
  - — 1: Maximum SAD is calculated.
- `id_mode` – This selects the type of id output.
  - — 0: block_count.
  - — 1: address
- `d_value` – This selects whether id or min/max value is output.
  - — 0: ID
  - — 1: min/max value
- `Thresh` – Threshold. When SAD is strictly below 2*threshold, command stops running, outputs current min/max or corresponding id/address.
- `center_block` – Indicates which block should have bias towards if there is a tie.
- `skip_field` – The bit value indicates which block to skip.

Output Arguments

- `pOutput` – Pointer to output array.

Returns

- `OMXResult`

Sadmultipleinterp_DLx

Prototype

```
OMXResult omxVCM4P2_Sadmultipleinterp_DLx(OMX_U8 *pTargetBlk, OMX_U8
    *pRefBlk, OMX_S16 *pOutput, OMXSadmultipleInterpParam *sparams);
```

Description

Performs halfpel motion estimation using interpolated data got from `M4P2_Allinterp`.

Input Arguments

- `pTargetBlk` – Pointer to target array (linear), 8-byte aligned.
- `pRefBlk` – Pointer to reference array. 1-byte aligned.
- `sparams` – Pointer to parameters for Sadmultiple.

`OMXSadmultipleInterpParam` elements:

- `block_width` – width of matching block.
- `block_height` – height of matching block.
- `target_width` – width of target array.

Output Arguments

- `pOutput` – Pointer to output array.

Returns

- `OMXResult`

DecodeMV_BVOP_Backward_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeMV_BVOP_Backward_DLx(const OMX_U8 ** ppBitStream,
    OMX_INT * pBitOffset, OMXVCMotionVector * pSrcDstMVB, OMX_INT
    fcodeBackward);
```

Description

Point-wise vector subtraction of 16-bit data type. This function subtracts the elements of one vector from the corresponding elements of a second vector.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer.

- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pSrcDstMVB` – pointer to the backward motion vector predictor.
- `fcodeBackward` – a code equal to `vop_fcode_backward` in MPEG-4 bit stream syntax so that it points to the current byte in the bit stream buffer.

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is decoded, so that it points to the current byte in the bit stream buffer.
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by ppBitStream.
- `pSrcDstMVB` – pointer to the backward motion vector of the current macroblock. The backward motion vector predictor should be reset to zero at the beginning of each macroblock row.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeMV_BVOP_Forward_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeMV_BVOP_Forward_DLx(const OMX_U8 ** ppBitStream,
    OMX_INT * pBitOffset, OMXVCMotionVector * pSrcDstMVF, OMX_INT
    fcodeForward);
```

Description

Decodes motion vectors of the macroblock in B-VOP forward mode. After decoding a macroblock of forward mode only, the forward predictor is set to the decoded forward vector.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7]
- `pSrcDstMVF` – pointer to the forward motion vector predictor.
- `fcodeForward` – a code equal to vop_fcode_forward in MPEG-4 bit stream syntax so that it points to the current byte in the bit stream buffer.

Output Arguments

- `ppBitStream` – ppBitStream is updated after the block is decoded, so that it points to the current byte in the bit stream buffer.
- `pBitOffset` – *pBitOffset is updated so that it points to the current bit position in the byte pointed by ppBitStream.
- `pSrcDstMVF` – pointer to the forward motion vector of the current macroblock. The forward motion vector predictor should be reset to zero at the beginning of each macroblock row.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeMV_BVOP_Interpolate_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeMV_BVOP_Interpolate_DLx(const OMX_U8 **
   ppBitStream, OMX_INT * pBitOffset, OMXVCMotionVector * pSrcDstMVF,
   OMXVCMotionVector * pSrcDstMVB, OMX_INT fcodeForward, OMX_INT
   fcodeBackward);
```

Description

Decodes motion vectors of the macroblock in B-VOP interpolate mode. After decoding a macroblock of interpolate mode, both the forward and backward predictor are updated separately with the decoded vectors of the same type (forward/backward).

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pSrcDstMVF` – pointer to the forward motion vector predictor. The forward motion vector predictor should be reset to zero at the beginning of each macroblock row.
- `pSrcDstMVB` – pointer to the backward motion vector predictor. The backward motion vector predictor should be reset to zero at the beginning of each macroblock row.
- `fcodeForward` – a code equal to `vop_fcode_forward` in MPEG-4 bit stream syntax.
- `fcodeBackward` – a code equal to `vop_fcode_backward` in MPEG-4 bit stream syntax.

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is decoded, so that it points to the current byte in the bit stream buffer.
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by `*ppBitStream`.
- `pSrcDstMVF` – pointer to the forward motion vector of the current macroblock.
- `pSrcDstMVB` – pointer to the backward motion vector of the current macroblock.

Returns

- Standard OMXResult. See enumeration for possible result codes.

DecodeMV_BVOP_Direct_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeMV_BVOP_Direct_DLx(const OMX_U8 ** ppBitStream,
   OMX_INT * pBitOffset, const OMXVCMotionVector * pSrcMV,
   OMXVCMotionVector * pDstMVF, OMXVCMotionVector * pDstMVB, OMX_U8 *
   pTranspSrcMB, OMX_INT TRB, OMX_INT TRD);
```

Description

Decodes motion vector(s) of the macroblock in B-VOP using direct mode.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte in the bit stream buffer.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pSrcMV` – pointer to the motion vector buffer of the co-located macroblock in the most recently decoded I- or P-VOP.
- `pTranspSrcMB` – pointer to the transparent status buffer of the co-located macroblock.
- `TRB` – the difference in temporal reference of the B-VOP and the previous reference VOP.
- `TRD` – the difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP.

Output Arguments

- `ppBitStream` – `*ppBitStream` is updated after the block is decoded, so that it points to the current byte in the bit stream buffer.
- `pBitOffset` – `*pBitOffset` is updated so that it points to the current bit position in the byte pointed by `*ppBitStream`.
- `pDstMVF` – pointer to the forward motion vector buffer of the current macroblock that contains decoded forward motion vector.
- `pDstMVB` – pointer to the backward motion vector buffer of the current macroblock which contains decoded backward motion vector.

Returns

Standard OMXResult. See enumeration for possible result codes.

DecodeMV_BVOP_DirectSkip_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeMV_BVOP_DirectSkip_DLx(const OMXVCMotionVector *
   pSrcMV, OMXVCMotionVector * pDstMVF, OMXVCMotionVector * pDstMVB, OMX_U8
   * pTranspSrcMB, OMX_INT TRB, OMX_INT TRD);
```

Description

Decodes motion vector(s) of the macroblock in B-VOP using direct mode when the current macroblock is skipped.

Input Arguments

- `pSrcMV` – pointer to the motion vector buffer of the co-located macroblock in the most recently decoded I- or P-VOP.
- `pTranspSrcMB` – pointer to the transparent status buffer of the co-located macroblock.
- `TRB` – the difference in temporal reference of the B-VOP and the previous reference VOP.
- `TRD` – the difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP.

Output Arguments

- `pDstMVF` – pointer to the forward motion vector buffer of the current macroblock which contains decoded forward motion vector.
- `out]pDstMVB` – pointer to the backward motion vector buffer of the current macroblock which contains decoded backward motion vector.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeCAEIntraH_U8_DLx

DecodeCAEIntraV_U8_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeCAEIntraH_U8_DLx(const OMX_U8 ** ppBitStream,
   OMX_INT *pBitOffset, OMX_U8 * pBinarySrcDst, OMX_INT step, OMX_INT
   blocksize);
OMXResult omxVCM4P2_DecodeCAEIntraV_U8_DLx(const OMX_U8 ** ppBitStream,
   OMX_INT *pBitOffset, OMX_U8 * pBinarySrcDst, OMX_INT step, OMX_INT
   blocksize);
```

Description

Performs Context Arithmetic Code decoding in intra macroblock. H indicates scan type is horizontal. V indicates scan type is vertical. Convert ratio is supported in these functions.

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte from which the intra block starts.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pBinarySrcDst` – pointer to the Source-Dest Binary macroblock the left and top border should be loaded before.
- `step` – width of source-dest binary plane, in bytes.
- `blocksize` – macroblock size, if convert ratio take effects, it means subsampled macro block size.

Output Arguments

- `ppBitStream` – pointer to the pointer to the current byte from which the intra block starts.

- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pBinarySrcDst` – pointer to the Source-Dest Binary macroblock the left and top border should be loaded before.

Returns

Standard OMXResult result. See enumeration for possible result codes.

DecodeCAEInterH_U8_DLx

DecodeCAEInterV_U8_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeCAEInterH_U8_DLx(const OMX_U8 ** ppBitStream,
    OMX_INT * pBitOffset, const OMX_U8 * pBinarySrcPred, OMX_INT offsetPred,
    OMX_U8 * pBinarySrcDst, OMX_INT step, OMX_INT blocksize);
```

```
OMXResult omxVCM4P2_DecodeCAEInterV_U8_DLx(const OMX_U8 ** ppBitStream,
    OMX_INT * pBitOffset, const OMX_U8 * pBinarySrcPred, OMX_INT offsetPred,
    OMX_U8 * pBinarySrcDst, OMX_INT step, OMX_INT blocksize);
```

Description

Performs Context Arithmetic Code decoding in inter macroblock. H indicates scan type is horizontal. V indicates scan type is vertical. Convert ratio is supported in these functions.

---

2 ***Note:*** *This function multiplies the elements of one vector to the corresponding elements of a second vector.*

Input Arguments

- `ppBitStream` – pointer to the pointer to the current byte from which the intra block starts.
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pBinarySrcPred` – pointer to the related macroblock in the reference binary plane. The left and top border should be loaded before. Pointer points to the top-left corner of this macro block, not the extended zone.
- `pBinarySrcDst` – pointer to the Source-Dest Binary macroblock. The left and top border should be loaded before.
- `offsetPred` – the bit position of first pixel in reference macroblock, valid within Bits 0 to 7. Where:
  - — MSB=zero (0)
  - — LSB=seven (7)
- `step` – width of source-dest binary plane and reference plane, in byte. If blocksize not equals to 16, it indicates binary buffer step.
- `blocksize` – macroblock size, if convert ratio take effects, it means subsampled macro block size.

Output Arguments

- `ppBitStream` – pointer to the pointer to the current byte from which the intra block starts
- `pBitOffset` – pointer to the bit position in the byte pointed to by `*ppBitStream`. `*pBitOffset` is valid within [0-7].
- `pBinarySrcDst` – pointer to the Source-Dest Binary macroblock the left and top border should be loaded before.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeMVS_DLx

Prototype

```
OMXResult omxVCM4P2_DecodeMVS_DLx(const OMX_U8 **ppBitStream, OMX_INT
    *pBitOffset, OMXVCMotionVector * pSrcDstMVS, const OMX_U8 * pSrcBABMode,
    OMX_INT stepBABMode, const OMXVCMotionVector * pSrcMVLeftMB, const
    OMXVCMotionVector * pSrcMVUpperMB, const OMXVCMotionVector *
    pSrcMVUpperRightMB, const OMX_U8 * pTranspLeftMB, const OMX_U8 *
    pTranspUpperMB, const OMX_U8 * pTranspUpperRightMB, OMX_INT predFlag);
```

Description

Decode MVs (Motion Vector of shape) according to the spec.

Input Arguments

- `ppBitStream` – Pointer to the pointer to the current byte in the bit stream buffer.
- `pBitOffset` – Pointer to the bit position in the byte pointed by `*ppBitStream`. Valid within 0 to 7.
- `pSrcDstMVS` – Pointer to the shape motion vector buffer of the current BAB.
- `pSrcBABMode` – Pointer to the BAB mode buffer of current BAB, which stored in the BAB mode plane.
- `stepBABMode` – The width of the BAB mode plane.
- `pSrcMVLeftMB` – Pointers to the motion vector buffers of the macroblocks spacially at the left, upper and upper-right side of the current macroblock respectively.
- `pSrcMVUpperMB` – Pointers to the motion vector buffers of the macroblocks spacially at the left, upper and upper-right side of the current macroblock respectively.
- `pSrcMVUpperRightMB` – Pointers to the motion vector buffers of the macroblocks spacially at the left, upper and upper-right side of the current macroblock respectively.
- `pTranspLeftMB` – Pointers to the transparent status buffers of the macroblocks, spacially at the left, upper, and upper-right side of, and the current macroblock respectively.
- `pTranspUpperMB` – Pointers to the transparent status buffers of the macroblocks, spacially at the left, upper, and upper-right side of, and the current macroblock respectively.
- `pTranspUpperRightMB` – Pointers to the transparent status buffers of the macroblocks, spacially at the left, upper, and upper-right side of, and the current macroblock respectively.
- `predFlag` – The flag will be set zero, while the current VOP is BVOP or the current VOL is shape only mode; else, the flag is nonzero.

Output Arguments

- `pSrcDstMVS` – Pointer to the decoded motion vector of shape.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

BlockMatchSOS_Integer_16x16_DLx

Prototype

```
OMXResult omxVCM4P2_BlockMatchSOS_Integer_16x16_DLx(OMX_U8 *pSrcRefBuf,
    OMX_INT refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf,
    OMXVCM4P2Coordinate * *pCurrPointPos, OMXVCMotionVector *pSrcPreMV,
    OMX_INT *pSrcPreSOS, OMX_INT searchRange, void *pMESpec,
    OMXVCMotionVector *pDstMV, OMX_INT *pDstSOS);
```

Description

Performs a 16x16 block search based on minimum sum of squared error (SOS); returns integer resolution motion vector and associated minimum SOS. Both the input and output motion vectors are represented using half-pixel units, and therefore a shift left or right by 1 bit may be required, respectively, to match the input or output MVs with other functions that either generate output MVs or expect input MVs represented using integer pixel units.

Input Arguments

- `pSrcRefBuf` – pointer to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- `refWidth` – width of the reference plane
- `pRefRect` – pointer to the valid reference plane rectangle; coordinates are specified relative to the image origin. Rectangle boundaries may extend beyond image boundaries if the image has been padded.
- `pSrcCurrBuf` – pointer to the current macroblock extracted from original plane (linear array, 256 entries); must be aligned on an 8-byte boundary.
- `pCurrPointPos` – position of the current macroblock in the current plane
- `pSrcPreMV` – pointer to predicted motion vector; NULL indicates no predicted MV
- `pSrcPreSOS` – pointer to SOS associated with the predicted MV (referenced by `pSrcPreMV`)
- `searchRange` – search range for 16X16 integer block; the search range is the same in all directions. It is in inclusive of the boundary and specified in terms of integer pixel units.
- `pMESpec` – vendor-specific motion estimation specification structure; must have been allocated and then initialized using `omxVCM4P2_MEInit` prior to calling the block matching function.

Output Arguments

- `pDstMV` – pointer to estimated MV
- `pDstSOS` – pointer to minimum SOS

Returns

- `OMX_StsNoErr` – no error

- `OMX_StsBadArgErr` – bad arguments

BlockMatchSOS_Integer_8x8_DLx

Prototype

```
OMXResult omxVCM4P2_BlockMatchSOS_Integer_8x8_DLx(OMX_U8 *pSrcRefBuf,
   OMX_INT refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf,
   OMXVCM4P2Coordinate * *pCurrPointPos, OMXVCMotionVector *pSrcPreMV,
   OMX_INT *pSrcPreSOS, OMX_INT searchRange, void *pMESpec,
   OMXVCMotionVector *pDstMV, OMX_INT *pDstSOS);
```

Description

Performs an 8x8 block search based on minimum sum of squared error (SOS); returns integer resolution motion vector and associated minimum SOS. Both the input and output motion vectors are represented using half-pixel units, and therefore a shift left or right by 1 bit may be required, respectively, to match the input or output MVs with other functions that either generate output MVs or expect input MVs represented using integer pixel units.

Input Arguments

- `pSrcRefBuf` – pointer to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- `refWidth` – width of the reference plane
- `pRefRect` – pointer to the valid reference plane rectangle; coordinates are specified relative to the image origin. Rectangle boundaries may extend beyond image boundaries if the image has been padded.
- `pSrcCurrBuf` – pointer to the current macroblock extracted from original plane (linear array, 256 entries); must be aligned on an 8-byte boundary. The step between lines of the 8x8 block is 16 bytes.
- `pCurrPointPos` – position of the current macroblock in the current plane
- `pSrcPreMV` – pointer to predicted motion vector; NULL indicates no predicted MV
- `pSrcPreSOS` – pointer to SOS associated with the predicted MV (referenced by `pSrcPreMV`)
- `searchRange` – search range for 8x8 integer block; the search range is the same in all directions. It is in inclusive of the boundary and specified in terms of integer pixel units.
- `pMESpec` – vendor-specific motion estimation specification structure; must have been allocated and then initialized using `omxVCM4P2_MEInit` prior to calling the block matching function.

Output Arguments

- `pDstMV` – pointer to estimated MV
- `pDstSOS` – pointer to minimum SOS

BlockMatchSOS_Half_16x16_DLx

Prototype

```
OMXResult omxVCM4P2_BlockMatchSOS_Half_16x16_DLx(OMX_U8 *pSrcRefBuf, OMX_INT
    refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate
    *pSearchPointRefPos, OMX_INT rndVal, OMXVCMotionVector *pSrcDstMV,
    OMX_INT *pDstSOS);
```

Description

Performs an 16x16 block search based on minimum sum of squared error (SOS); returns half-pixel resolution motion vector and associated minimum SOS. This function estimates the half-pixel motion vector by interpolating the integer resolution motion vector referenced by the input parameter `pSrcDstMV`, i.e., the initial integer MV is generated externally. The input parameters `pSrcRefBuf` and `pSearchPointRefPos` should be shifted by the winning MV of 16x16 integer search prior tocalling `BlockMatchSOS_Half_16x16_DLx`. The function `BlockMatchSOS_Integer_16x16_DLx` may be used for integer motion estimation.

Input Arguments

- `pSrcRefBuf` – pointer to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- `RefWidth` – width of the reference plane
- `pRefRect` – reference plane valid region rectangle
- `pSrcCurrBuf` – pointer to the current macroblock extracted from original plane (linear array, 256 entries); must be aligned on an 8-byte boundary.
- `pSearchPointRefPos` – position of the starting point for half pixel search (specified in terms of integer pixel units) in the reference plane.
- `rndVal` – rounding control bit for half pixel motion estimation; 0=rounding control disabled; 1=rounding control enabled
- `pSrcDstMV` – pointer to the initial MV estimate; typically generated during a prior 16X16 integer search; specified in terms of half-pixel units.

Output Arguments

- `pDstMV` – pointer to estimated MV
- `pDstSOS` – pointer to minimum SOS

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

BlockMatchSOS_Half_8x8_DLx

Prototype

```
OMXResult omxVCM4P2_BlockMatchSOS_Half_8x8_DLx(OMX_U8 *pSrcRefBuf, OMX_INT
   refWidth, OMXRect *pRefRect, OMX_U8 *pSrcCurrBuf, OMXVCM4P2Coordinate
   *pSearchPointRefPos, OMX_INT rndVal, OMXVCMotionVector *pSrcDstMV,
   OMX_INT *pDstSOS);
```

Description

Performs an 8x8 block search based on minimum sum of squared error (SOS); returns half-pixel resolution motion vector and associated minimum SOS. This function estimates the half-pixel motion vector by interpolating the integer resolution motion vector referenced by the input parameter `pSrcDstMV`, i.e., the initial integer MV is generated externally. The input parameters `pSrcRefBuf` and `pSearchPointRefPos` should be shifted by the winning MV of 8x8 integer search prior tocalling `BlockMatchSOS_Half_8x8_DLx`. The function `BlockMatchSOS_Integer_8x8_DLx` may be used for integer motion estimation.

Input Arguments

- `pSrcRefBuf` – `pointer` to the reference Y plane; points to the reference MB that corresponds to the location of the current macroblock in the current plane.
- `refWidth` – width of the reference plane
- `pRefRect` – reference plane valid region rectangle
- `pSrcCurrBuf` – pointer to the current macroblock extracted from original plane (linear array, 256 entries); must be aligned on an 8-byte boundary. The step between lines of the 8x8 block is 16 bytes.
- `pSearchPointRefPos` – position of the starting point for half pixel search (specified in terms of integer pixel units) in the reference plane.
- `rndVal` – rounding control bit for half pixel motion estimation; 0=disable rounding control, 1=enable rounding control.
- `pSrcDstMV` – pointer to the initial MV estimate; typically generated during a prior 8x8 integer search, specified in terms of half-pixel units.

Output Arguments

- `pSrcDstMV` – pointer to estimated MV
- `pDstSOS` – pointer to minimum SOS

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

PadCurrent_16x16_U8_I_DLx

PadCurrent_8x8_U8_I_DLx

Prototype

```
OMXResult omxVCM4P2_PadCurrent_16x16_U8_I_DLx (const OMX_U8 * pSrcBAB,
    OMX_INT stepBinary, OMX_U8 * pSrcDst, OMX_INT stepTexture);
OMXResult omxVCM4P2_PadCurrent_8x8_U8_I_DLx (const OMX_U8 * pSrcBAB, OMX_INT
    stepTexture, OMX_U8 * pSrcDst);
```

Description

Performs horizontal and vertical repetitive padding process on luminance/alpha macroblock or chrominance block. The horizontal and vertical repetitive padding processes are specified in subclause 7.6.1.1 and 7.6.1.2 of *ISO/IEC 14496-2* respectively.

Input Arguments

- `pSrcDst` – pointer to the block to be padded
- `stepTexture` – width of the source texture (Luminance, Chrominance or Grayscale alpha) plane (numbered with pixel)
- `stepBinary` – width of the source binary alpha plane (for 16X16 version) or source binary alpha buffer (for 8X8 version) (numbered with byte)
- `pSrcBAB` – pointer to the binary alpha plane (for 16X16 version) or binary alpha block buffer (for 8X8 version). In 8X8 version, the buffer contains 32 bytes (256 bits) for 16 by 16 luminance or alpha block, or 8 bytes (64 bits) for 8 by 8 chrominance block

---

2 ***Note:*** *For chrominance components, the BAB is generated by subsampling the shape block of the corresponding luminance component.*

Output Arguments

`pSrcDst` – pointer to the padded block

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - At least one of the pointers is NULL: `pSrcDst` or `pSrcBAB`.
    or
  - In 16 by 16 case, at least one of below case: `stepTexture` < 16, `stepBinary` < 2, `stepTexture` is not 4 multiple.
    or
  - In 8 by 8 case, at least one of below case: `stepTexture` < 8, `stepTexture` is not 4 multiple.
    or

— `pSrcDst` is not 32-bit aligned.
or

— All the elements of current BAB are zero.

PadMBHorizontal_U8_DLx

Prototype

```
OMXResult omxVCM4P2_PadMBHorizontal_U8_DLx (const OMX_U8 * pSrcY, const
    OMX_U8 * pSrcCb, const OMX_U8 * pSrcCr, const OMX_U8 * pSrcA, OMX_U8 *
    pDstY, OMX_U8 * pDstCb, OMX_U8 * pDstCr, OMX_U8 * pDstA, OMX_INT stepYA,
    OMX_INT stepCbCr);
```

Description

Performs horizontal extended padding process on exterior macroblock, which includes luminance, chrominance and alpha (if available) blocks, immediately next to the boundary macroblock.

Note: *The MB version pads all blocks of luminance, chrominance, and alpha (if it exists) in one MB, while the 16x16 version is used to pad only four luminance or alpha blocks, and 8x8 version to pad one chrominance (Cb or Cr) block.*

Input Arguments

- `stepYA` – width of the luminance or alpha planes. (numbered with pixel)
- `stepCbCr` – width of the Chrominance planes. (numbered with pixel)
- `pSrcY` – pointer to one of the vertical border of the boundary luminance blocks that are chosen to pad the exterior macroblock
- `pSrcCb` –pointer to one of the vertical border of the boundary Cb block that is chosen to pad the exterior macroblock
- `pSrcCr` – pointer to one of the vertical border of the boundary Cr block that is chosen to pad the exterior macroblock
- `pSrcA` – pointer to one of the horizontal border of the boundary alpha blocks that are chosen to pad the exterior macroblock. If `pSrcA` equals to NULL, then no alpha plane is available. Otherwise, the alpha plane should be padded.

Output Arguments

- `pDstY` – pointer to the padded exterior luminance blocks
- `pDstCb` – pointer to the padded exterior Cb block
- `pDstCr` – pointer to the padded exterior Cr block
- `pDstA` – pointer to the padded exterior alpha blocks

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments

— At least one of the following pointers is NULL: `pSrcY`, `pSrcCb`, `pSrcCr`, `pDstY`, `pDstCb`, `pDstCr`.
or

— If `pSrcA` != NULL, `pDstA` = NULL or `pDstA` is not 32-bit aligned.
or

— At least one of `pDstY`, `pDstCb`, or `pDstCr` is not 32-bit aligned.
or

— At least one of the following conditions is true:

– `stepYA` < 16

– `stepCbCr` < 8

– `stepYA` or `stepCbCr` is not a multiple of 4

PadMBVertical_U8_DLx

Prototype

```
OMXResult omxVCM4P2_PadMBVertical_U8_DLx (const OMX_U8 * pSrcY, const OMX_U8
    * pSrcCb, const OMX_U8 * pSrcCr, const OMX_U8 * pSrcA, OMX_U8 * pDstY,
    OMX_U8 * pDstCb, OMX_U8 * pDstCr, OMX_U8 * pDstA, OMX_INT stepYA, OMX_INT
    stepCbCr);
```

Description

Performs vertical extended padding process on exterior macroblock, which includes luminance, chrominance and alpha (if available) blocks, immediately next to the boundary macroblock.

---

2 ***Note:*** *The MB version pads all blocks of luminance, chrominance, and alpha (if the exist) in one MB, while the 16X16 version could be used to pad only four luminance or alpha blocks, and 8X8 version to pad one chrominance (Cb or Cr) block.*

Input Arguments

- `stepYA` – width of the Luminance and/or alpha planes
- `stepCbCr` – width of the Chrominance planes
- `pSrcY` – pointer to one of the horizontal border of the boundary luminance blocks that are chosen to pad the exterior macroblock
- `pSrcCb` – pointer to one of the horizontal border of the boundary Cb block that is chosen to pad the exterior macroblock
- `pSrcCr` – pointer to one of the horizontal border of the boundary Cr block that is chosen to pad the exterior macroblock
- `pSrcA` – pointer to one of the horizontal border of the boundary alpha blocks that are chosen to pad the exterior macroblock. If `pSrcA` equals to NULL, then no alpha plane is available. Otherwise, the alpha plane should be padded in MB version.

Output Arguments

- `pDstY` – pointer to the padded exterior luminance blocks
- `pDstCb` – pointer to the padded exterior Cb block
- `pDstCr` – pointer to the padded exterior Cr block
- `pDstA` – pointer to the padded exterior alpha blocks

Returns

- `OMX_StsNoErr` – no error
- `OMX_StsBadArgErr` – bad arguments
  - — At least one of the following pointers is NULL:

1. `pSrcY`, `pSrcCb`, `pSrcCr`, `pDstY`, `pDstCb`, `pDstCr`
   - — At least one of below case: stepYA < 16, stepCbCr < 8, stepYA or stepCbCr is not a multiple of 4.
   - — At least one of the following is not 32-bit aligned:
2. `pSrcY`, `pSrcCb`, `pSrcCr`, `pDstY`, `pDstCb`, `pDstCr`
   - — If `pSrcA` is not NULL, `pSrcA` is not 32-bit aligned, `pDstA` is NULL or `pDstA` is not 32-bit aligned.

PadMBGray_U8_DLx

Prototype

```
OMXResult omxVCM4P2_PadMBGray_U8_DLx (OMX_U8 grayVal, OMX_U8 * pDstY, OMX_U8
    * pDstCb, OMX_U8 * pDstCr, OMX_U8 * pDstA, OMX_INT stepYA, OMX_INT
    stepCbCr);
```

---

2 ***Note:*** *The MB version pads all blocks of luminance, chrominance and alpha (if exist) in one MB, while 16X16 version could be used to pad only 4 luminance or alpha blocks, and 8X8 version to pad one chrominance (Cb or Cr) block.*

Description

Fills gray value in exterior macroblock (includes luminance, chrominance and alpha (if available) blocks) that is not located next to any boundary macroblock.

Input Arguments

- `grayVal` – the gray value to fill the exterior macroblock/block. It should be set to $2^{bits\_per\_pixel} - 1$, where bits_per_pixel = 8 here.
- `stepYA` – width of the Luminance and/or alpha planes.(numbered with pixel)
- `stepCbCr` – width of the Chrominance planes.

Output Arguments

- `pDstY` – pointer to the padded exterior luminance blocks. `pDstY` should be 32-bit aligned.

- pDstCb – pointer to the padded exterior Kb block. DstCb should be 32-bit aligned.
- pDstCr – pointer to the padded exterior Cr block. pDstCr should be 32-bit aligned.
- pDstA – pointer to the padded exterior alpha blocks. If pDstA equals to NULL, then no alpha plane is available. Otherwise, the alpha plane should be padded in MB version. pDstA should be 32-bit aligned.

Returns

- OMX_StsNoErr – no error
- OMX_StsBadArgErr – bad arguments
  - — At least one of the following pointers is NULL: pDstY, pDstCb, pDstCr.
    or
  - — At least one of below case: stepYA < 16, stepCbCr < 8, grayvalue <= 0, stepYA or stepCbCr is not a multiple of 4.
    or

—At least one of pDstY, pDstCb, pDstCr not 32-bit aligned, If pDstA != NULL, pDstA not 32-bit aligned.

Functions for the Video Coding H.264 Subdomain (omxVCM4P10)

DequantTransformResidualFromPair_C1_DLx

Prototype

```
OMXResult omxVCM4P10_DequantTransformResidualFromPair_C1_DLx(OMX_U8 **ppSrc,
    OMX_S16 *pDst,OMX_INT QP,OMX_S16* pDC,int AC );
```

Description

Reconstruct the 4x4 residual block from coefficient-position pair buffer, perform dequantisation and integer inverse transformation for 4x4 block of residuals and update the pair buffer pointer to next non-empty block.

Input Arguments

- ppSrc – double pointer to residual coefficient-position pair buffer output by CALVC decoding
- pDC – pointer to the DC coefficient of this block; NULL if it doesn't exist
- QP – quantization parameter
- AC – flag indicating if at least one non-zero coefficient exists

Output Arguments

- pDst – pointer to the reconstructed 4x4 block

Returns

- Standard OMXError result. See enumeration for possible result codes.

Functions for the Image Coding JPEG Sub-Domain (omxICJP)

DCTQuantFwd_Multiple_S16_I_DLx

Prototype

```
OMXResult omxICJP_DCTQuantFwd_Multiple_S16_I_DLx (OMX_S16* pSrcDst, OMX_INT
    nBlocks, const OMX_U16 *pQuantFwdTable);
```

Description

This function implements forward DCT with quantization for the 8-bit image data. It processes multiple adjacent blocks (8x8).The blocks are assumed to be part of a planarized buffer. This function needs to be called separately for luma and chroma buffers with the respective quantization table. The output matrix is the transpose of the explicit result. As a result, the Huffman coding functions in this library handle transpose as well.

Input Arguments

- `nBlocks` – the number of 8x8 blocks to be processed.
- `pQuantFwdTable` – identifies the quantization table that was generated from " `DCTQuantFwdTableInit`". The table length is 64. This start address must be 8-byte aligned.

In-Out Arguments

- `[in,out]pSrcDst` – Identifies coefficient block(8x8) buffer for in-place processing. This start address must be 8-byte aligned. The input components are bounded on the interval [-128, 127] within a signed 16-bit container. To achieve better performance, the output 8x8 matrix is the transpose of the explicit result. This transpose will be handled in Huffman encoding. Each 8x8 block in the buffer is stored as 64 entries (16-bit) linearly in a buffer, and the multiple blocks to be processed must be adjacent.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DCTQuantInv_Multiple_S16_I_DLx

Prototype

```
OMXResult omxICJP_DCTQuantInv_Multiple_S16_I (OMX_S16* pSrcDst, OMX_INT
    nBlocks, const OMX_U16 *pQuantInvTable);
```

Description

This function implements inverse DCT with dequantization for 8-bit image data. It processes multiple blocks (each 8x8). The blocks are assumed to be part of a planarized buffer. This function needs to be called separately for luma and chroma buffers with the respective quantization table. The start address of `pQuantRawTable` and `pQuantInvTable` must be 8-byte aligned.

Input Arguments

- `nBlocks` – the number of 8x8 blocks to be processed.

- `pQuantInvTable` – identifies the quantization table which was generated from "`DCTQuantInvTableInit_JPEG_U8_U16`". The table length is 64 entries by 16-bit. The start address must be 8-byte aligned.

In-Out Arguments

- `pSrcDst` – identifies input coefficient block(8x8) buffer for in-place processing. The start address must be 8-byte aligned.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

EncodeHuffman8x8_DCFirst_S16U1_C1_DLx

Prototype

```
OMXResult omxICJP_EncodeHuffman8x8_DCFirst_S16U1_C1_DLx(const OMX_S16* pSrc,
   OMX_U8* pDst, OMX_INT dstBytesLen, OMX_INT* pDstCurrPos, OMX_S16*
   pDCPred, OMX_INT al, const OMXICJPHuffmanEncodeSpec* pDCHuffTable,
   OMX_sEncodeHuffmanState* pHuffState, OMX_INT bFlushState);
```

Description

This function implements Huffman DC encoding of the first scan for progressive mode. The encoding procedure conforms to G.1.2 of T.81.

Input Arguments

- `pSrc` – pointer to the source coefficient data in the 8x8 block.
- `dstBytesLen` – length of bit stream buffer
- `pDstCurrPos` – pointer to the current byte position in the bit stream buffer.
- `pDCPred` – pointer to the DC value of the previous 8x8 block.
- `al` – low bit position of successive approximation.
- `pDCHuffTable` – pointer to the structure of Huffman encoding table for DC. The table must be retrieved by calling `EncodeHuffmanSpecInit_U8`.
- `pHuffState` – pointer to the structure of Huffman encoding state.
- `bFlushState` – set to 1 for the last 8x8 block to flush any pending bits in the state.

Output Arguments

- `pDst` – pointer to encoded Huffman stream data.
- `pDstCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pDCPred` – pointer to the DC value for next prediction.
- `pHuffState` – pointer to the updated structure of Huffman encoding state.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

EncodeHuffman8x8_DCRefine_S16U1_C1_DLx

Prototype

```
OMXResult omxICJP_EncodeHuffman8x8_DCRefine_S16U1_C1_DLx(const OMX_S16*
    pSrc, OMX_U8* pDst, OMX_INT dstBytesLen, OMX_INT* pDstCurrPos, OMX_INT
    al, OMX_sEncodeHuffmanState* pHuffState, OMX_INT bFlushState);
```

Description

This function implements Huffman DC encoding of the refined scans for progressive mode. The encoding procedure conforms to G.1.2 of T.81.

Input Arguments

- `pSrc` – pointer to the source coefficient data in the 8x8 block.
- `dstBytesLen` – length of bit stream buffer.
- `pDstCurrPos` – pointer to the current byte position in the bit stream buffer.
- `al` – low bit position of successive approximation.
- `pHuffState` – pointer to the structure of Huffman encoding state.
- `bFlushState` – set to 1 for the last 8x8 block to flush any pending bits in the state.

Output Arguments

- `pDst` – pointer to encoded huffman stream data.
- `pDstCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pHuffState` – pointer to the updated structure of Huffman encoding state.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

EncodeHuffman8x8_ACFirst_S16U1_C1_DLx

Prototype

```
OMXResult omxICJP_EncodeHuffman8x8_ACFirst_S16U1_C1_DLx (const OMX_S16*
    pSrc, OMX_U8* pDst, OMX_INT dstBytesLen, OMX_INT* pDstCurrPos, OMX_INT
    ss, OMX_INT se, OMX_INT al, const OMXICJPHuffmanEncodeSpec*
    pACHuffTable, OMX_sEncodeHuffmanState* pHuffState, OMX_INT bFlushState);
```

Description

This function implements Huffman AC encoding of the first scans for progressive mode. The encoding procedure conforms to G.1.2 of T.81.

Input Arguments

- `pSrc` – pointer to the source coefficient data in the 8x8 block.
- `dstBytesLen` – length of bit stream buffer.
- `pDstCurrPos` – pointer to the current byte position in the bit stream buffer.
- `ss` – start index of spectral selection.

- `se` – end index of spectral selection.
- `al` – low bit position of successive approximation.
- `pACHuffTable` – pointer to the structure of Huffman encoding table for AC. The table must be retrieved by calling `EncodeHuffmanSpecInit_U8`.
- `pHuffState` – pointer to the structure of Huffman encoding state.
- `bFlushState` – set to 1 for the last 8x8 block to flush any pending bits in the state.

Output Arguments

- `pDst` – pointer to encoded huffman stream data.
- `pDstCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pHuffState` – pointer to the updated structure of Huffman encoding state.

Returns

Standard OMXResult result. See enumeration for possible result codes.

EncodeHuffman8x8_ACRefine_S16U1_C1_DLx

Prototype

```
OMXResult omxICJP_EncodeHuffman8x8_ACRefine_S16U1_C1_DLx(const OMX_S16*
   pSrc, OMX_U8* pDst, OMX_INT dstBytesLen, OMX_INT* pDstCurrPos, OMX_INT
   ss, OMX_INT se, OMX_INT al, const OMXICJPHuffmanEncodeSpec*
   pACHuffTable, OMX_sEncodeHuffmanState *pHuffState, OMX_INT bFlushState);
```

Description

This function implements Huffman AC encoding of the refined scans for progressive mode. The encoding procedure conforms to G.1.2 of T.81.

Input Arguments

- `pSrc` – pointer to the source coefficient data in the 8x8 block.
- `dstBytesLen` – length of bit stream buffer.
- `pDstCurrPos` – pointer to the current byte position in the bit stream buffer.
- `ss` – start index of spectral selection.
- `se` – end index of spectral selection.
- `al` – low bit position of successive approximation.
- `pACHuffTable` – pointer to the structure of Huffman encoding table for AC. The table must be retrieved by calling `EncodeHuffmanSpecInit_U8`.
- `pHuffState` – pointer to the structure of Huffman encoding state.
- `bFlushState` – set to 1 for the last 8x8 block to flush any pending bits in the state.

Output Arguments

- `pDst` – pointer to encoded Huffman stream data.
- `pDstCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pHuffState` – pointer to the updated structure of Huffman encoding state.

Returns

Standard OMXResult result. See enumeration for possible result codes.

DecodeHuffman8x8_DCFirst_U1S16_C1_DLx

Prototype

```
OMXResult omxICJP_DecodeHuffman8x8_DCFirst_U1S16_C1_DLx (const OMX_U8* pSrc,
   OMX_INT srcBytesLen, OMX_INT* pSrcCurrPos, OMX_S16* pDst, OMX_S16*
   pDCPred, OMX_INT* pMarker, OMX_INT al, const OMXICJPHuffmanDecodeSpec*
   pDCHuffTable, OMX_sDecodeHuffmanState* pHuffState);
```

Description

This function implements Huffman DC decoding of the first scan for progressive mode. The decoding procedure conforms to G.2 of T.81.

Input Arguments

- `pSrc` – pointer to the input bit stream buffer.
- `srcBytesLen` – length of the bit stream.
- `pSrcCurrPos` – pointer to the current byte position in the bit stream buffer.
- `pDCPred` – pointer to the DC value of the previous 8x8 block.
- `pMarker` – pointer to the marker.
- `al` – low bit position of successive approximation.
- `pDCHuffTable` – pointer to the structure of the Huffman decoding table for DC. The table must be retrieved by calling `OMXICJPHuffmanDecodeSpecInit_JPEG_8u`.
- `pHuffState` – pointer to the structure of Huffman decoding state.

Output Arguments

- `pSrcCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pDst` – pointer to the output block of decoded coefficient data.
- `pDCPred` – pointer to the DC value for next prediction.
- `pMarker` – pointer to the marker.
- `pHuffState` – pointer to the updated structure of Huffman decoding state.

In-Out Arguments

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeHuffman8x8_DCRefine_U1S16_C1_DLx

Prototype

```
OMXResult omxICJP_DecodeHuffman8x8_DCRefine_U1S16_C1_DLx(const OMX_U8* pSrc,
    OMX_INT srcBytesLen, OMX_INT* pSrcCurrPos, OMX_S16* pDst, OMX_INT*
    pMarker, OMX_INT al, OMX_sDecodeHuffmanState* pHuffState);
```

Description

This function implements Huffman DC decoding of the refined scans for progressive mode. The decoding procedure conforms to G.2 of T.81.

Input Arguments

- `pSrc` – pointer to the input bit stream buffer.
- `srcBytesLen` – length of bit stream.
- `pSrcCurrPos` – pointer to the current byte position in the bit stream buffer.
- `pMarker` – pointer to the marker.
- `al` – low bit position of successive approximation.
- `pHuffState` – pointer to the structure of Huffman decoding state.

Output Arguments

- `pSrcCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pDst` – pointer to the output block of decoded coefficient data.
- `pMarker` – pointer to the marker.
- `pHuffState` – pointer to the updated structure of Huffman decoding state.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeHuffman8x8_ACFirst_U1_S16_C1_DLx

Prototype

```
OMXResult omxICJP_DecodeHuffman8x8_ACFirst_U1_S16_C1_DLx (const OMX_U8*
    pSrc, OMX_INT srcBytesLen, OMX_INT* pSrcCurrPos, OMX_S16* pDst, OMX_INT*
    pMarker, OMX_INT ss, OMX_INT se, OMX_INT al, const
    OMXICJPHuffmanDecodeSpec* pACHuffTable, OMX_sDecodeHuffmanState*
    pHuffState);
```

Description

This function implements Huffman AC decoding of the first scans for progressive mode. The decoding procedure conforms to G.2 of T.81.

Input Arguments

- `pSrc` – pointer to the input bit stream buffer.
- `srcBytesLen` – length of bit stream.

- `pSrcCurrPos` – pointer to the current byte position in the bit stream buffer.
- `pMarker` – pointer to the marker.
- `ss` – start index of spectral selection.
- `se` – end index of spectral selection.
- `al` – low bit position of successive approximation.
- `pACHuffTable` – pointer to the structure of Huffman decoding table for AC. The table must be retrieved by calling `OMXICJPHuffmanDecodeSpecInit_JPEG_8u`.
- `pHuffState` – pointer to the structure of Huffman decoding state.

Output Arguments

- `pSrcCurrPos` – pointer to the updated byte position in the bit stream buffer.
- `pDst` – pointer to the output block of decoded coefficient data.
- `pMarker` – pointer to the marker.
- `pHuffState` – pointer to the updated structure of Huffman decoding state.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

DecodeHuffman8x8_ACRefine_U1S16_C1_DLx

Prototype

```
OMXResult omxICJP_DecodeHuffman8x8_ACRefine_U1S16_C1_DLx (const OMX_U8*
    pSrc, OMX_INT  srcBytesLen, OMX_INT* pSrcCurrPos, OMX_S16* pDst,
    OMX_INT* pMarker, OMX_INT ss, OMX_INT se, OMX_INT al, const
    OMXICJPHuffmanDecodeSpec* pACHuffTable, OMX_sDecodeHuffmanState*
    pHuffState);
```

Description

This function implements Huffman AC decoding of the first scans for progressive mode. The decoding procedure conforms to G.2 of T.81.

Input Arguments

- `pSrc` – pointer to the input bit stream buffer.
- `pSrcCurrPos` – pointer to the current byte position in the bit stream buffer.
- `pMarker` – pointer to the marker.
- `ss` – start index of spectral selection.
- `se` – end index of spectral selection.
- `al` – low bit position of successive approximation.
- `pACHuffTable` – pointer to the structure of Huffman decoding table for AC. The table must be gotten by calling OMXICJPHuffmanDecodeSpecInit_JPEG_8u.
- `pHuffState` – pointer to the structure of Huffman decoding state.

Output Arguments

- `pSrcCurrPos` – pointer to the updated byte position in the bit stream buffer.

- `pDst` – pointer to the output block of decoded coefficient data.
- `pMarker` – pointer to the marker.
- `pHuffState` – pointer to the updated structure of Huffman decoding state.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTGetBufSize_B53_S16_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTGetBufSize_B53_S16_C1IR_DLx (const OMXRect *pTileRect,
    OMX_INT *pSize);
```

Description

Get the buffer size (in bytes) for one-level 2D forward and inverse wavelet transformation on an image tile of 16-bit data using 5-3 reversible filter.

Input Arguments

- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.

Output Arguments

- `pSize` – pointer to an integer of size of the internal buffer used by 2D 5-3 wavelet transformation.

Returns

Standard OMXResult result. See enumeration for possible result codes.

WTGetBufSize_B53_S32_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTGetBufSize_B53_S32_C1IR_DLx (const OMXRect *pTileRect,
    OMX_INT *pSize);
```

Description

Get the buffer size (in bytes) for both one-level 2D forward and inverse wavelet transformation on an image tile (of 32-bit data) using 5-3 reversible filter.

Input Arguments

- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.

Output Arguments

- `pSize` – pointer to an integer of size of the internal buffer used by 2D 5-3 wavelet transformation.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTFwd_B53_S16_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTFwd_B53_S16_C1IR_DLx (OMX_S16 *pSrcDstTile, OMX_INT
   step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function makes a one-level forward 2D wavelet transformation on one image tile using the 5-3 reversible filter. The DWT coefficients are de-interleaved into LL, HL, LH and HH subbands and written back to the buffer of input data. The image tile data are in 16-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input image tile. The start address of `pSrcDstTile` must be 8-byte aligned. It is better to make it 32-byte aligned.
- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform.The start address of pBuffer must be 8-byte aligned. It is better to make it 32-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called. Its size can be retrieved by calling `ICJP2K_WTGetBufSize_B53_S16_C1IR`.

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output DWT coefficients.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTFwd_B53_S32_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTFwd_B53_S32_C1IR_DLx (OMX_S32 *pSrcDstTile, OMX_INT
   step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function makes a one-level forward 2D wavelet transformation on one image tile using the 5-3 reversible filter. The DWT coefficients are de-interleaved into LL, HL, LH and HH subbands and written back to the buffer of input data. The image tile data are in 32-bit data type.

2 ***Note:*** *The input image tile may be either a one tile-component of the image or the LL subband of the higher levels DWT coefficients. The DWT results are de-interleaved into four subbands, which are written to the buffer pointed to by* `*pSrcDstTile`.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input image tile. The start address of `pSrcDstTile` must be 8-byte aligned.

2 ***Note:*** *Although the start address of* `pSrcDstTile` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. It is better to make it 32-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called, and its size may be determined by calling ICJP2K_WTGetBufSize_B53_S32_C1IR.

2 ***Note:*** *Although the start address of* `pSrcDstTile` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output DWT coefficients.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTInv_B53_S16_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTInv_B53_S16_C1IR_DLx (OMX_S16 *pSrcDstTile, OMX_INT
    step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function interleaves the LL, HL, LH and HH subbands of DWT coefficients and then makes a one-level inverse 2D wavelet transformation on them using the 5-3 reversible filter. The results are written back to the buffer of input data. The image tile data are in 16-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input LL, HL, LH and HH subbands of DWT coefficients. The start address of pSrcDstTile must be 8-byte aligned.

---

2 ***Note:*** *Although the start address of* `pSrcDstTile` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called, and its size can be retrieved by calling `ICJP2K_WTGetBufSize_B53_S16_C1IR`.

---

2 ***Note:*** *Although the start address of* `pBuffer` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output image tile.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTInv_B53_S32_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTInv_B53_S32_C1IR_DLx (OMX_S32 *pSrcDstTile, OMX_INT
   step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function interleaves the LL, HL, LH and HH subbands of DWT coefficients and then makes a one-level inverse 2D wavelet transformation on them using the 5-3 reversible filter. The results are written back to the buffer of input data. The image tile data are in 32-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input LL, HL, LH and HH subbands of DWT coefficients. The start address of `pSrcDstTile` must be 8-byte aligned.

2 ***Note:*** *Although the start address of* `pBuffer` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. The work buffer pointed by `pBuffer` must be allocated before the function is called, and its size may be determined by calling `ICJP2K_WTGetBufSize_B53_S32_C1IR.`

2 ***Note:*** *Although the start address of* `pBuffer` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output image tile.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTGetBufSize_D97_S16_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTGetBufSize_D97_S16_C1IR_DLx (const OMXRect *pTileRect,
    OMX_INT *pSize);
```

Description

Get the buffer size (in bytes) for one-level 2D forward and inverse wavelet transformation on an image tile of 16-bit data using a 9-7 reversible filter.

Input Arguments

- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.

Output Arguments

- `pSize` – pointer to an integer of size of the internal buffer used by a 2D 9-7 wavelet transformation.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTGetBufSize_D97_S32_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTGetBufSize_D97_S32_C1IR_DLx (const OMXRect *pTileRect,
    OMX_INT *pSize);
```

Description

Get the buffer size (in bytes) for both one-level 2D forward and inverse wavelet transformation on an image tile of 32-bit data using a 9-7 reversible filter.

Input Arguments

- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.

Output Arguments

- `pSize` – pointer to an integer of size of the internal buffer used by 2D 9-7 wavelet transformation.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTFwd_D97_S16_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTFwd_D97_S16_C1IR_DLx (OMX_S16 *pSrcDstTile, OMX_INT
    step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function makes a fix-point implementation of one-level forward 2D wavelet transformation on one image tile using the 9-7 irreversible filter. The DWT coefficients are de-interleaved into LL, HL, LH and HH subbands and written back to the buffer of input data. The image tile data are in 16-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input image tile. The start address of pSrcDstTile must be 8-byte aligned.

---

2 ***Note:*** *Although the start address of* `pSrcDstTile` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.

- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called. Its size can be retrieved by calling `ICJP2K_WTGetBufSize_D97_S16_C1IR`.

---

2 ***Note:*** *Although the start address of* `pBuffer` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output DWT coefficients.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTFwd_D97_S32_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTFwd_ D97_S32_C1IR_DLx (OMX_S32 *pSrcDstTile, OMX_INT
    step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function makes a fix-point implementation of one-level forward 2D wavelet transformation on one image tile using the 9-7 irreversible filter. The DWT coefficients are de-interleaved into LL, HL, LH and HH subbands and written back to the buffer of input data. The image tile data are in 32-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input image tile. The start address of `pSrcDstTile` must be 8-byte aligned.

---

2 ***Note:*** *Although the start address of* `pSrcDstTile` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure, which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called. Its size can be gotten by calling `ICJP2K_WTGetBufSize_D97_S32_C1IR.`

> 2 ***Note:*** *Although the start address of pBuffer must be 8-byte aligned, it is better to make it 32-byte aligned.*

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output DWT coefficients.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTInv_D97_S16_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTInv_D97_S16_C1IR_DLx(OMX_S16 *pSrcDstTile, OMX_INT
    step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function interleaves the LL, HL, LH and HH subbands of DWT coefficients and then makes a fix-point implementation of one-level inverse 2D wavelet transformation on them using the 9-7 irreversible filter. The results are written back to the buffer of input data. The image tile data are in 16-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input LL, HL, LH and HH subbands of DWT coefficients. The start address of `pSrcDstTile` must be 8-byte aligned. It is better to make it 32-byte aligned.

> 2 ***Note:*** *Although the start address of* `pBuffer` *must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called. Its size can be gotten by calling `ICJP2K_WTGetBufSize_D97_S16_C1IR`.

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output image tile.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

WTInv_D97_S32_C1IR_DLx

Prototype

```
OMXResult omxICJP2K_WTInv_D97_S32_C1IR_DLx (OMX_S32 *pSrcDstTile, OMX_INT
    step, const OMXRect *pTileRect, OMX_U8 *pBuffer);
```

Description

This function interleaves the LL, HL, LH and HH subbands of DWT coefficients and then makes a fix-point implementation of one-level inverse 2D wavelet transformation on them using the 9-7 irreversible filter. The results are written back to the buffer of input data. The image tile data are in 32-bit data type.

Input Arguments

- `pSrcDstTile` – pointer to the buffer of input LL, HL, LH and HH subbands of DWT coefficients. The start address of pSrcDstTile must be 8-byte aligned. It is better to make it 32-byte aligned.

---

2 ***Note:*** *Although the start address of pBuffer must be 8-byte aligned, it is better to make it 32-byte aligned.*

- `step` – specifies the number of bytes in a line of the input data buffer. step is in bytes, and must be an integer multiple of 8.
- `pTileRect` – pointer to an OMXRect data structure which indicates the position and size of the image tile.
- `pBuffer` – pointer to the work buffer for transform. The start address of pBuffer must be 8-byte aligned. The work buffer pointed by pBuffer must be allocated before the function is called. Its size can be retrieved by calling `ICJP2K_WTGetBufSize_D97_S32_C1IR`.

Output Arguments

- `pSrcDstTile` – pointer to the buffer of output image tile.

Returns

- Standard OMXResult result. See enumeration for possible result codes.

Pre-processing / Post-processing Sub-Domain (omxIPPP)

Dering Functions

Dering_Luma_DLx

Prototype

```
OMXResult omxIPPP_Dering_Luma_DLx (const OMX_U8 *pSrc, const OMXSize
    *roiSize, const OMX_U32 srcStep, const OMX_U32 dstStep, const OMX_S16
    *pQuant, OMX_U8 *pDst);
```

Description

This filter de-rings a region within the luminance plane of an image. The input and output buffers represent the luminance color plane of an image, 8-bits per pixel. The output buffer region is required to have the same size as that of the input region, although the width of the overall images may differ.

Input Arguments

- `pSrc` – pointer to the input buffer
- `roiSize` – the dimensions of the input region in pixels
- `srcStep` – step in bytes through the source image
- `dstStep` – step in bytes through the destination image
- `pQuant` – buffer specifying the quantization factor of each 16x16 block (not used if NULL)

Output Arguments

- `pDst` – pointer to the output buffer (a single image plane, 8bpp)

Returns

- Standard OMXResult result. See enumeration for possible result codes.

Dering_Chroma_DLx

Prototype

```
OMXResult omxIPPP_Dering_Chroma_DLx (const OMX_U8 *pSrc, const OMXSize
   *roiSize, const OMX_INT srcStep, const OMX_INT dstStep, const OMX_S16
   *pQuant, OMX_U8 *pDst);
```

Description

This filter de-rings a region within a chrominance plane of an image. The input and output buffers represent a single chroma color plane of an image, 8-bits per pixel. The output buffer region is required to have the same size as that of the input region, although the width of the overall images may differ.

Input Arguments

- `pSrc` – pointer to the input buffer (a single image plane, 8bpp)
- `roiSize` – the dimensions of the input region in pixels
- `srcStep` – step in bytes through the source image
- `dstStep` – step in bytes through the destination image
- `pQuant` – buffer specifying the quantization factor of each 16x16 block (not used if NULL)

Output Arguments

- `pDst` – pointer to the output buffer (a single image plane, 8bpp)

Returns

- Standard OMXResult result. See enumeration for possible result codes.

Pre-processing / Post-processing Sub-Domain (omxIPPP)

Dering Functions

Raw Pixel Processing Functions

Data Structures and Enumerated Types

The table below defines data structures to support raw pixel processing.

Configuration Data Structures

Type Name	Corresponding data type in C

OMXRawPixProcCfg_P3R	`typedef struct`
	`{`
	`    OMX_INT srcStep; /** Distance, in bytes, between the start of lines in the source image */`
	`      OMXSize srcSize;    /** Dimensions, in pixels, of the source regions of interest */`
	
	`    OMX_INT dstStep[3]; /** A 3-element vector containing the distance, in bytes, between the start of lines in each of the output image planes */`
	`    OMXSize dstSize;     /* Dimensions, in pixels, of the output source regions of interest */`
	
	`    OMXIPInterpolation interpolation; /** Interpolation method used for CFA interpolation and resizing */`
	`    OMXIPRotation rotation;   /** Rotation control parameter */`
	`    OMXIPColorSpace colorConversion; /** Color conversion control parameter */`
	`} OMXRawPixProcCfg_P3R;`

OMXCAMCfg	`typedef struct`
	`{`
	`      OMX_INT BitDepth; /** Bit depth of the input raw data from Sensor.`
	`                        * Typically 8, 9, or 10 bits */`
	`    OMXGam GammaFlag; /** Indicates the type of gamma`

Type Name	Corresponding data type in C

	table
	                        * 0: R/G/B use one predefined table
	                        * 1: R/G/B use one application-specific table
	                        * 2: R/G/B use three predefined tables repectively
	                        * 3: R/G/B use three application-specific tables
	                        * repectively */
	    OMX_INT GammaIndex[3]; /** Indicates the index of predefined gamma tables
	                              * when GammaFlag set to predefined options*/
	    OMX_U8 *pGammaTable[3]; /** A 3-element vector containing the pointer to the gamma tables for R/G/B respectively when GammaFlag set to application-specific option */
	
	    OMX_U32 *pDeadPixMap; /** Pointer to the start of the absolute Dead
	                              * Pixel Map of the sensor array */
	
	    OMX_INT DPMLen; /** Length, in pixels, of the Dead Pixel Map */
	    OMXPoint DPMOffset; /** Offset of start position of the cropping
	                              * window, from the (0,0), DPMOffset.x is the row offset
	                             * and DPMOffset.y is the column offset */
	    OMXIPInterpolation DPInterp; /** Interpolation method used for dead pixel
	                       *substitution */
	    OMX_S16 *pCCMatrix; /** Color correction Matrix */
	
	} OMXCAMCfg;

Type Name	Corresponding data type in C

OMXCAMCfg
(continued)
	`OMXIPInterpolation DPInterp;`
	`/* Inperpolation method used for dead pixel substitution */`
	`OMX_S16* pCCMatrix;`
	`/* Color correction Matrix */`
	`} OMXCAMCfg;`

OMXRawPixProcSpec_P3R;	`typedef void OMXRawPixProcSpec_P3R;OMXRawPixProcSpec_P3R;`

The table below defines enumerated types to support raw pixel processing.

Enumerated Types

Enumerated Type Name	Symbolic Values	Constant Value
OMXGam	omxGamPreOneTable	0
	omxGamCusOneTable	1
	omxGamPreThreeTable	2
	omxGamCusThreeTable	3

RGGBtoYCbCrGetBufSize_DLx
RGGBtoYCbCrInit_DLx
RGGBtoYCbCr_RotRsz_8u_P3R_DLx

Prototype

```
OMXResult omxIPCS_RGGBtoYCbCrGetBufSize_DLx(const OMXCAMCfg *pCAMCfg, const
    OMXRawPixProcCfg_P3R *pRPPCfg, OMX_INT *pSize);
OMXResult omxIPCS_RGGBtoYCbCrInit_DLx(const OMXCAMCfg *pCAMCfg, const
    OMXRawPixProcCfg_P3R *pRPPCfg, OMXRawPixProcSpec_P3R *pRPPSpec);
OMXResult omxIPCS_RGGBtoYCbCr_RotRsz_U8_P3R_DLx(OMX_U8 *pSrc, OMX_U8
    *pDst[3], OMXRawPixProcSpec_P3R *pRPPSpec);
```

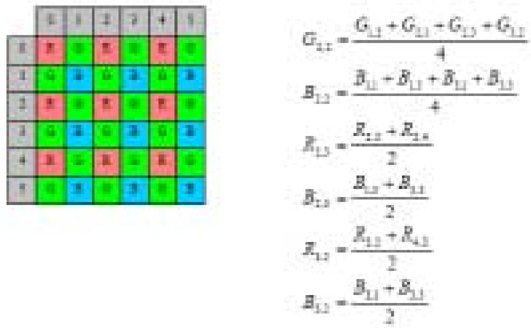
Description

The raw pixel processing functions convert raw RGGB pixel data to YCbCr422/420 planar data using the following sequence of operations: optional dead pixel correction, companding and gamma correction, optional scale reduction, color synthesis, color correction, color conversion, and optional rotation.

The color synthesis methodology is a function of the specified resize ratio. Three cases are possible: a) no scale reduction, b) 2:1 scale reduction, or c) 4:1 scale reduction

a. No scale reduction

Bilinear interpolation is applied. Missing colors are synthesized by averaging the nearest neighbor pixel intensities of the same color, possibly as shown in the figure below.



b. 2:1 scale reduction

R, G and B values for one output pixel are calculated from values of four adjacent source pixels (2*2 block of RGGB). R and B values are set equal to the values of the nearest red and blue pixels, while G is set equal to the average of the two nearest green pixels, possibly as shown in the figure below.

c. 4:1 scale reduction

A possible approach is shown in the figure below.

The color space conversion is computed as follows:

$$Y = 0.29900 * R + 0.58700 * G + 0.11400 * B$$

$$Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B + 128$$

$$Cr = 0.50000 * R - 0.41869 * G - 0.08131 * B + 128$$

The initialization function <`omxIPCS_RGGBtoYCbCrInit`> should be called prior to <`omxIPCS_RGGBtoYCbCr_RotRsz_U8_P3R`> in order to initialize the raw data processing state structure.

Input Arguments

- `pSrc` – pointer to the start of the buffer containing the pixel-oriented RGGB input image. The input image buffer referenced by `pSrc` must contain one pixel of padding on all four edges; ie the buffer size should be equal to the image height+2 by the image width+2. `pSrc` should point to the start to the ROI but not the start of the padding region.

- pCAMCfg – pointer to the sensor-specific configuration structure
- pRPPCfg – pointer to the application-specific configuration structure
- pRPPSpec – pointer to the implementation-specific state structure

Output Arguments

- pDst – a 3-element vector containing pointers to the start of the YCbCr422/YCbCr420 output planes
- pSize – pointer to the variable to hold the structure size.

Returns

- If the function runs without error, it returns OMX_StsNoErr
- If one of the following cases occurs, the function returns OMX_StsBadArgErr:
  - — Any of the below pointer is NULL:

    pSrc, pDst[0], pDst[1], pDst[2], pCAMCfg, pRPPCfg, pRPPSpec pCAMCfg or pRPPCfg contains invalid values.

    pDst[0], pDst[1] or pDst[2] does not match the corresponding alignment requirement. Refer to the following table for alignment requirement details.
- For pCAMCfg, following cases are invalid:
  - — BitDepth != 10.
  - — Invalid values for enumerated parameter GammaFlag
  - — When GammaFlag is specified as OMXGamPreOneTable, GammaIndex[0] is not between 0~1
  - — When GammaFlag is specified as OMXGamCusOneTable, pGammaTable[0] is NULL
  - — GammaFlag is specified as OMXGamPreThreeTable or OMXGamCusThreeTable
  - — When pDeadPixMap is not NULL, DPMLen<=0 or DPMOffset.x<0 or DPMOffset.y<0.
  - — DPInterp != omxCameraInterpNearest
  - — pCCMatrix is NULL.
- For pRPPCfg, the following cases are invalid:
  - — For no-resizing cases, srcSize.width or srcSize.height is less than 2
  - — For 2:1 and 4:1 scale down cases, srcSize.width is less than 4 or srcSize.height is less than 2
  - — srcStep is less than srcSize.width, dstStep[0] is less than dstSize.width, dstStep[1] is less than dstSize.width/2 or dstStep[2] is less than dstSize.width/2.

    "dstSize.width or dstSize.height is not even.
  - — For no resizing cases, dstStep[0] is not multiple of 8, dstStep[1] is not multiple of 4 or dstStep[2] is not multiple of 4.
  - — srcSize is not dstSize multiply 1, 2 or 4
  - — Sizes of the input image and the output image are incompatible with the rotation configuration.
  - — Invalid values for enumerated parameters: interpolation, rotation, colorConversion.

— For no scale reduction, interpolation is not `omxCameraInterpMedian`. For 2:1 and 4:1 scale reduction, interpolation is not `omxCameraInterpNearLinear`

— `rotation` is not set to one of the following values: `OMX_IP_DISABLE`, `OMX_IP_ROTATE90L`, `OMX_IP_ROTATE90R`, `OMX_IP_ROTATE180`.

— `colorConversion` is not set to either `OMX_IP_YCBCR422` or `OMX_IP_YCBCR420`.

---

2 ***Note:*** *Image buffer requirement: Input RGGB raw data should be arranged as follows: ":R-G-G-B," i.e., the pSrc(0,0) should be 'R', pSrc(0,1) should be 'G', pSrc(1,0) should be 'G', pSrc(1,1) should be 'B'.*

Image Size Requirements

The `srcSize.width` and `srcSize.height` must be even. When resizing is applied, the `dstSize.width` should be multiple of 4 and `dstSize.height` should be multiple of 2.

Alignment Requirements

Alignment requirements are shown in the table. Pointers must meet the alignment requirements, and in addition the corresponding image steps must be a multiple of the pointer alignment.

Alignment requirements

	pDst[0]		pDst[1]		pDst[2]	
Scale Reduction	**Alignment (byte)**	**Offset (byte)**	**Alignment (byte)**	**Offset (byte)**	**Alignment (byte)**	**Offset (byte)**
1:1	8	0	4	0	4	0
2:1	4	0	2	0	2	0
4:1	4	0	2	0	2	0

Gamma Correction Requirements

Either predefined or application-specific gamma correction tables can be applied. Moreover, either a single gamma table or three different gamma tables can be applied independently to the R, G and B components.

Gamma Table Usage

GammaFlag	Remarks
OMXGamPreOneTable	GammaIndex[0] must set to a valid predefined gamma table index.

GammaFlag	Remarks
OMXGamPreThreeTable	GammaIndex[0], GammaIndex[1] and GammaIndex[2] must set to valid predefined gamma table indices.
OMXGamCusOneTable	pGammaTable[0] must point to a valid table address.
OMXGamCusThreeTable	pGammaTable[0], pGammaTable[1] and pGammaTable[2] must point to valid table addresses.

The following table provides the indices of predefined gamma tables.

Gamma Tables and Indexes

Index	Gamma Correction Formula	Remarks
0	$y = x$	For linear display.
1	$= \begin{bmatrix} 4.5x, 0 \le 0.018(\chi) \\ 1.099x^{0.45} - (0.99, 018 \le x \le 1) \end{bmatrix}$	Rec. 709's transfer function.

Below is an example of RGGB to YCbCr420 planar with a 2:1 scale reduction.