VC-1 Decoder Software Technical Reference Manual

Embedded Software Division

Document version: 2.1
Date of Issue: 14 January 2005
Author: ARM

Abstract

Documentation on the implementation and operation of a reference implementation of the VC-1 video bitstream format standard.

Keywords

VC-1,Video,Standard,Decoder

Contents

1 ABOUT THIS DOCUMENT

1.1 Change control

1.1.1 Current status

This document describes release-quality code that, at the time of publication, conforms to CD2r1 of the VC-1 specification (reference [1] below). Changes from the previous release are documented in the accompanying release note (reference [4] below).

1.1.2 Change history

Issue	Date	By	Change
1.0	15 September 2004	ARM	First release.
2.0	7 December 2004	ARM	Second release.
2.1	14 January 2005	Microsoft	Updated copyright statement

1.2 References

This document refers to the following documents.

Ref	Document	Author(s)	Title
[1]	SMPTE Standard xxxM	Microsoft	VC-1 Compressed Video Bitstream Format and Decoding Process
[2]	VC-1 decoder library source code	ARM	
[3]	SMPTE Recommended Practice xxx	Microsoft	VC-1 Decoder and Bitstream Conformance
[4]	Decoder Software Release Note	ARM	

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
API	Application Program Interface
BDU	Bitstream Data Unit
CBP	Coded block pattern.
MB	Macroblock
MV	Motion Vector
SMPTE	Society of Motion Picture and Television Engineers
VC-1	The proposed SMPTE VC-1 video bitstream format standard

2 SCOPE

This document describes the structure and API of the VC-1 decoder implementation by ARM [2]. It provides a high level overview of how the decoder operates. Details of functions and structures used in the library are contained within comments in the source code [2]. Some of these details are also presented in sections 8 and 9 of this document, but the source is the reference for any fine detail.

3 INTRODUCTION

The VC-1 decoder described here is designed as a reference decoder supporting all the functionality defined in the standard [1]. It is capable of decoding I, P, B, BI and skipped frames, in all valid profiles. The decoder is written with portability and simplicity in mind. It is not optimized for performance although care has been taken to keep data structures to a reasonable size.

This document is split into the following sections:

- Description of project structure
    - Directory and module structure
    - Design decisions
    - General information on each module
- Decoder operation
    - Decoder overview
    - Decoder example
    - Decoder API
- Decoder application
    - Description of code implementing functionality not related to the specification.

- Reference Section
    - Data structures used
    - Functions used

4 PROJECT STRUCTURE

4.1 Directories and modules

The decoder consists of various modules. The items within the modules follow a naming convention:

- if the module is part of the bitstream format as defined in the specification, the name starts with `vc1`
- the name of the top-level module is then added
- the module name for the component follows
- the descriptive part of the name then follows:
    - an underscore separates the prefix from the descriptive part
    - a 'limited Hungarian' set of prefixes is then used:
        - r pointer to a function
        - p pointer to any other item
        - e enumerated type
        - s variable or type is a structure
    - the name follows, using CapitalsToMarkWords (rather than underscores).

For example, the function to unpack sequence layer information is named `vc1DECSEQ_UnpackSequenceLayer`. This is made up from:

- the bitstream format prefix: `vc1`
- the top-level module prefix: `DEC`
- the specific module prefix: `SEQ`
- the function description: `_UnpackSequenceLayer`

Each module is contained within a source file. The filename consists of the module prefixes (lowercased) as the main part of the name, followed by the type suffix (for example `.c` or `.h`). Each top-level module is held within a separate directory. For example, the above function is declared in file `decoder/vc1decseq.h`.

4.2 Design decisions

The code has been designed to be portable across a wide variety of systems, including ones with restricted facilities. The major points are discussed in the following sections.

4.2.1 Type abstraction

All variables with specific range requirements are defined in terms of abstracted types that can be redefined as needed for alternative compilers, with different basic sizes. The types are defined in `shared\vc1types.h`, and are as follows:

Type name	Type definition
BYTE8	A signed 8-bit value
UBYTE8	An unsigned 8-bit value
HWD16	A signed 16-bit value
UHWD16	An unsigned 16-bit value
WORD32	A signed 32-bit value
UWORD32	An unsigned 32-bit value
LLONG64	A signed 64-bit value
ULLONG64	An unsigned 64-bit value
FLAG	A Boolean value

Figure 1 - Abstract Types

4.2.2 Memory allocation

The release build of the decoder library does not use global variables, static variables, or heap memory allocation (`malloc`, etc.). Memory must be allocated per-session by the calling application. This means that the library is fully re-entrant, and is not dependent on the availability of a heap manager. The debug build of the library does use global and static variables.

Note that the file handling module does use global and static variables, and also calls `malloc`. The file handling is regarded as being part of the application, and not the decoder library.

4.2.3 Library use

The code makes minimal use of the C library functions. Currently, only `memset` and `memcpy` are used, in non-debug builds. These functions are easy to re-implement on any system lacking them.

Debug builds make more use of library functions, such as `printf`. However, the debug code is small and self-contained, and so can be retargeted to available system primitives.

4.2.4 Error handling

The code returns error indicators throughout, rather than relying on any exception mechanism. The error codes are defined by an enumerated type (`vc1_eResult`), which can take the following values:

vc1_ResultFatal	Fatal condition detected
vc1_ResultWarn	Continuable fault detected
vc1_ResultOK	Function completed successfully
vc1_ResultNoFrame	No frame to display (due to buffering)
vc1_ResultSlice	More slice data required to complete decode
vc1_ResultField	Second field data required to complete decode

vc1_ResultInvalidParameter	Option value not accepted
vc1_ResultBadFile	File will not open
vc1_ResultBadLine	Line in option file not parseable
vc1_ResultBadType	Invalid type for parameter
vc1_ResultNoMemory	malloc() failed
vc1_ResultNoData	Failed to read picture data
vc1_ResultBufferExhausted	No more data to read from buffer
vc1_ResultBadImageSize	Size breaks restrictions set by the standard
vc1_ResultImageTooBig	Size bigger than profile limits
vc1_ResultImageSizeChanged	Size changed in simple or main profile
vc1_ResultUnsupportedTransform	Invalid transform
vc1_ResultACRunLevelDecodeFailed	Decoder failed to read the AC coef run levels
vc1_ResultNoStartCode	Start code not found in file input data

Note that not all calls can return all error codes. Note also that some codes do not indicate errors: ***vc1_ResultNoFrame***, ***vc1_ResultSlice*** and ***vc1_ResultField*** indicate that the function completed successfully but the decoder has reached a specific state.

4.3 Decoder modules

The table below describes the contents of each of the modules that make up the VC-1 decoder library. The top-level module with prefix `DEC` is held within the `decoder` directory.

Module name	Contents
vc1DEC	Decoder top level API functions
vc1DEC3DH	Decoder variable length code decoding functions
vc1DEC3DHTAB	Decoder variable length code decoding tables
vc1DECBIT	Decoder bitstream reading functions
vc1DECBITPL	Decoder bitplane decoding functions
vc1DECBITPLTAB	Decoder bitplane decode tables
vc1DECBLK	Decoder block level decode functions
vc1DECBLKTAB	Decoder block level decode tables
vc1DECENT	Decoder entry point layer decode functions
vc1DECMB	Decoder macroblock level decode functions
vc1DECMBTAB	Decoder macroblock level decode tables
vc1DECMV	Decoder motion vector decode functions
vc1DECPIC	Decoder picture level decode functions
vc1DECPICTAB	Decoder picture level decode tables

vc1DECSEQ	Decoder sequence layer reading functions
vc1DECSLICE	Decoder slice layer reading functions
vc1DECZZ	Decoder de-zigzag functions

4.4 Generic modules

Generic code has no top-level module prefix. It is held within the `shared` directory.

Module name	Module functions
vc13DHTAB	Tables for variable length code decode
vc1CROPMV	Motion vector pull-back/crop functions
vc1DEBLOCK	In-loop deblocking filter functions
vc1DEBUG	Debug routines (not required in a release build)
vc1DERIVEMV	Motion vector derivation routines
vc1GENTAB	General tables
vc1HRD	Hypothetical reference decoder
vc1INTERP	Block interpolation functions
vc1IQUANT	Inverse Quantize functions
vc1ITRANS	Inverse Transform functions
vc1PRED	Routines for use by prediction functions
vc1PREDCBP	Prediction of CBPY (coded block pattern Y values)
vc1PREDDCAC	Prediction of DC and AC quantized coefficients
vc1PREDMV	Motion vector prediction
vc1RECON	Block level reconstruction (inverse quantize, transform, smooth, clamp)
vc1SCALEMV	Motion vector scaling
vc1SMOOTH	Overlap smooth filter
vc1TOOLS	General tool functions
vc1ZZTAB	Zigzag tables

4.5 Building the decoder

The decoder is written in standard ANSI C, and should compile and run with any standard C compiler.

For Visual C++ users, a project workspace file is supplied within the decoder directory. To build the decoder, load the project workspace file (`decoder\decoder.dsw`) into Visual C++, and select menu entry `Build -> Build`, or press F7. Visual C++ version 6.0 (with all Service

Packs installed) or higher is required to be sure of correct operation – this reflects the systems used to test the code during development.

For other compilers, a makefile is supplied within the decoder directory. Adapt this to suit the local development environment.

4.5.1 Selecting and controlling debug output

To include code to produce debugging information, ensure that the 'Win32 Debug' build variant is selected, and that the build is up to date. The debug output is divided into zones which can each be independently enabled or disabled at runtime. For the available zone names and bit settings, see the `shared\vc1debug.h` header file. For more information on how to select the zones to be displayed, see section 6.2.3.4, DebugMask.

5 DECODER OPERATION

5.1 Decoder overview

This section provides an overview of the decoder API functions. For details of the API functions and structures, see the header file `decoder\vc1dec.h`.

The decoder takes a single command-line parameter, which is the name of the options file. This file defines all the options used when running the decoder. For details of the option file format and contents, see section 6.2, Option processing.

The following graphs show the relationship between the major functions within a decoder application:

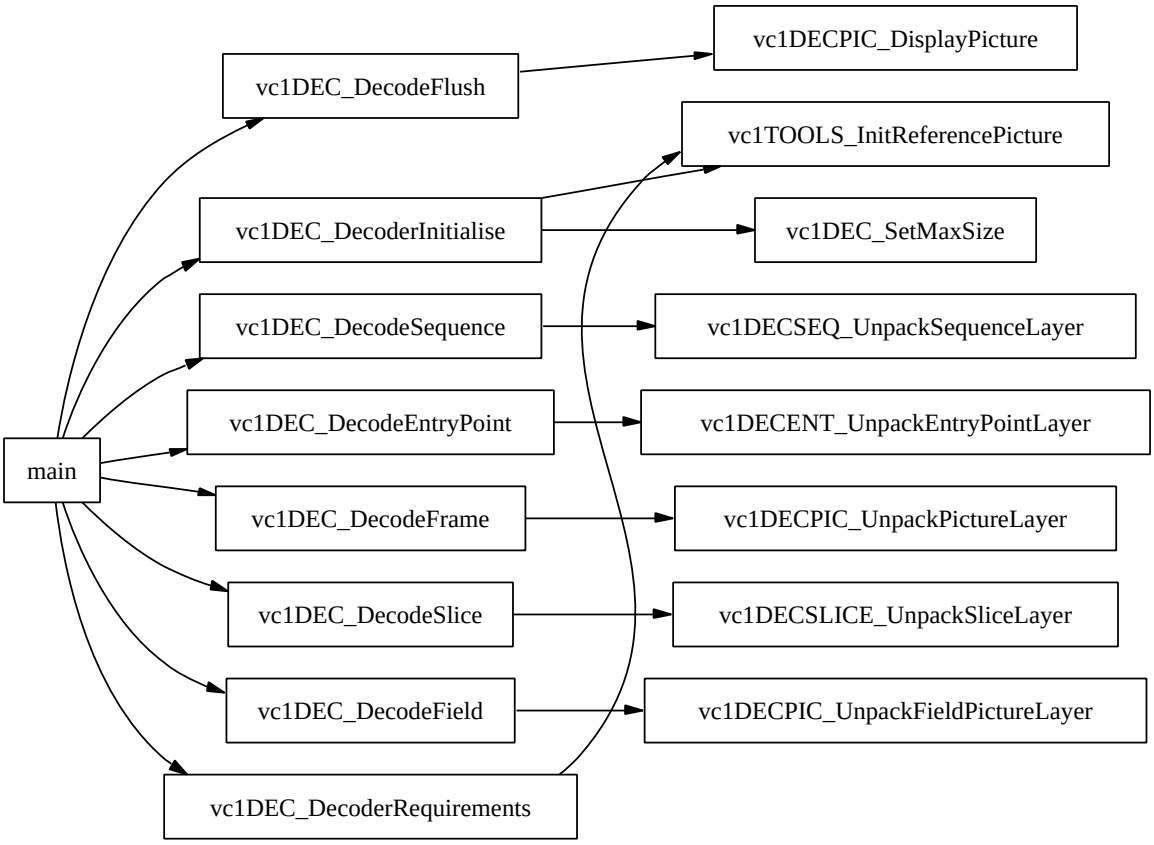

Figure 2: Major decoder functions

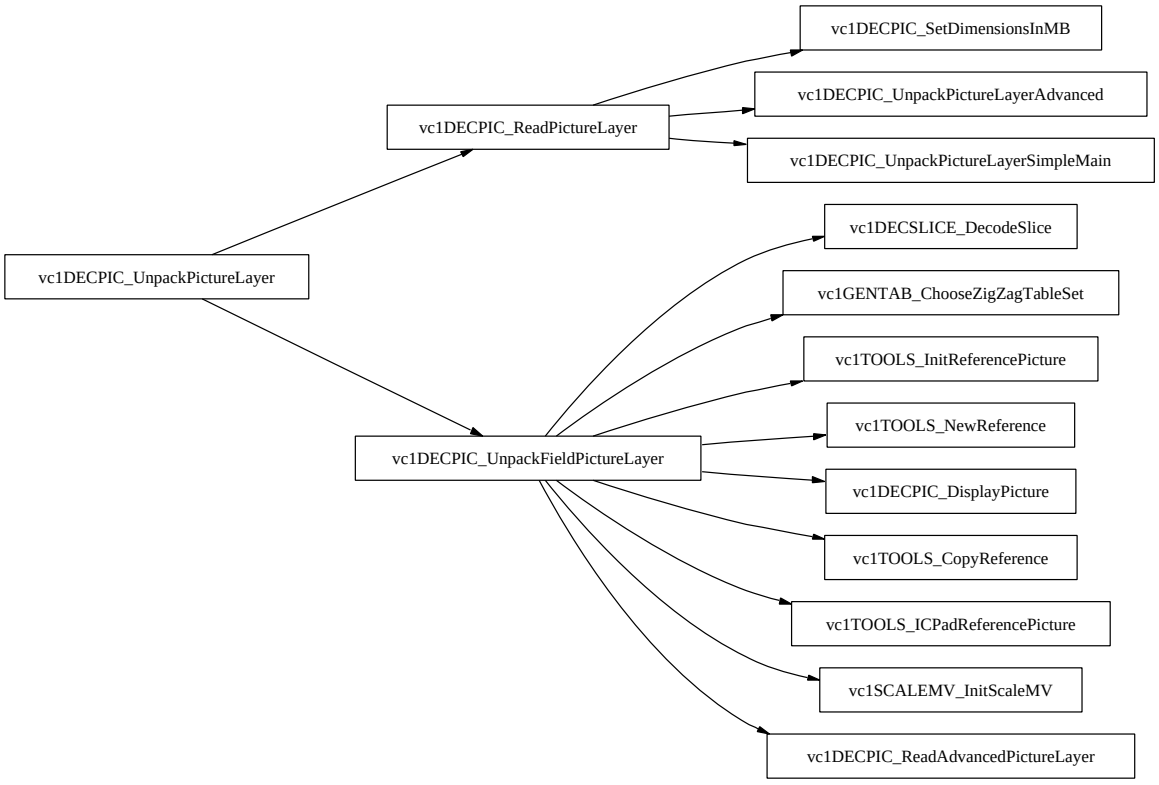

Figure 3: Unpack Picture Layer function

5.2 Decoding process

Decoding an I picture involves unpacking the picture, macroblock and block layers, predicting the information about the current block from previous blocks, decoding the DC and AC coefficients, de-zigzagging and dequantising the transform coefficients, and finally performing an inverse transform. Processing operations may be applied to the image in the display buffer.

Decoding a P picture involves unpacking the picture, macroblock and block layers, predicting information about the current block from previous blocks, decoding the AC coefficients, decoding the motion vectors of the predicted block, de-zigzagging and dequantising the transform coefficients, performing the inverse transform, and finally combining the difference block with the motion vector predicted block.

Decoding a B picture is similar to the P picture case, but up to two motion vectors are decoded, which are used to predict blocks from up to two reference pictures.

A complete macroblock is decoded before moving on to the next macroblock; within each macroblock, all blocks are unpacked, then all blocks decoded, predicted and written to the picture. Note that deblocking is not included within this decoding: deblocking is performed as a separate pass.

5.3 Decoder example session

The following pseudo-code shows a simple example of a decoder session, for processing a Main profile bitstream:

```
Configuration = Initial Decoder Configuration;
vc1DEC_DecoderRequirements(&Size, &Configuration, &Bitstream);
State = malloc(Size);
vc1DEC_DecoderInitialise(State, &Configuration);
vc1DEC_DecodeSequence(State, &Bitstream);
while ( Bitstream not exhausted )
{
        vc1DEC_DecodePicture(State, &Bitstream, &Picture);
        /* display or write picture to disk */
}
vc1DEC_DecodeFlush(State, &Picture);
/* display or write picture to disk */
```

Figure 4 - decoder example

5.4 Details of decoder operation

The following sections describe, in order, the stages involved in decoding a picture, from bitstream input to YUV pixel output.

5.4.1 Bitstream unpack

The sequence, picture and macroblock layers of the bitstream are unpacked according to sections ‘Bitstream syntax and semantics’, ‘Progressive bitstream syntax and semantics’ and ‘Interlace syntax and semantics’ of the standard [1], and the data they represent stored in the decoder’s state structure.

5.4.2 Prediction

The CBP, MV, DC and AC coefficients are predicted for the current block based upon the previous blocks, as described in sections ‘Coded Block Pattern’, ’Motion Vector Predictors’, ‘DC Predictor’ and ‘AC Prediction’ of the standard [1] respectively.

5.4.3 DC coefficient decode

If the picture is type I, or the block is intra type, the quantised DC coefficient is extracted from the block layer, added to the predicted DC coefficient, and stored in the first element of the transform coefficient array. The predicted DC coefficient is added at this stage.

5.4.4 AC coefficient reconstruction

If the CBP indicates AC coefficients are present in this block, the algorithm described in the figure named ‘Coefficient decode pseudo-code’ of the standard [1] is used to obtain each run and level pair. These are used to reconstruct the AC coefficients of the transform coefficient array according to the algorithm described in the figure named ‘Run-level decode pseudo-code’ of the standard [1]. These are stored in a temporary array.

5.4.5 Zigzag scan

If the CBP indicated AC coefficients are present in this block, the temporary transform coefficient array is rearranged using a zigzag scan array, and the result put into the transform coefficient array. The results are combined with predicted AC coefficients, if AC prediction is enabled.

5.4.6 Dequantise

The DC coefficient is dequantised according to the algorithm described in section ‘DC Inverse-quantization’ of the standard [1].

If the CBP indicated AC coefficients are present in this block, they are dequantised according to the algorithm described in section ‘Inverse AC Coefficient Quantization’ of the standard [1]. The transform coefficient array is dequantised in place.

5.4.7 Inverse transform

The inverse transform operation is performed according to the specification in the annex ‘Inverse Transform Specification’ of the standard [1]. The operation is performed on the transform coefficient array, in place.

Unclamped results from the macroblock are stored for use in overlap smoothing.

5.4.8 Block prediction

If the picture is type P, a macroblock is obtained, interpolated if necessary, from the reference picture, and combined with the resulting macroblock from the inverse transform.

If the picture is type B, one or two macroblocks are obtained, interpolated if necessary, from one or two reference pictures, averaged and combined with the resulting macroblock from the inverse transform.

The macroblock is clamped and copied into the key frame buffer in the decoder’s state structure, for future prediction use.

5.4.9 Post process

If enabled, overlap smoothing is performed as described in section ‘Overlapped Transform’ of the standard [1]. Data from previous macroblocks required to perform the operation is stored in the decoder state. The image is also deblocked, if the bitstream requires it.

5.4.10 Display

The key frame buffer is copied to the display buffer, which is returned to the application. Range reduction and resolution scaling is performed at the same time, if required by the bitstream.

5.5 Decoder API functions

This section describes the API functions used by the decoder. For more detailed internal structure and function documentation, see section 8, VC-1 Data Structure Documentation, and section 9, VC-1 File Documentation.

5.5.1 Decoding

5.5.1.1 vc1DEC_DecoderRequirements()

This function is called once per bitstream. It will examine the bitstream and calculate the amount of memory that the application must allocate for the decoder to decode the bitstream.

5.5.1.2 vc1DEC_DecoderInitialise()

This function initialises the area of memory allocated by the application for use by the decoder.

5.5.1.3 vc1DEC_DecodeFrame()

This function is called once for each picture to be decoded. It will decode a whole or part of a picture, depending on whether the picture is split into slices or fields. It will write the resulting data into the decoder's picture pipeline if a complete picture is available, otherwise further calls to vc1DEC_DecodeSlice() or vc1DEC_DecodeField() will be required. A picture will be copied to the application's picture buffer if the pipeline has been filled, which will be indicated by the function's return value.

5.5.1.4 vc1DEC_DecodeFlush()

This function is called once per bitstream, at the end of a decoding session. It flushes the decoder's picture pipeline, and writes the picture to the application's picture buffer. No bitstream is consumed, and no decoding operations are performed.

5.5.1.5 vc1DEC_DecodeSequence()

This function is called when a sequence layer BDU is received. It decodes the layer, and puts the information it contains into the sequence parameters structure, which may be accessed by the application.

5.5.1.6 vc1DEC_DecodeEntryPoint()

This function is called when an entry point layer BDU is received. It decodes the layer, and puts the information it contains into the sequence parameters structure, which may be accessed by the application.

5.5.1.7 vc1DEC_DecodeSlice()

This function is called when a slice layer BDU is received. It decodes the layer, and any image data it contains. The decoder's internal picture buffer is updated with more image data. If the slice completes a picture, the picture is returned to the application. Otherwise, further calls to vc1DEC_DecodeField() or vc1DEC_DecodeSlice() may be required. This is indicated by the function's return value.

5.5.1.8 vc1DEC_DecodeField()

This function is called when a field BDU is received. It decodes the image data in the field, and updates the decoder's internal picture buffer. If the field completes a picture, the picture is returned to the application. Otherwise, further calls to vc1DEC_DecodeField() or vc1DEC_DecodeSlice() may be required. This is indicated by the function's return value.

5.5.1.9 vc1DEC_UpdateBuffers()

This function is called to update the Hypothetical Reference Decoder model within the decoder at the end of decoding a frame. The function will return vc1_ResultHrdUnderflow or vc1_ResultHrdOverflow if any of the HRD buffers underflow or overflow respectively.

5.5.2 File handling

The application must provide bitstream reading and YUV buffer output. See the reference code for an example of how to update the decoder library bitstream information.

6 DECODER APPLICATION

6.1 Access to bitstream information

The decoder library parses many informational items from the bitstream which are potentially of interest to the calling application, but are not used within the decoder library itself. The library stores this information within the allocated state information. If the application wishes to read particular values, it can do so via the allocated state structure:

- `pState-> sSeqParams` provides the sequence layer and entry point layer information
- `pState-> sPicture` provides information on each output picture.

6.2 Option processing

The decoder requires a single command line parameter when run. This parameter is the name of a simple text file that defines all the particular options used by that run of the program. The option file can define options for use by the decoder only, or other associated tools – simple conditional inclusion allows tool-specific options to be parsed only when appropriate.

This code is not part of the operation defined by the standard, so the modules do not use the `vc1` prefix.

6.2.1 Option file format

Option lines are name and value pairs, with the name separated from the value by a colon (‘`:`’). For options controlling arrays, an index in square brackets, such as [0], can be added after the option name. Comments are allowed; any text following a ‘`#`‘ or ‘`;`’ character will be ignored (unless it is recognised by the conditional processing described below). Case is ignored throughout. Whitespace is also ignored throughout (except for the values for string-type options, which have leading and trailing whitespace removed).

Conditional processing recognises only `#if` and `#endif` directives. These cannot be nested. The condition consists of the tool name, or a list of tool names separated by the C logical-OR operator (‘`||`’).

Here is a simple option file to demonstrate the syntax.

```
# Options for SMPTE VC-1 reference decoder.
# Comments in this file can start with # or ; at any point in the line
; Blank lines are allowed.

#if decoder
DebugMask      : 0x80000000         # See debug.h for bits to set/clear
#endif

Picture width  : 176
Picture height : 144
```

```
Bitstream File : testseq.bits
Level          : Low
Output YUV     : testout.yuv
```

6.2.2 Option types

Each option has a type. The types in use are:

UWORD32	the option can take any unsigned 32-bit value (unless otherwise restricted).
WORD32	the option can take any signed 32-bit value (unless otherwise restricted).
UHWD16	the option can take any unsigned 16-bit value (unless otherwise restricted).
UBYTE8	the option can take any unsigned 8-bit value (unless otherwise restricted).
Enum	the option can take one value from a predetermined list.
String	the option can accept a text value, usually checked for some particular syntax.
Flag	the option is a Boolean value, which can be specified as 0, 1, Yes or No. The value ‘Yes’ sets the flag to 1, the value ‘No’ sets it to 0.

Numeric options can accept values in decimal, or hexadecimal (indicated by a 0x prefix).

6.2.3 Available options

6.2.3.1 BitstreamFile

Type: String

Value: valid filename

Function: Sets the name of the file read by the decoder.

6.2.3.2 ChainAfter

Type: UWORD32

Value: 0 – 0xFFFFFFFF

Function: Sets the number of frames to decode before reading in the next option file. A value of 0 means chaining of options will happen immediately before the next frame is decoded (and so can be used to suppress debug output during sequence layer parsing).

6.2.3.3 ChainOptions

Type: String

Value: valid filename

Function: Sets the name of the file to be read if the ChainAfter count expires.

Note Not all options can meaningfully be reset in a chained option file. Updating some values will cause faulty behaviour of the decoder. This option is primarily intended for debugging assistance: verbose debug output can be suppressed until the frame demonstrating a problem.

6.2.3.4 DebugMask

Type: UWORD32

Value: 0 – 0xFFFFFFFF

Function: Sets the mask controlling debug output in debug builds. All debug output is marked as being within one or more zones. Zones are defined within file `shared/vc1debug.h` by the `vc1DEBUG_eZone` enumerated type. To enable output from a particular zone, set the relevant bit in the option value. The zones are:

Zone name	Bit to set	Output enabled
vc1DEBUG_API	0x00000001	API debug (initialization etc)
vc1DEBUG_FRAME	0x00000002	Frame debug (print frame number/type)
vc1DEBUG_SEQ	0x00000004	Sequence Layer debug
vc1DEBUG_PIC	0x00000008	Picture Layer debug
vc1DEBUG_SLICE	0x00000010	Slice Layer debug
vc1DEBUG_MB	0x00000020	Macroblock layer debug
vc1DEBUG_BLK	0x00000040	Block layer debug
vc1DEBUG_RC	0x00000080	Rate control/hypothetical ref decoder
vc1DEBUG_CMP	0x00000100	Display run, level, mv values
vc1DEBUG_BIT	0x00000200	Display low level bitstream data
vc1DEBUG_BITPL	0x00000400	Display bitplane coding debug
vc1DEBUG_PDCAC	0x00000800	Display DCAC predicted quantized coefficients
vc1DEBUG_DCAC	0x00001000	Display DCAC prediction
vc1DEBUG_QUANT	0x00002000	Display quantized coefficients
vc1DEBUG_TRANS	0x00004000	Display transformed coefficients
vc1DEBUG_DBLK	0x00008000	Display difference image block (not transformed)
vc1DEBUG_PBLK	0x00010000	Display motion predicted image block
vc1DEBUG_RBLK	0x00020000	Display reconstructed block
vc1DEBUG_SMOOTH	0x00040000	Display overlap smoothing result
vc1DEBUG_DEBLK	0x00080000	Display deblocking filter result
vc1DEBUG_IBLK	0x00100000	Display raw input block
vc1DEBUG_ZZ	0x00200000	Display zigzagged coefficients
vc1DEBUG_MV	0x00400000	Display motion vectors
vc1DEBUG_REFPICT	0x00800000	Dump reference pictures to a file
vc1DEBUG_ME	0x01000000	Display motion estimation process

vc1DEBUG_MBSUM	0x02000000	Display block type summary for each macroblock
vc1DEBUG_PADIC	0x04000000	Display padding and intensity compensation
vc1DEBUG_HRD	0x08000000	Display Hypothetical Reference Decoder status
vc1DEBUG_ENT	0x10000000	Display entry point layer debug
vc1DEBUG_OPTIONS	0x40000000	Option parsing
vc1DEBUG_COVERAGE	0x80000000	Coverage output for automated testing

6.2.3.5 Level

Type: Enum

Value: Low, Medium, High, L0, L1, L2, L3, L4 or Unknown

Function: Defines the level within the profile. It is not necessary to specify this option for advanced profile bitstreams. If the level is specified as Unknown, then the decoder will select the lowest level that the coded size will fit into.

6.2.3.6 OutputYUV

Type: String

Value: a valid filename

Function: Sets the name of the file that the decoder output will be written to. Set to `stdout` to obtain YUV data on standard output.

6.3 File handling

The decoder top-level module contains file handling. This code is not part of the operation defined by the standard, so the module does not use the `vc1` prefix. The code handles the following file formats:

Format	Usage
RAW bitstream	Default input format for the decoder. This format is the RCV v2 raw format developed by Microsoft. For format details see the RCV v2 format information in the conformance specification [3].
ELEMENTARY bitstream	Default format for Advanced profile bitstreams. The decoder will inspect the supplied file to determine if it is in ELEMENTARY format. This file format consists of a sequence of Encapsulated Bitstream Data Units (EBDU), as defined in the standard [1].
YUV	This is the output format for the decoder. Each frame consists of the Y values as an array of bytes, followed by the U and V byte arrays. The data is in 4:2:0 format.

The decoder inspects the supplied file to determine the file format.

It is intended that all implementation details of file handling should be handled within the `decfile` module. Adding alternative file formats should involve changes only to this code (other than possibly adding an option to select the new format).

7 REFERENCE SECTION

The following sections contain more detailed documentation on selected components of the decoder. These sections have been generated directly from the source. They do not exhaustively document all features of the code – they are intended only to cover the major features. See the source itself for any items not covered here. There are two automatically generated sections:

- documentation of the major data-structure types used by the decoder
- file-by-file documentation of functions and other elements

8 VC-1 DATA STRUCTURE DOCUMENTATION

8.1 RDOPTS_sOptionDefinition Struct Reference

`#include <rdopts.h>`

Data Fields

- RDOPTS_eType **eType**
- const char * **Name**
- void * **Value**
- **WORD32 MaxIndex**
- **vc1_eResult**(* **rValidateString**)(char *)
- const void * **pDummy**
- const RDOPTS_sWord32Range * **pWRange**
- const RDOPTS_sUWord32Range * **pURange**
- const RDOPTS_sFloatRange * **pFRange**
- const RDOPTS_sEnumeratorValue * **pEnumeratorValue**

Detailed Description

Top-level option specifier record.

Field Documentation

RDOPTS_eType eType

Type of value (byte, word, string, ...)

WORD32 MaxIndex

Maximum array index: 0 means scalar value

const char* Name

Option name

const void* pDummy

The structure starts with a void pointer, because static initialisation uses the first field to determine the acceptable values. A (void *) means that as long as all other union elements are pointers, they can be initialised.

const RDOPTS_sEnumeratorValue* pEnumeratorValue

Enumerable values

const RDOPTS_sFloatRange* pFRange

Range limits for FLOAT data

const RDOPTS_sUWord32Range* pURange

Range limits for UWORD32 data

const RDOPTS_sWord32Range* pWRange

Range limits for WORD32 data

vc1_eResult(* rValidateString)(char *)

String parameters are optionally validated by calling a function. A NULL pointer means no validation will be done. This would be just another member of the above union, but standard C does not allow conversion of function pointers to void *.

void* Value

Pointer to (untyped) value to update

8.2 vc1_sBFraction Struct Reference

`#include <vc1types.h>`

Data Fields

- **UBYTE8 Numerator**

- **UBYTE8 Denominator**
- **UBYTE8 ScaleFactor**

Detailed Description

B Fraction numerator and denominator structure.

Field Documentation

UBYTE8 Denominator

BFraction denominator

UBYTE8 Numerator

BFraction numerator

UBYTE8 ScaleFactor

Approximated Numerator*256/Denominator

8.3 vc1_sBlk Struct Reference

```
#include <vc1types.h>
```

Data Fields

- **vc1_eBlkType eBlkType**
- **FLAG Coded**
- union {
- **vc1_sBlkIntra sIntra**
- **vc1_sBlkInter sInter**
- } **u**

Detailed Description

Structure defining the state required for a block.

Field Documentation

FLAG Coded

Non zero AC coefficients for Intra, non zero AC/DC for Inter

vc1_eBlkType eBlkType

Block type

vc1_sBlkInter sInter

Inter block state information

vc1_sBlkIntra sIntra

Intra block state information

union { ... } u

Intra/Inter union

8.4 vc1_sBlkInter Struct Reference

`#include <vc1types.h>`

Data Fields

- vc1_NumZeroCoef **NZC** [4]
- **vc1_sMotion sMotion** [2]

Detailed Description

Structure defining the state required for inter blocks.

Field Documentation

vc1_NumZeroCoef NZC[4]

NUMZERO and NUMCOEF for sub-blocks (includes DC)

vc1_sMotion sMotion[2]

Forward and Backward motion parameters

8.5 vc1_sBlkIntra Struct Reference

`#include <vc1types.h>`

Data Fields

- vc1_NumZeroCoef **NZC**
- **HWD16 DC**
- **HWD16 ACTop** [7]
- **HWD16 ACLeft** [7]
- **HWD16 SmoothRows** [16]

Detailed Description

Structure defining the state required for intra blocks.

Field Documentation

HWD16 ACLeft[7]

Quantized AC left column for prediction

HWD16 ACTop[7]

Quantized AC top row for prediction

HWD16 DC

Quantized DC for prediction

vc1_NumZeroCoef NZC

NUMZERO and NUMCOEF (excludes DC)

HWD16 SmoothRows[16]

Bottom two rows kept for overlap smoothing

8.6 vc1_sComponent Struct Reference

`#include <vc1types.h>`

Data Fields

- **UBYTE8** * **pData**
- int **Bpl**

Detailed Description

Defines the format of a single image component.

Field Documentation

int Bpl

Number of bytes per line

UBYTE8* pData

Pointer to raster scan data

8.7 vc1_sField Struct Reference

`#include <vc1types.h>`

Data Fields

- **vc1_ePictureType ePictureType**
- vc1_eCondOver **eCondOver**

- **vc1_eQuantMode eQuantMode**
- vc1_eMVRange **eMVRange**
- **vc1_eMVMode eMVMode**
- **vc1_eBlkType eBlkType**
- **FLAG PostProcess**
- **FLAG DMVExtendX**
- **FLAG DMVExtendY**
- **UBYTE8 NumRef**
- **UBYTE8 RefField**
- **UBYTE8 MVCodingTable**
- **UBYTE8 MBModeTable**
- **UBYTE8 BP2MVTable**
- **UBYTE8 BP4MVTable**
- **UBYTE8 CBPCodingTable**
- **UBYTE8 ACCodingSetIntra**
- **UBYTE8 ACCodingSetInter**
- **UBYTE8 DCCodingSet**
- **UHWD16 SliceRows**

Detailed Description

Defines the format of a single field.

This structure contains information that can be local to one of the two Fields in an Interlaced Field Picture.

Field Documentation

UBYTE8 ACCodingSetInter

Range 0-2, Coding set to use for Inter or for Cb,Cr Intra

UBYTE8 ACCodingSetIntra

Range 0-2, Coding set to use for Y Intra

UBYTE8 BP2MVTable

Range 0-3, Block pattern 2MV table

UBYTE8 BP4MVTable

Range 0-3, Block pattern 4MV table

UBYTE8 CBPCodingTable

Range 0-3, Inter CBP variable length code table

UBYTE8 DCCodingSet

Range 0-1, Coding set to use for DC

FLAG DMVExtendX

Extend X DMV range

FLAG DMVExtendY

Extend Y DMV range

vc1_eBlkType eBlkType

Transform type; one of Inter8x8,8x4,4x8,4x4, or Any

vc1_eCondOver eCondOver

Conditional overlap mode

vc1_eMVMode eMVMode

Motion vector mode

vc1_eMVRange eMVRange

Motion vector range

vc1_ePictureType ePictureType

I, P, B or BI

vc1_eQuantMode eQuantMode

Quantization mode

UBYTE8 MBModeTable

Range 0-7, MB mode variable length code table

UBYTE8 MVCodingTable

Range 0-7, Motion vector variable length code table

UBYTE8 NumRef

Number of reference fields 0=one 1=two

FLAG PostProcess

Post processing flag

UBYTE8 RefField

Reference field 0=last 1=last-but-one

UHWD16 SliceRows

Rows per slice (0=slices not used)

8.8 vc1_sHrdState Struct Reference

`#include <vc1types.h>`

Data Fields

- **UBYTE8 NumLeakyBuckets**
- **vc1_sLeakyBucket sLeakyBucket** [VC1_MAX_HRD_NUM_LEAKY_BUCKETS]

Detailed Description

Structure defining the state of the hypothetical reference decoder.

Field Documentation

UBYTE8 NumLeakyBuckets

Buckets (0 if none specified)

vc1_sLeakyBucket sLeakyBucket[VC1_MAX_HRD_NUM_LEAKY_BUCKETS]

Per-bucket information

8.9 vc1_sImagePosition Struct Reference

`#include <vc1types.h>`

Data Fields

- **UWORD32 Width**
- **UWORD32 Height**
- **vc1_sRectangle sImageRectangle**
- **vc1_sRectangle sPadFromRectangle**
- **vc1_sRectangle sPadToRectangle**

Detailed Description

A structure to hold the rectangles to control padding and cropping.

Field Documentation

UWORD32 Height

Total height of buffer

vc1_sRectangle sImageRectangle

Image rectangle in pels relative to buffer origin

vc1_sRectangle sPadFromRectangle

Rectangle to pad outwards from in pels relative to buffer origin

vc1_sRectangle sPadToRectangle

Rectangle limits to pad outwards to in pels relative to buffer origin

UWORD32 Width

Total width of buffer

8.10 vc1_sIntensityComp Struct Reference

`#include <vc1types.h>`

Data Fields

- **FLAG IntensityCompFlag**
- **UBYTE8 LuminanceScale**
- **UBYTE8 LuminanceShift**

Detailed Description

Intensity compensation information structure.

Field Documentation

FLAG IntensityCompFlag

Intensity Compensation Enable

UBYTE8 LuminanceScale

Intensity Compensation Scale

UBYTE8 LuminanceShift

Intensity Compensation Shift

8.11 vc1_sInterpolate Struct Reference

`#include <vc1types.h>`

Data Fields

- **vc1_sComponent sC**

- **UBYTE8 SizeX**
- **UBYTE8 SizeY**
- **FLAG RndCtrl**

Detailed Description

A structure holding information required by the bilinear and bicubic interpolation functions.

Field Documentation

FLAG RndCtrl

Rounding control for the frame

vc1_sComponent sC

Component from which to obtain the patch to be filtered

UBYTE8 SizeX

X size in pixels of the resulting filtered patch

UBYTE8 SizeY

Y size in pixels of the resulting filtered patch

8.12 vc1_sLeakyBucket Struct Reference

`#include <vc1types.h>`

Data Fields

- HRDVALUE **Rate**
- HRDVALUE **Buffer**
- HRDVALUE **Fullness**
- **UWORD32 FullFraction**
- **UWORD32 FullDenominator**

Detailed Description

Hypothetical reference decoder information.

Remarks:

Note that the decoder can always round up the Rate, Buffer size and Fullness parameters. Note that 0 <= FullFraction/FullDenominator < 1 bit.

Field Documentation

HRDVALUE Buffer

Buffer size in bits

UWORD32 FullDenominator

Denominator of fractional bit buffer fullness count

UWORD32 FullFraction

Numerator of fractional bit buffer fullness count

HRDVALUE Fullness

Buffer fullness in complete bits

HRDVALUE Rate

Maximum bit rate in bits per second

8.13 vc1_sLevelLimit Struct Reference

`#include <vc1types.h>`

Data Fields

- **UWORD32 MBs**
- **UWORD32 MBf**
- **UWORD32 Rmax**
- **UWORD32 Bmax**
- vc1_eMVRange **eMVRange**

Detailed Description

A structure holding the limits for a given profile and level.

Field Documentation

UWORD32 Bmax

Maximum buffer size in multiples of 16kbits

vc1_eMVRange eMVRange

Motion vector range allowed

UWORD32 MBf

Maximum macroblocks per frame

UWORD32 MBs

Maximum macroblocks per second

UWORD32 Rmax

Maximum peak transmission rate in kpbs

8.14 vc1_sMB Struct Reference

`#include <vc1types.h>`

Data Fields

- **vc1_eMBType eMBType**
- **vc1_eACPred eACPred**
- **vc1_eBlkType eBlkType**
- **FLAG OverlapFilter**
- **FLAG Skipped**
- **UBYTE8 CBPCY**
- **UBYTE8 MVBP**
- **vc1_sQuant sQuant**

- **vc1_sBlk sBlk** [VC1_BLOCKS_PER_MB]

Detailed Description

A structure to hold macroblock data.

Field Documentation

UBYTE8 CBPCY

Coded block pattern, where:

- bit 5 set means Y0 is coded
- bit 4 set means Y1 is coded
- bit 3 set means Y2 is coded
- bit 2 set means Y3 is coded
- bit 1 set means Cb is coded
- bit 0 set means Cr is coded

vc1_eACPred eACPred

AC prediction status

vc1_eBlkType eBlkType

One of 8x8,8x4,4x8,4x4, or Any (Any=block based choice)

vc1_eMBType eMBType

Macroblock type

UBYTE8 MVBP

Motion vector block pattern, with:

- bit3 set if dmv!=0 for Y0
- bit2 set if dmv!=0 for Y1
- bit1 set if dmv!=0 for Y2
- bit0 set if dmv!=0 for Y3

FLAG OverlapFilter

Overlap filter active for this macroblock

vc1_sBlk sBlk[VC1_BLOCKS_PER_MB]

Block level information

FLAG Skipped

Indicated macroblock is motion predicted only

vc1_sQuant sQuant

Macroblock Quantizer information

8.15 vc1_sMotion Struct Reference

`#include <vc1types.h>`

Data Fields

- **vc1_eHybridPred eHybridPred**
- **vc1_sMV sMV**
- **vc1_sMV sDMV**

Detailed Description

Structure defining motion vector and associated parameters.

Field Documentation

vc1_eHybridPred eHybridPred

Hybrid Prediction mode

vc1_sMV sDMV

Differential motion vector (X,Y) in 1/4 pel units

vc1_sMV sMV

Motion vector

8.16 vc1_sMV Struct Reference

`#include <vc1types.h>`

Data Fields

- **HWD16 X**
- **HWD16 Y**
- **FLAG BottomField**

Detailed Description

Structure to hold a motion vector.

Field Documentation

FLAG BottomField

0=TopField 1=BottomField

HWD16 X

X component of the motion vector (offset to target)

HWD16 Y

Y component of the motion vector (offset to target)

8.17 vc1_sPanScanParams Struct Reference

`#include <vc1types.h>`

Data Fields

- **FLAG PanScanPresent**
- **vc1_sPanScanWindow sPanScanWindow** [VC1_MAX_PAN_SCAN_WINDOWS]

Detailed Description

Structure holding all the pan and scan window information.

Field Documentation

FLAG PanScanPresent

PS present

vc1_sPanScanWindow sPanScanWindow[VC1_MAX_PAN_SCAN_WINDOWS]

Per-window information

8.18 vc1_sPanScanWindow Struct Reference

```
#include <vc1types.h>
```

Data Fields

- **UWORD32 HOffset**
- **UWORD32 VOffset**
- **UHWD16 Width**
- **UHWD16 Height**

Detailed Description

Structure holding individual Pan Scan Window information.

Field Documentation

UHWD16 Height

Height in pixels

UWORD32 HOffset

Horizontal offset in pixels

UWORD32 VOffset

Vertical offset in pixels

UHWD16 Width

Width in pixels

8.19 vc1_sPicture Struct Reference

```
#include <vc1types.h>
```

Data Fields

- **UWORD32 Frame**
- **vc1_ePictureFormat ePictureFormat**
- **vc1_sComponent sY**
- **vc1_sComponent sU**
- **vc1_sComponent sV**
- **vc1_sField sField** [2]
- **vc1_ePictureRes ePicRes**
- **FLAG TFF**
- **FLAG RFF**
- **FLAG RangeReduction**
- **FLAG INTERPFRM**
- **FLAG UVSAMP**
- **UBYTE8 RPTFRM**
- **vc1_sPanScanParams sPanScanParams**
- vc1_ePostProcessing **ePostProcessing**

Detailed Description

A Picture is the basic image unit decoded by the library. A Picture can be one of the following formats:

- A Progressive Frame
- An Interlaced Top Field
- An Interlaced Bottom Field
- An Interlaced Frame

Field Documentation

vc1_ePictureRes ePicRes

Picture resolution index

vc1_ePictureFormat ePictureFormat

ProgressiveFrame, InterlacedField/Frame

vc1_ePostProcessing ePostProcessing

Out of loop post processing mode

UWORD32 Frame

Frame number modulo (1<<32)

FLAG INTERPFRM

Frame interpolation hint

FLAG RangeReduction

Range reduction used

FLAG RFF

Repeat First Field flag

UBYTE8 RPTFRM

Repeat Frame Count field

vc1_sField sField[2]

Field information, First then Second

vc1_sPanScanParams sPanScanParams

Pan scan window coordinates

vc1_sComponent sU

U/Cb chrominance values

vc1_sComponent sV

V/Cr chrominance values

vc1_sComponent sY

Y luminance values

FLAG TFF

Top Field First flag

FLAG UVSAMP

UV sampling format

8.20 vc1_sPosition Struct Reference

`#include <vc1types.h>`

Data Fields

- **vc1_ePictureType ePictureType**
- **vc1_ePictureFormat ePictureFormat**
- **vc1_eProfile eProfile**
- **vc1_eMVMode eMVMode**
- vc1_eMVRange **eMVRange**
- **FLAG BottomField**
- **FLAG SecondField**
- **vc1_sMB** * **pCurMB**
- **vc1_sMB** * **pStartMB**
- vc1_sMotionHist * **pMVHist**
- **UWORD32 SizeMB**
- **UHWD16 X**
- **UHWD16 Y**
- **UHWD16 SliceY**
- **UHWD16 WidthMB**
- **UHWD16 HeightMB**
- **UWORD32 CodedWidth**
- **UWORD32 CodedHeight**
- **UWORD32 DisplayWidth**
- **UWORD32 DisplayHeight**
- **UBYTE8 PQuant**
- **UBYTE8 BFraction**
- **UBYTE8 NumRef**
- **UBYTE8 RefField**

- **UBYTE8 IntraBias**
- **UBYTE8 RangeYScale**
- **UBYTE8 RangeUVScale**
- **FLAG FastUVMC**
- **vc1_ePictureRes ePictureRes**
- **vc1_sReferencePicture** * **pReferenceOld**
- **vc1_sReferencePicture** * **pReferenceNew**
- **vc1_sReferencePicture** * **pReferenceB**
- **vc1_sReferencePicture** * **pReferenceNoIC**
- **vc1_sScaleMV pScaleMV** [2]
- **HWD16 pSmooth** [6][64]

Detailed Description

Current position structure describing the macroblock being processed and the slice, field, and picture it lies in.

Field Documentation

UBYTE8 BFraction

BFRACTION syntax element

FLAG BottomField

0=Top Field, 1=Bottom Field

UWORD32 CodedHeight

Height in pixels of coded picture

UWORD32 CodedWidth

Width in pixels of coded picture

UWORD32 DisplayHeight

Height in pixels of display picture

UWORD32 DisplayWidth

Width in pixels of display picture

vc1_eMVMode eMVMode

Motion vector mode for this picture

vc1_eMVRange eMVRange

Motion vector range setting for this picture

vc1_ePictureFormat ePictureFormat

Picture format: Progressive, Interlace Field/frame

vc1_ePictureRes ePictureRes

Picture resolution scale mode

vc1_ePictureType ePictureType

Picture type: I, P, B or BI

vc1_eProfile eProfile

Profile Simple/Main/Advanced

FLAG FastUVMC

Fast U,V motion compensation flag

UHWD16 HeightMB

Height in macroblocks of coded picture

UBYTE8 IntraBias

Bias to add to intra blocks post transform

UBYTE8 NumRef

Number of reference fields-1

vc1_sMB* pCurMB

Pointer to the current macroblock

vc1_sMotionHist* pMVHist

Current position in motion vector history buffer

UBYTE8 PQuant

Picture quantizer

vc1_sReferencePicture* pReferenceB

Reconstructed B picture

vc1_sReferencePicture* pReferenceNew

Pointer to new/current I/P

vc1_sReferencePicture* pReferenceNoIC

Backup copy of reference before intensity compensation applied

vc1_sReferencePicture* pReferenceOld

Pointer to old I/P reference picture

vc1_sScaleMV pScaleMV[2]

MV scaling (forward, backward)

HWD16 pSmooth[6][64]

Overlap smoothing macroblock history (left)

vc1_sMB* pStartMB

Pointer to start of macroblock circular buffer

UBYTE8 RangeUVScale

UV scaling factor times 8

UBYTE8 RangeYScale

Y scaling factor times 8

UBYTE8 RefField

Reference field when NumRef==0

FLAG SecondField

0=First Field, 1=Second Field

UWORD32 SizeMB

Circular buffer size in macroblocks

UHWD16 SliceY

Y macroblock offset of slice in picture

UHWD16 WidthMB

Width in macroblocks of coded picture

UHWD16 X

X macroblock offset in current slice

UHWD16 Y

Y macroblock offset in current slice

8.21 vc1_sQuant Struct Reference

`#include <vc1types.h>`

Data Fields

- **UBYTE8 Quant**
- **UBYTE8 HalfStep**
- **FLAG NonUniform**

Detailed Description

Quantizer structure.

Field Documentation

UBYTE8 HalfStep

Quantizer Half Step value 0 or 1

FLAG NonUniform

Selects Uniform/NonUniform Quantizer

UBYTE8 Quant

Quantizer Step in the range 1 to 31

8.22 vc1_sRectangle Struct Reference

`#include <vc1types.h>`

Data Fields

- **WORD32 XMin**
- **WORD32 XMax**
- **WORD32 YMin**
- **WORD32 YMax**

Detailed Description

Structure defining a rectangular area.

Field Documentation

WORD32 XMax

Maximum x coordinate (inclusive)

WORD32 XMin

Minimum x coordinate

WORD32 YMax

Maximum y coordinate (inclusive)

WORD32 YMin

Minimum y coordinate

8.23 vc1_sReferencePicture Struct Reference

`#include <vc1types.h>`

Data Fields

- **FLAG Valid**
- **UBYTE8 Padded**
- **UBYTE8 RangeYScale**
- **UBYTE8 RangeUVScale**
- **UBYTE8 RefDist**
- **UWORD32 Frame**
- **FLAG TFF**
- **FLAG RFF**
- **vc1_sPanScanParams sPanScanParams**
- **FLAG FrameInterpolationHint**
- **FLAG UVSampleMode**
- **UBYTE8 RepeatFrameCount**
- vc1_ePostProcessing **ePostProcessing**
- **UHWD16 CodedWidth**
- **UHWD16 CodedHeight**
- **vc1_ePictureFormat ePictureFormat**
- **vc1_ePictureType ePictureType** [2]
- **vc1_ePadMode ePadMode**
- **vc1_ePictureRes ePictureRes**
- **vc1_sComponent sY**
- **vc1_sComponent sU**
- **vc1_sComponent sV**
- **UBYTE8** * **pImageY**
- **UBYTE8** * **pImageU**
- **UBYTE8** * **pImageV**
- **vc1_sImagePosition sImagePosLuma**
- **vc1_sImagePosition sImagePosChroma**

Detailed Description

A structure to hold reference key frame image information.

Field Documentation

UHWD16 CodedHeight

Height of the image in the buffer

UHWD16 CodedWidth

Width of the image in the buffer

vc1_ePadMode ePadMode

Padding mode to use for this picture

vc1_ePictureFormat ePictureFormat

Format of reference picture

vc1_ePictureRes ePictureRes

Picture resolution scale

vc1_ePictureType ePictureType[2]

Picture type for each reference field

vc1_ePostProcessing ePostProcessing

Out of loop post processing mode

UWORD32 Frame

Frame number modulo (1<<32)

FLAG FrameInterpolationHint

Frame interpolation hint. Not used in the decoding process

UBYTE8 Padded

1=Top field padded, 2=bottom 3=all

UBYTE8* pImageU

Image U top left corner

UBYTE8* pImageV

Image V top left corner

UBYTE8* pImageY

Image Y top left corner

UBYTE8 RangeUVScale

UV scaling factor times 8

UBYTE8 RangeYScale

Y scaling factor times 8

UBYTE8 RefDist

Number of frames between this and the last reference frame

UBYTE8 RepeatFrameCount

Repeated frame count

FLAG RFF

Repeat First Field flag

vc1_sImagePosition sImagePosChroma

Position of chroma samples in the image buffer

vc1_sImagePosition sImagePosLuma

Position of luma samples in the image buffer

vc1_sPanScanParams sPanScanParams

Pan and scan window coordinates

vc1_sComponent sU

U/Cb chrominance values

vc1_sComponent sV

V/Cr chrominance values

vc1_sComponent sY

Y luminance values

FLAG TFF

Top Field First flag

FLAG UVSampleMode

UV plane sampling mode. Only relevant to interlaced frames

FLAG Valid

Buffer contains valid data

8.24 vc1_sScaleMV Struct Reference

```
#include <vc1types.h>
```

Data Fields

- **UHWD16 Scale**
- **UHWD16 Scale1**
- **UHWD16 Scale2**
- **UHWD16 ScaleZone1X**
- **UHWD16 ScaleZone1Y**
- **UHWD16 Zone1OffsetX**
- **UHWD16 Zone1OffsetY**
- **FLAG ScaleUpOpp**
- **FLAG BottomField**
- **FLAG Back**

Detailed Description

Structure describing how to scale interlaced motion vectors.

Field Documentation

FLAG Back

0=Forward scale, 1=backward scale

FLAG BottomField

0=Top Field 1=Bottom Field

UHWD16 Scale

Down scale factor * 256

UHWD16 Scale1

Up scale factor * 256 if in Zone1

UHWD16 Scale2

Up scale factor * 256 if not in Zone1

FLAG ScaleUpOpp

0=ScaleDownForOpposite 1=ScaleUpForOpposite

UHWD16 ScaleZone1X

Zone1 X size

UHWD16 ScaleZone1Y

Zone1 Y size

UHWD16 Zone1OffsetX

Zone1 X offset

UHWD16 Zone1OffsetY

Zone1 Y offset

8.25 vc1_sSequenceLayer Struct Reference

`#include <vc1types.h>`

Data Fields

- **vc1_eProfile eProfile**
- **UHWD16 MaxCodedWidth**
- **UHWD16 MaxCodedHeight**
- **UHWD16 CodedWidth**
- **UHWD16 CodedHeight**
- **UHWD16 DisplayWidth**
- **UHWD16 DisplayHeight**
- **UHWD16 AspectWidth**
- **UHWD16 AspectHeight**
- **vc1_eLevel eLevel**
- **FLAG Interlace**
- **UWORD32 FrameRateNumerator**
- **UHWD16 FrameRateDenominator**
- **FLAG ColorFormatIndicatorFlag**
- vc1_eChromaFormat **eChromaFormat**
- vc1_eColorPrimaries **eColorPrimaries**
- vc1_eTransChar **eTransChar**
- vc1_eMatrixCoefficients **eMatrixCoefficients**
- **vc1_sHrdState sHrdInitialState**
- **FLAG LoopFilter**
- **FLAG MultiResCoding**
- **FLAG FastUVMC**
- **FLAG ExtendedMV**
- **FLAG ExtendedDMV**
- **UBYTE8 DQuant**
- **FLAG VSTransform**
- **FLAG OverlappedTransformFlag**
- **FLAG SyncmarkerFlag**
- **FLAG RangeRedFlag**
- **UBYTE8 MaxBFrames**
- **vc1_eQuantizer eQuantizer**
- **FLAG PostProcessingFlag**
- **FLAG FrameCounterFlag**
- **FLAG PullDownFlag**
- **UBYTE8 QFrameRateForPostProc**
- **UBYTE8 QBitRateForPostProc**

- **FLAG PanScanFlag**
- **FLAG ReservedRTMFlag**
- **FLAG FrameInterpolationFlag**
- **UBYTE8 RangeYScale**
- **UBYTE8 RangeUVScale**
- **UBYTE8 NumPanScanWin**
- **FLAG BrokenLink**
- **FLAG ClosedEntry**
- **FLAG RefDistFlag**
- **FLAG FrameUserDataFlag**

Detailed Description

Sequence and Layer parameters.

Field Documentation

UHWD16 AspectHeight

Aspect height or 0 if not specified

UHWD16 AspectWidth

Aspect width or 0 if not specified

FLAG BrokenLink

Non-zero if the broken link flag is set in the bitstream

FLAG ClosedEntry

Non-zero if the closed entry flag is set in the bitstream

UHWD16 CodedHeight

Coded height or 0 if not specified

UHWD16 CodedWidth

Coded width or 0 if not specified

FLAG ColorFormatIndicatorFlag

See standard

UHWD16 DisplayHeight

Display height or 0 if not specified

UHWD16 DisplayWidth

Display width or 0 if not specified

UBYTE8 DQuant

See standard

vc1_eChromaFormat eChromaFormat

See standard

vc1_eColorPrimaries eColorPrimaries

See standard

vc1_eLevel eLevel

See standard

vc1_eMatrixCoefficients eMatrixCoefficients

See standard

vc1_eProfile eProfile

See standard

vc1_eQuantizer eQuantizer

See standard

vc1_eTransChar eTransChar

See standard

FLAG ExtendedDMV

See standard

FLAG ExtendedMV

See standard

FLAG FastUVMC

See standard

FLAG FrameCounterFlag

See standard

FLAG FrameInterpolationFlag

See standard

UHWD16 FrameRateDenominator

1000, 1001, or 32 (0 if not specified)

UWORD32 FrameRateNumerator

0 if not specified

FLAG FrameUserDataFlag

Non-zero if frame user data is present in the bitstream

FLAG Interlace

See standard

FLAG LoopFilter

See standard

UBYTE8 MaxBFrames

See standard

UHWD16 MaxCodedHeight

Maximum coded height

UHWD16 MaxCodedWidth

Maximum coded width

FLAG MultiResCoding

See standard

UBYTE8 NumPanScanWin

Number of pan scan windows

FLAG OverlappedTransformFlag

See standard

FLAG PanScanFlag

See standard

FLAG PostProcessingFlag

See standard

FLAG PullDownFlag

See standard

UBYTE8 QBitRateForPostProc

See standard

UBYTE8 QFrameRateForPostProc

See standard

FLAG RangeRedFlag

See standard

UBYTE8 RangeUVScale

Scale value times 8

UBYTE8 RangeYScale

Scale value times 8

FLAG RefDistFlag

Non-zero if the ref dist flag is set in the bitstream

FLAG ReservedRTMFlag

See standard

vc1_sHrdState sHrdInitialState

Initial state of HRD

FLAG SyncmarkerFlag

See standard

FLAG VSTransform

See standard

8.26 vc1DEC3DH_sRunLevel Struct Reference

```
#include <vc1dec3dh.h>
```

Detailed Description

Structure to hold a run, level and last triple.

Remarks:

This structure is used as an array element, where the index value is the code for the run, level and last triple.

8.27 vc1DEC_sBitplane Struct Reference

`#include <vc1dec.h>`

Data Fields

- **FLAG * pBitplane**
- **FLAG RawMode**
- **UWORD32 Position**

Detailed Description

Structure to contain bitplane coding information.

Field Documentation

FLAG* pBitplane

Pointer to bitplane table. NULL indicates RAW mode

UWORD32 Position

Index of the next bit in the bitplane

FLAG RawMode

True if raw mode is used in this bitplane

8.28 vc1DEC_sBitstream Struct Reference

`#include <vc1dec.h>`

Data Fields

- **UBYTE8 * pBitBuffer**
- **UBYTE8 * pEndByte**
- **UBYTE8 EndBitsValid**
- **UBYTE8 EncapsulatedIDU**
- **UWORD32 ZeroRun**
- **UWORD32 BitCounter**
- **UWORD32 Buffer0**
- **UWORD32 Buffer1**

- **UBYTE8 BitsUsed**
- **UBYTE8 BitsValid**

Detailed Description

Structure containing the fields defining a bitstream.

Field Documentation

UWORD32 BitCounter

Number of bits read from bitstream, for use with HRD

UBYTE8 BitsUsed

Number of used bits in the buffer 0 to 31

UBYTE8 BitsValid

Number of valid bits in the buffer 0 to 64

UWORD32 Buffer0

First half of 64-bit holding buffer

UWORD32 Buffer1

Second half of 64-bit holding buffer

UBYTE8 EncapsulatedIDU

Non-zero if the independently decodable units are encapsulated

UBYTE8 EndBitsValid

Number of bits valid in the last used byte 1 to 8

UBYTE8* pBitBuffer

Pointer to byte containing the next bit in the bitstream

UBYTE8* pEndByte

Pointer to the byte after the last used byte of the bitstream buffer

UWORD32 ZeroRun

Number of consecutive zeroes read from the bitstream

8.29 vc1DEC_sDecoderConfiguration Struct Reference

```
#include <vc1dec.h>
```

Data Fields

- **UHWD16 MaxCodedWidth**
- **UHWD16 MaxCodedHeight**
- **vc1_eProfile eProfile**
- **vc1_eLevel eLevel**
- **UWORD32 FrameRateNumerator**
- **UHWD16 FrameRateDenominator**

Detailed Description

Structure containing configuration information for the decoder. This structure contains information available to the application from demultiplexing the container format.

Field Documentation

vc1_eLevel eLevel

Highest level supported by the decoder

vc1_eProfile eProfile

Highest profile supported by the decoder

UHWD16 FrameRateDenominator

Explicit default framerate denominator

- 1000, 1001, or 32 (0 if not specified)

UWORD32 FrameRateNumerator

Explicit default framerate numerator (0 if not specified)

UHWD16 MaxCodedHeight

Maximum coded picture height

UHWD16 MaxCodedWidth

Maximum coded picture width

8.30 vc1DEC_sPictureLayerParams Struct Reference

`#include <vc1dec.h>`

Data Fields

- **UBYTE8 FrameCount**
- **vc1_ePictureType ePictureType** [2]
- **UBYTE8 BufferFullness**
- **UBYTE8 PQIndex**
- **vc1_eQuantizer eQuantizer**
- **UBYTE8 PQuant**
- **FLAG HalfQPStep**
- **UBYTE8 FrameTransformACCodingSetIndex**
- **FLAG IntraTransformDCTable**
- **UBYTE8 TemporalRefFrameCounter**
- **FLAG TopFieldFirst**
- **FLAG RepeatFirstField**
- **FLAG UVSampleMode**
- vc1_ePostProcessing **ePostProcessing**
- **vc1_eQuantMode eQuantMode**
- **UBYTE8 AltPQuant**
- **vc1_sInterpolate sInterpolate**
- const **vc1DEC_sVLCCode** * **pMotionVectorTable**
- const **vc1DEC_sVLCCode** * **pCodedBlockPatternTable**
- **FLAG MBTransformTypeFlag**
- **vc1_eBlkType eFrameTransformType**
- **UBYTE8 RepeatFrameCount**
- **FLAG FrameInterpolationHint**
- vc1_eCondOver **eConditionalOverlap**
- **vc1_sPanScanParams sPanScanParams**
- **FLAG DQuantFrame**
- **vc1DEC_sBitplane sBPPredictAC**
- **vc1DEC_sBitplane sBPSkipMB**

- **vc1DEC_sBitplane sBPMotionVectorType**
- **vc1DEC_sBitplane sBPBFrameDirectMode**
- **vc1DEC_sBitplane sBPOverflags**
- **vc1DEC_sBitplane sBPForwardMB**
- **vc1DEC_sBitplane sBPFieldTX**
- **FLAG ExtendHorizDMVRange**
- **FLAG ExtendVertDMVRange**
- const **vc1DEC_sVLCCode** * **pMBModeTable**
- const **vc1DEC_sVLCCode** * **pMB4MVBlockPatternTable**
- const **vc1DEC_sVLCCode** * **pMB2MVBlockPatternTable**
- **vc1_sIntensityComp sIC** [2]

Detailed Description

Structure containing the parameters read from the bitstream for the picture layer. This is derived from the picture layer section of the standard.

Field Documentation

UBYTE8 AltPQuant

See the standard

UBYTE8 BufferFullness

See the standard

FLAG DQuantFrame

Per MB quantisation mode

vc1_eCondOver eConditionalOverlap

See the standard

vc1_eBlkType eFrameTransformType

See the standard

vc1_ePictureType ePictureType[2]

See the standard

vc1_ePostProcessing ePostProcessing

See the standard

vc1_eQuantizer eQuantizer

Per-picture quantizer

vc1_eQuantMode eQuantMode

Quantization mode

FLAG ExtendHorizDMVRange

Extend horizontal differential MV

FLAG ExtendVertDMVRange

Extend vertical differential MV

UBYTE8 FrameCount

See the standard

FLAG FrameInterpolationHint

See the standard

UBYTE8 FrameTransformACCodingSetIndex

See the standard

FLAG HalfQPStep

See the standard

FLAG IntraTransformDCTable

See the standard

FLAG MBTransformTypeFlag

See the standard

const vc1DEC_sVLCCode* pCodedBlockPatternTable

See the standard

const vc1DEC_sVLCCode* pMB2MVBlockPatternTable

See the standard

const vc1DEC_sVLCCode* pMB4MVBlockPatternTable

See the standard

const vc1DEC_sVLCCode* pMBModeTable

See the standard

const vc1DEC_sVLCCode* pMotionVectorTable

See the standard

UBYTE8 PQIndex

See the standard

UBYTE8 PQuant

PQuant value

FLAG RepeatFirstField

See the standard

UBYTE8 RepeatFrameCount

See the standard

vc1DEC_sBitplane sBPBFrameDirectMode

Bitplane for Direct MB

vc1DEC_sBitplane sBPFieldTX

Bitplane for FieldTX MB

vc1DEC_sBitplane sBPForwardMB

Bitplane for Forward MB

vc1DEC_sBitplane sBPMotionVectorType

Bitplane for MV type

vc1DEC_sBitplane sBPOverflags

Bitplane for overlap flags

vc1DEC_sBitplane sBPPredictAC

Bitplane for AC prediction

vc1DEC_sBitplane sBPSkipMB

Bitplane for MB skip

vc1_sIntensityComp sIC[2]

Intensity compensation information for the first and second (if it exists) fields

vc1_sInterpolate sInterpolate

See the standard

vc1_sPanScanParams sPanScanParams

See the standard

UBYTE8 TemporalRefFrameCounter

See the standard

FLAG TopFieldFirst

See the standard

FLAG UVSampleMode

See the standard

8.31 vc1DEC_sState Struct Reference

`#include <vc1dec.h>`

Data Fields

- **vc1_sPosition sPosition**
- int **FrameNum**
- **vc1_sMB** * **pMB**
- int **NumFields**
- int **MaxMBs**
- const **vc1_sLevelLimit** * **pLevelLimit**
- **vc1_sSequenceLayer sSeqParams**
- **vc1DEC_sPictureLayerParams sPicParams**
- **UBYTE8 NotFirstMode3InFrame**
- **UBYTE8 LevelCodeSize**
- **UBYTE8 RunCodeSize**
- **UBYTE8 ZigZagTableIndex**
- **FLAG FirstFrameInStream**
- **FLAG BitplaneCodingUsed**
- **UBYTE8 FirstCodedBlock**
- **vc1_sReferencePicture** * **pCurrentRef**
- vc1_sMotionHist * **pMVHistBuffer**
- **UWORD32 FieldCount**

Detailed Description

Structure containing the state of the decoder.

Field Documentation

FLAG BitplaneCodingUsed

Non-zero if bitplane coding is in use

UWORD32 FieldCount

Number of fields present in the current picture

UBYTE8 FirstCodedBlock

First coded block in current macroblock

FLAG FirstFrameInStream

Non-zero if this is the first frame

int FrameNum

Current frame number

UBYTE8 LevelCodeSize

Level code size for mode 3 escape, per frame

int MaxMBs

Maximum number of macroblocks

UBYTE8 NotFirstMode3InFrame

1 if not first mode 3 escape in frame

int NumFields

Number of fields per frame

vc1_sReferencePicture* pCurrentRef

Pointer to the current reference picture into which blocks are decoded

const vc1_sLevelLimit* pLevelLimit

Limits for current level and profile

vc1_sMB* pMB

Pointer to macroblock data

vc1_sMotionHist* pMVHistBuffer

Pointer to the motion vector history buffer

UBYTE8 RunCodeSize

Run code size for mode 3 escape, per frame

vc1DEC_sPictureLayerParams sPicParams

Data from the current picture layer

vc1_sPosition sPosition

Macroblock position structure

vc1_sSequenceLayer sSeqParams

Data from the sequence layer

UBYTE8 ZigZagTableIndex

Index to select zigzag table

8.32 vc1DEC_sVLCCode Struct Reference

`#include <vc1dec.h>`

Data Fields

- **UWORD32 Bits**
- **UBYTE8 Length**
- **UWORD32 Value**

Detailed Description

Structure for holding a variable length code.

Remarks:

In an array of these, entry 0 has a special meaning:

- Bits = 0;
- Length = Number of codes in the array (Length of array - 1)
- Value = Maximum code length

Field Documentation

UWORD32 Bits

Bit pattern of the code

UBYTE8 Length

Length of the code

UWORD32 Value

Value that the code represents

---

9 VC-1 FILE DOCUMENTATION

9.1 decfile.h File Reference

Functions

- **vc1_eResult DECFILE_InitialiseFrameWriter** (**vc1_sPicture** *pPicture, const **vc1DEC_sState** *pState)
- void **DECFILE_FinaliseFrameWriter** (**vc1_sPicture** *pPicture)
- **vc1_eResult DECFILE_InitialiseBitstreamFileReader** (**vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult DECFILE_AdjustBitstreamBuffering** (**vc1DEC_sBitstream** *pBitstream)
- void **DECFILE_FinaliseBitstreamFileReader** (**vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult DECFILE_ReadBitstreamFile** (**vc1DEC_sBitstream** *pBitstream, **UWORD32** FramesOutput, **UWORD32** FramesDecoded, **UBYTE8** *pStartCodeSuffix)
- **UWORD32 DECFILE_GetHRDBytesRead** (void)
- void **DECFILE_SetHRDBytesRead** (**UWORD32** Bytes)

Detailed Description

These functions provide file IO for the decoder. They are not a part of the actual decoding process. The input functions handle two types of stream: RCV format and ELEMENTARY format. The output functions handle data in 4:2:0 YUV format.

Function Documentation

vc1_eResult DECFILE_AdjustBitstreamBuffering (vc1DEC_sBitstream * *pBitstream*)

Adjust input file buffering to match the profile and level.

Parameters:

pBitstream - the bitstream buffer to adjust

Returns:

- `vc1_ResultOK` on success

void DECFILE_FinaliseBitstreamFileReader (vc1DEC_sBitstream * *pBitstream*)

Free any resources held by the bitstream file reader component.

Parameters:

pBitstream - the bitstream buffer to free

void DECFILE_FinaliseFrameWriter (vc1_sPicture * *pPicture*)

Free any resources held by the frame writer component.

Parameters:

pPicture - a pointer to the picture structure holding buffer pointers

UWORD32 DECFILE_GetHRDBytesRead (void)

Get the number of bytes read from the bitstream, for HRD model

Returns:

Number of bytes read

vc1_eResult DECFILE_InitialiseBitstreamFileReader (vc1DEC_sBitstream * *pBitstream*)

Initialise the bitstream file reader component.

Parameters:

pBitstream - the bitstream buffer to initialise

Returns:

- `vc1_ResultOK` on success

vc1_eResult DECFILE_InitialiseFrameWriter (vc1_sPicture * *pPicture*, const vc1DEC_sState * *pState*)

Initialise the frame writer component, including allocation of frame buffers.

Parameters:

pPicture - a pointer to the picture structure to hold buffer pointers
pState - a pointer to the state structure holding the picture size

Outputs:

- *pPicture* is updated with pointers to the allocated buffers

Returns:

- `vc1_ResultOK` on success

vc1_eResult DECFILE_ReadBitstreamFile (vc1DEC_sBitstream * *pBitstream*, UWORD32 *FramesOutput*, UWORD32 *FramesDecoded*, UBYTE8 * *pStartCodeSuffix*)

Read more data into the bitstream buffer.

Parameters:

pBitstream - the bitstream buffer to refill
FramesOutput - the number of frames written to the output
FramesDecoded - the number of frames read from the input
pStartCodeSuffix - pointer to a byte to receive the start code type for advanced profile

Outputs:

- **pStartCodeSuffix* - start code type if in advanced profile

Returns:

- `vc1_ResultOK` on success

void DECFILE_SetHRDBytesRead (UWORD32 *Bytes*)

Set the number of bytes read from the bitstream, for HRD model

Parameters:

Bytes - Value to set HRDBytesRead. Usually 0.

9.2 decopts.h File Reference

Functions

- **vc1_eResult DECOPTS_ValidateParameters** (void)
- **UWORD32 DECOPTS_CalculateBitstreamBufferSize** (void)
- **vc1_eResult DECOPTS_UpdateOptions** (void)

Variables

- const **RDOPTS_sOptionDefinition DECOPTS_sDecoderOptions** []

Detailed Description

This file provides handling of the input options for the decoder. It is not a part of the actual decoding process.

Function Documentation

UWORD32 DECOPTS_CalculateBitstreamBufferSize (void)

Calculate a bitstream buffer size that will not be overflowed by any frame.

Returns:

the required buffer size in bytes

vc1_eResult DECOPTS_UpdateOptions (void)

Check for any chained option updates. This is useful for debugging: debug output can be disabled until the frame of interest.

vc1_eResult DECOPTS_ValidateParameters (void)

Performs any required verification of parameters not done as part of option parsing.

Returns:

- `vc1_ResultOK` on success

Variable Documentation

const RDOPTS_sOptionDefinition DECOPTS_sDecoderOptions[]

The main table of options for the decoder. This defines all the options accepted by the decoder, and defines what values will be placed in which configuration variables.

9.3 rdopts.h File Reference

Data Structures

- struct **RDOPTS_sEnumeratorValue**
- struct **RDOPTS_sFloatRange**
- struct **RDOPTS_sOptionDefinition**
- struct **RDOPTS_sUWord32Range**
- struct **RDOPTS_sWord32Range**

Enumerations

- enum **RDOPTS_eBitstreamFormatType**

Functions

- **vc1_eResult RDOPTS_ReadOptions** (const char *ToolName, const char *OptionFileName, const **RDOPTS_sOptionDefinition** *pOptionList)
- void **RDOPTS_FreeOptions** (const **RDOPTS_sOptionDefinition** *pOptionList)
- **vc1_eResult RDOPTS_ValidateFrameRate** (char *pValue, **UWORD32** *pNumerator, **UHWD16** *pDenominator)

Detailed Description

This file contains generic option handling, used by application-specific option handling. It is not part of the standard bitstream format handling process.

Enumeration Type Documentation

enum RDOPTS_eBitstreamFormatType

Available bitstream format types

Function Documentation

void RDOPTS_FreeOptions (const RDOPTS_sOptionDefinition * *pOptionList*)

Free option storage memory allocated during RDOPTS_ReadOptions.

Parameters:

pOptionList - pointer to the array of option definitions.

vc1_eResult RDOPTS_ReadOptions (const char * *ToolName*, const char * *OptionFileName*, const RDOPTS_sOptionDefinition * *pOptionList*)

Parse options from a parameter file as specified by an option list.

Parameters:

ToolName - pointer to the name to accept in IF processing
OptionFileName - pointer to the parameter file to open and parse
pOptionList - pointer to the array of option definitions.

Outputs: Variables referenced by the option list updated as required.

vc1_eResult RDOPTS_ValidateFrameRate (char * *pValue*, UWORD32 * *pNumerator*, UHWD16 * *pDenominator*)

Verify value has a valid framerate description.

Parameters:

pValue - pointer to the value string
pNumerator - pointer to numerator variable to set
pDenominator - pointer to demoninator variable to set

Outputs:

- **pNumerator* - updated if valid value
- **pDenominator* - updated if valid value

Returns:

- `vc1_ResultOK` on success

9.4 vc1cropmv.h File Reference

Functions

- void **vc1CROPMV_ChromaPullBack** (**vc1_sPosition** *pPosition, **vc1_sMV** *pMV)

Detailed Description

This file defines the functions used to crop motion vectors to the reference picture limits.

Function Documentation

void vc1CROPMV_ChromaPullBack (vc1_sPosition * *pPosition*, vc1_sMV * *pMV*)

Crop the integer part of a chroma motion vector.

Remarks:

Crops integer part to the range [-8,WidthMB*8] x [-8, HeightMB*8]. The fractional part is not altered. This operation is performed before interpolation of chroma motion vectors, or Direct storage and also performed on Chroma motion vectors before they are used for deblocking comparison.

Remarks:

Simple and main profile only.

Parameters:

pPosition - pointer to current position
pMV - pointer to motion vector to pull back

9.5 vc1deblock.h File Reference

Functions

- void **vc1DEBLOCK_DeblockSlice** (**vc1_sPosition** *pPosition)

Detailed Description

This file defines the functions used to deblock images.

Function Documentation

void vc1DEBLOCK_DeblockSlice (vc1_sPosition * *pPosition*)

Deblock a picture slice.

Parameters:

pPosition - Position of the END of the slice

Outputs: Picture slice deblocked

9.6 vc1debug.h File Reference

Detailed Description

This file defines the macros and functions used in debug builds.

9.7 vc1dec.h File Reference

Data Structures

- struct **vc1DEC_sBitplane**
- struct **vc1DEC_sBitstream**
- struct **vc1DEC_sDecoderConfiguration**
- struct **vc1DEC_sPictureLayerParams**
- struct **vc1DEC_sState**
- struct **vc1DEC_sVLCCode**

Functions

- **vc1_eResult vc1DEC_DecoderRequirements** (**UWORD32** *pSize, **vc1DEC_sDecoderConfiguration** *pConfig, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DEC_DecoderInitialise** (**vc1DEC_sState** *pState, **vc1DEC_sDecoderConfiguration** *pConfig)
- **vc1_eResult vc1DEC_DecodeSequence** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DEC_DecodeFrame** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream, **vc1_sPicture** *pPicture)
- **vc1_eResult vc1DEC_DecodeField** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream, **vc1_sPicture** *pPicture)
- **vc1_eResult vc1DEC_DecodeSlice** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream, **vc1_sPicture** *pPicture)
- **vc1_eResult vc1DEC_DecodeEntryPoint** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DEC_DecodeFlush** (**vc1DEC_sState** *pState, **vc1_sPicture** *pPicture)
- **vc1_eResult vc1DEC_UpdateBuffers** (**vc1DEC_sState** *pState, **UWORD32** Bits)

Detailed Description

This file defines the top-level decoder library API and structures.

Function Documentation

vc1_eResult vc1DEC_DecodeEntryPoint (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Decode the entry point layer of the bitstream.

Remarks:

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure

Outputs:

- *pState* - updated with new entry point layer information
- *pBitstream* - updated with new bitstream position

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecodeField (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*, vc1_sPicture * *pPicture*)

Decode a field.

Remarks:

Must be called after a call to **vc1DEC_DecodeFrame()** (*p.82*) has been made, otherwise most of the state will be invalid.

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure
pPicture - pointer to an initialised picture structure

Outputs:

- *pState* - updated with data read from the bitstream
- *pBitstream* - updated with new bitstream position
- *pPicture* - updated with decoded picture information

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecodeFlush (vc1DEC_sState * *pState*, vc1_sPicture * *pPicture*)

Return the final picture in a stream.

Remarks:

The decoder must buffer reference pictures in order to return B frames in the correct sequence, so this function is called to flush that buffer to the application.

Parameters:

pState - pointer to the decoder state structure

pPicture - pointer to an initialised picture structure, into which the new picture will be written

Outputs:

- *pPicture* - updated with decoded picture information

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecodeFrame (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*, vc1_sPicture * *pPicture*)

Decode a frame

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure
pPicture - pointer to an initialised picture structure

Outputs:

- *pState* - updated with data read from the bitstream
- *pBitstream* - updated with new bitstream position
- *pPicture* - updated with decoded picture information

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecoderInitialise (vc1DEC_sState * *pState*, vc1DEC_sDecoderConfiguration * *pConfig*)

Initialise the state structure of the decoder.

Remarks:

Initialises the state structure, sets up picture structures, etc.

Parameters:

pState - pointer to the decoder state structure which will be initialised
pConfig - pointer to a configuration structure, used to determine memory requirement

Outputs:

- *pState* - initialised for use by the decoder

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecoderRequirements (UWORD32 * *pSize*, vc1DEC_sDecoderConfiguration * *pConfig*, vc1DEC_sBitstream * *pBitstream*)

Obtain memory requirements for decoding a bitstream.

Remarks:

Examines the profile (and if present, the level) to determine the memory requirement.

Parameters:

pSize - pointer to UWORD32 into which the memory requirement will be written
pConfig - pointer to a configuration structure, used to determine memory requirement

pBitstream - pointer to the start of the bitstream

Outputs:

- *pSize* - updated with memory requirement
- *pConfig* - updated with profile and level from bitstream.

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecodeSequence (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Decode the sequence layer of the bitstream.

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure

Outputs:

- *pState* - updated with new sequence layer information
- *pBitstream* - updated with new bitstream position

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_DecodeSlice (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*, vc1_sPicture * *pPicture*)

Decode a slice.

Remarks:

Must be called after a call to **vc1DEC_DecodeFrame()** (*p.82*) or **vc1DEC_DecodeField()** (*p.81*) has been made, otherwise most of the state will be invalid.

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure
pPicture - pointer to an initialised picture structure

Outputs:

- *pState* - updated with data read from the bitstream
- *pBitstream* - updated with new bitstream position
- *pPicture* - updated with decoded picture information

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DEC_UpdateBuffers (vc1DEC_sState * *pState*, UWORD32 *Bits*)

Update the internal bitstream buffers

Remarks:

Updates the hypothetical reference decoder buffers, if used by the bitstream, and will warn on underflow or overflow. If HRD isn't used the function does nothing, and returns vc1_ResultOK.

Parameters:

pState - pointer to the decoder state structure
Bits - number of bits used

Outputs:

- *pState* - HRD model is updated

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.8 vc1dec3dh.h File Reference

Data Structures

- struct **vc1DEC3DH_sRunLevel**

Functions

- **vc1_eResult vc1DEC3DH_DecodeACRunLevel** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream, **HWD16** *pCoefs, **vc1_sPosition** *pPosition, int eBlk)

Detailed Description

This file provides functions and structures for handling variable length coding tables.

Function Documentation

vc1_eResult vc1DEC3DH_DecodeACRunLevel (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*, HWD16 * *pCoefs*, vc1_sPosition * *pPosition*, int *eBlk*)

Decode AC run/level information.

Remarks:

Calls UnpackTransformACCoef() for each coefficient, and stores the result in the pCoefs array. This code is based on the pseudo-code in figure titled "Coefficient decode pseudo-code" of the standard. Possible improvements include combining this function with the de-zigzag operation.

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure
pCoefs - pointer to a 64 element array, into which coefficients will be written
pPosition - pointer to position structure indicating current MB
eBlk - the block number to which these AC coefficients relate

Outputs:

- *pState* - updated with data read from the bitstream
- *pBitstream* - updated with new bitstream position
- *pCoefs* - updated with coefficients unpacked from the bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.9 vc1decbit.h File Reference

Typedefs

- typedef const **vc1DEC_sVLCCode** * **vc1DECBIT_pVLCZone**

Functions

- **WORD32 vc1DECBIT_GetBits** (**vc1DEC_sBitstream** *pBitstream, int BitCount)
- **WORD32 vc1DECBIT_LookBits** (**vc1DEC_sBitstream** *pBitstream, int BitCount)
- **WORD32 vc1DECBIT_GetVLC** (**vc1DEC_sBitstream** *pBitstream, const **vc1DEC_sVLCCode** *pTable)
- **vc1_eResult vc1DECBIT_InitialiseBitstream** (**vc1DEC_sBitstream** *pBitstream, **UBYTE8** *pBuffer, int Length, int Encapsulated)
- **UBYTE8** * **vc1DECBIT_BufferGet** (**vc1DEC_sBitstream** *pBitstream)
- **WORD32 vc1DECBIT_BufferSet** (**vc1DEC_sBitstream** *pBitstream, **UBYTE8** *pBuffer, int Length)
- **UWORD32 vc1DECBIT_ByteCountGet** (**vc1DEC_sBitstream** *pBitstream)
- **WORD32 vc1DECBIT_ByteCountSet** (**vc1DEC_sBitstream** *pBitstream, **UWORD32** Length)
- **UWORD32 vc1DECBIT_BitCountGet** (**vc1DEC_sBitstream** *pBitstream)
- **WORD32 vc1DECBIT_AlignBit** (**vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DECBIT_TrimBuffer** (**vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that read fixed-length and variable-length codes, with error handling.

Typedef Documentation

typedef const vc1DEC_sVLCCode* vc1DECBIT_pVLCZone

Type for holding a variable length coding zone.

Remarks:

This is just a pointer to an array of vc1DECBIT_sVLCCode values, but the typedef is needed for the automatic table generator.

Function Documentation

WORD32 vc1DECBIT_AlignBit (vc1DEC_sBitstream * *pBitstream*)

Align bitstream input to a byte boundary.

Remarks:

Aligns bitstream pointer to the next byte boundary.

Parameters:

pBitstream - pointer to structure representing position in bitstream

Outputs:

- *pBitstream* - bitstream information updated

Returns:

- VC1DECBIT_EOF if the byte aligned to is past the end of the buffer
- 0 otherwise

UWORD32 vc1DECBIT_BitCountGet (vc1DEC_sBitstream * *pBitstream*)

Returns the number of bits remaining in the bitstream buffer.

Parameters:

pBitstream - pointer to structure representing position in bitstream

Returns:

The number of bits remaining in the bitstream buffer.

UBYTE8* vc1DECBIT_BufferGet (vc1DEC_sBitstream * *pBitstream*)

Returns the current buffer pointer for the bitstream.

Parameters:

pBitstream - pointer to structure representing position in bitstream

Returns:

The current bitstream buffer pointer.

WORD32 vc1DECBIT_BufferSet (vc1DEC_sBitstream * *pBitstream*, UBYTE8 * *pBuffer*, int *Length*)

Sets the current buffer pointer and length.

Parameters:

pBitstream - pointer to structure representing position in bitstream
pBuffer - new buffer pointer
Length - new buffer

Outputs:

- *pBitstream* - bitstream information updated

Returns:

- VC1DECBIT_EOF if illegal encapsulation encountered

UWORD32 vc1DECBIT_ByteCountGet (vc1DEC_sBitstream * *pBitstream*)

Returns the number of whole or partial bytes remaining unread in the bitstream buffer. This does not include bytes read into the internal bitstream state. Use vc1DECBIT_BitCountGet for information on the amount of readable data left.

Parameters:

pBitstream - pointer to structure representing position in bitstream

Returns:

The number of whole or partial bytes remaining in the bitstream buffer.

WORD32 vc1DECBIT_ByteCountSet (vc1DEC_sBitstream * *pBitstream*, UWORD32 *Length*)

Sets the current number of whole bytes in the buffer

Parameters:

pBitstream - pointer to structure representing position in bitstream
Length - new length

Outputs:

- *pBitstream* - bitstream information updated

Returns:

- VC1DECBIT_EOF if illegal encapsulation encountered

WORD32 vc1DECBIT_GetBits (vc1DEC_sBitstream * *pBitstream*, int *BitCount*)

Get bits from a memory location.

Remarks:

Reads BitCount bits from pBuffer, returning them in the least significant bits of the result. This function can read up to 31 bits per call.

Parameters:

pBitstream - pointer to structure representing position in bitstream
BitCount - number of bits to read from memory. Max 31.

Outputs:

- *pBitstream* - structure updated to new position in bitstream

Returns:

- VC1DECBIT_EOF if there are not enough bytes in the buffer
- the value read from the bitstream otherwise

WORD32 vc1DECBIT_GetVLC (vc1DEC_sBitstream * *pBitstream*, const vc1DEC_sVLCCode * *pTable*)

Read VLC bits from a memory location.

Remarks:

Reads variable length code bits from pBuffer, according to table pTable, returning them in the least significant bits of the result. This function can read up to 31 bits per call.

Parameters:

pBitstream - pointer to structure representing position in bitstream
pTable - VLC table

Outputs:

- *pBitstream* - structure updated to new position in bitstream

Returns:

- VC1DECBIT_EOF if not enough bytes in the buffer, or code unmatched
- value read from the bitstream otherwise

vc1_eResult vc1DECBIT_InitialiseBitstream (vc1DEC_sBitstream * *pBitstream*, UBYTE8 * *pBuffer*, int *Length*, int *Encapsulated*)

Initialises the bitstream structure.

Remarks:

The initial buffer pointer will be set to the value provided.

Parameters:

pBitstream - Pointer to structure representing position in bitstream
pBuffer - Initial buffer pointer
Length - Bitstream length
Encapsulated - TRUE if the bitstream is encapsulated

Outputs:

- *pBitstream* - bitstream information updated

Returns:

Result code.

WORD32 vc1DECBIT_LookBits (vc1DEC_sBitstream * *pBitstream*, int *BitCount*)

Look at bits at a memory location, not updating the bitstream pointer.

Remarks:

Used to examine bits for a VLC code. This function can look at up to 31 bits per call.

Parameters:

pBitstream - pointer to structure representing position in bitstream
BitCount - number of bits to examine. Max 31.

Outputs: None.

Returns:

The value read from the bitstream.

vc1_eResult vc1DECBIT_TrimBuffer (vc1DEC_sBitstream * *pBitstream*)

Set up the bit resolution buffer length in the bitstream structure.

Parameters:

pBitstream - pointer to structure representing position in bitstream

Outputs:

- *pBitstream* - bitstream information updated

Returns:

- `vc1_ResultFatal` - if last byte is entirely 0
- `vc1_ResultOK` - otherwise

9.10 vc1decbitpl.h File Reference

Functions

- **vc1_eResult vc1DECBITPL_ReadBitplane** (**vc1DEC_sState** *pState, **vc1DEC_sBitplane** *pBitplane, **vc1DEC_sBitstream** *pBitstream)
- **WORD32 vc1DECBITPL_ReadBitplaneBit** (**vc1DEC_sBitplane** *pBitplane, **vc1DEC_sBitstream** *pBitstream)
- void **vc1DECBITPL_SkipBitplaneBit** (**vc1DEC_sBitplane** *pBitplane)

Detailed Description

This file provides functions for reading bitplane information.

Function Documentation

vc1_eResult vc1DECBITPL_ReadBitplane (vc1DEC_sState * *pState*, vc1DEC_sBitplane * *pBitplane*, vc1DEC_sBitstream * *pBitstream*)

Read a bitplane from a bitstream into a bitplane structure.

Remarks:

If the bitplane is in raw mode, fields of the bitplane structure are set to 0, and bits read later from the bitplane actually come directly from the bitstream.

Parameters:

pState - pointer to decoder state
pBitplane - pointer to a bitplane structure, which will store the bitplane information
pBitstream - pointer to a structure representing the position in a bitstream

Outputs:

- *pBitplane* - updated with bitplane information
- *pBitstream* - updated with new position in bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

WORD32 vc1DECBITPL_ReadBitplaneBit (vc1DEC_sBitplane * *pBitplane*, vc1DEC_sBitstream * *pBitstream*)

Read a bit from a bitplane at a specified co-ordinate.

Remarks:

If the bitplane is in raw mode, bits are read directly from the bitstream rather than the bitplane.

Parameters:

pBitplane - pointer to a bitplane structure, containing the bitplane information
pBitstream - pointer to a structure representing the position in a bitstream

Outputs:

- *pBitstream* - updated with new position in bitstream

Returns:

- VC1DECBIT_EOF if buffer exhausted
- bit value otherwise

void vc1DECBITPL_SkipBitplaneBit (vc1DEC_sBitplane * *pBitplane*)

Skip the next bit in a bitplane.

Remarks:

If the bitplane is in raw mode, nothing happens. If in non-raw mode, the bitplane pointer is updated.

Parameters:

pBitplane - pointer to a bitplane structure, containing the bitplane information

Outputs:

- *pBitplane* - updated with new position in bitplane

9.11 vc1decblk.h File Reference

Functions

- **vc1_eResult vc1DECBLK_DecodeBlockLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that extract and store data from the block layer, and reconstruct the blocks that data represents.

Function Documentation

vc1_eResult vc1DECBLK_DecodeBlockLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Decode the block layer of the bitstream.

Remarks:

Unpacks and decodes the block layer. Operations depending on the bitstream, and that can fail due to this, are in UnpackBlock(). Operations purely dependent on data in state or other structures are in ReconstructBlock().

Parameters:

pState - pointer to the decoder state structure
pBitstream - pointer to the bitstream structure

Outputs:

- *pState* - updated with data read from the bitstream
- *pBitstream* - updated with new bitstream position

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.12 vc1decent.h File Reference

Functions

- **vc1_eResult vc1DECENT_UnpackEntryPointLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that unpack and store data from the entry point layer of a stream.

Function Documentation

vc1_eResult vc1DECENT_UnpackEntryPointLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack the entry point layer of the bitstream, and put the values into the state structure.

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pState* - updated with data from bitstream
- *pBitstream* - updated with new position in bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.13 vc1decmb.h File Reference

Data Structures

- struct **vc1DECMB_sMBMode**

Functions

- **vc1_eResult vc1DECMB_UnpackMacroblockLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that unpack and store data from the macroblock layer of a stream.

Function Documentation

vc1_eResult vc1DECMB_UnpackMacroblockLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack the macroblock layer of the bitstream, and put the values into the state structure.

Remarks:

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pState* - updated with data from bitstream
- *pBitstream* - updated with new position in bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.14 vc1decmv.h File Reference

Functions

- void **vc1DECMV_ApplyMVPrediction** (**vc1DEC_sState** *pState, int eBlk, **vc1_sMV** *pPred, int Backwards)
- **vc1_eResult vc1DECMV_UnpackMVData** (**vc1_sMB** *pMB, **vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream, int eBlk, int Backwards)
- **vc1_eResult vc1DECMV_UnpackMVDataInterlace** (**vc1_sMB** *pMB, **FLAG** *pPredFlag, **vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream, int eBlk, **FLAG** TwoField, int Backwards)

Detailed Description

This file provides motion vector prediction functions specific to the decoder.

Function Documentation

void vc1DECMV_ApplyMVPrediction (vc1DEC_sState * *pState*, int *eBlk*, vc1_sMV * *pPred*, int *Backwards*)

Apply predicted motion vectors to the decoded differential motion vectors.

Remarks:

None.

Parameters:

pState - pointer to the decoder's state
eBlk - block number of current block
pPred - pointer to the predicted motion vectors
Backwards - 0=forward mv, 1=backward mv

Outputs:

- *pState* - updated with new motion vectors

vc1_eResult vc1DECMV_UnpackMVData (vc1_sMB * *pMB*, vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*, int *eBlk*, int *Backwards*)

Unpack differential motion vectors for progressive bitstreams.

Remarks:

None.

Parameters:

pMB - pointer to current macroblock

pState - pointer to the decoder's state
pBitstream - pointer to struct representing the current bitstream
eBlk - block number of current block
Backwards - 0=forward mv, 1=backward mv

Outputs:

- *pMB* - updated with new differential motion vectors
- *pBitstream* - updated with new position

Returns:

Standard vc1_eResult result. See the enumeration for possible result codes.

vc1_eResult vc1DECMV_UnpackMVDataInterlace (vc1_sMB * *pMB*, FLAG * *pPredFlag*, vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*, int *eBlk*, FLAG *TwoField*, int *Backwards*)

Unpack differential motion vectors for interlaced bitstreams.

Remarks:

None.

Parameters:

pMB - pointer to current macroblock
pPredFlag - pointer to location into which can be written the prediction flag
pState - pointer to the decoder's state
pBitstream - pointer to struct representing the current bitstream
eBlk - block number of current block
TwoField - if true, there are two reference field, else one
Backwards - 0=forward mv, 1=backward mv

Outputs:

- *pMB* - updated with new differential motion vectors
- *pPredFlag* - updated with predicted flag status
- *pBitstream* - updated with new position

Returns:

Standard vc1_eResult result. See the enumeration for possible result codes.

9.15 vc1decpic.h File Reference

Functions

- **vc1_eResult vc1DECPIC_UnpackPictureLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DECPIC_UnpackFieldPictureLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DECPIC_UnpackInSlicePictureLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DECPIC_UnpackSyncmarker** (**vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that unpack and store data from the picture layer of a stream.

Function Documentation

vc1_eResult vc1DECPIC_UnpackFieldPictureLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack the field picture and lower layers of the bitstream.

Remarks:

None

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pBitstream* - updated with new position in bitstream
- *pState* - filled out with information from bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DECPIC_UnpackInSlicePictureLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack a picture layer present in a slice

Remarks:

None

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pBitstream* - updated with new position in bitstream
- *pState* - filled out with information from bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes

vc1_eResult vc1DECPIC_UnpackPictureLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack the picture and lower layers of the bitstream.

Remarks:

This function decides what profile and frame type is being decoded, and selects sub functions accordingly.

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pBitstream* - updated with new position in bitstream
- *pState* - filled out with information from bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DECPIC_UnpackSyncmarker (vc1DEC_sBitstream * *pBitstream*)

Unpack the syncmarker.

Remarks:

Discards any payload, although it is available via debug output.

Parameters:

pBitstream - pointer to struct representing the current bitstream

Outputs:

- *pBitstream* - updated with new position

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.16 vc1decseq.h File Reference

Functions

- **vc1_eResult vc1DECSEQ_UnpackSequenceLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that unpack and store data from the sequence layer of a stream.

Function Documentation

vc1_eResult vc1DECSEQ_UnpackSequenceLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack the sequence layer of the bitstream, and put the values into the state structure.

Remarks:

No remarks.

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pBitstream* - updated with new position in bitstream
- *pState* - filled out for information from bitstream

Returns:

Standard vc1_eResult result. See the enumeration for possible result codes.

9.17 vc1decslice.h File Reference

Functions

- **vc1_eResult vc1DECSLICE_DecodeSlice** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)
- **vc1_eResult vc1DECSLICE_UnpackSliceLayer** (**vc1DEC_sState** *pState, **vc1DEC_sBitstream** *pBitstream)

Detailed Description

This file provides functions that unpack and store data from the slice layer of a stream.

Function Documentation

vc1_eResult vc1DECSLICE_DecodeSlice (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Decode a slice of picture data.

Remarks:

There may be no slices, in which case this will decode the entire image.

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pState* - updated with data from bitstream
- *pBitstream* - updated with new position in bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

vc1_eResult vc1DECSLICE_UnpackSliceLayer (vc1DEC_sState * *pState*, vc1DEC_sBitstream * *pBitstream*)

Unpack the slice layer of the bitstream, and put the values into the state structure.

Parameters:

pState - decoder state structure, into which data will be put
pBitstream - structure representing position in bitstream

Outputs:

- *pState* - updated with data from bitstream
- *pBitstream* - updated with new position in bitstream

Returns:

Standard vc1_eResult result. See enumeration for possible result codes.

9.18 vc1deczz.h File Reference

Functions

- void **vc1DECZZ_DeZigZagBlock** (**HWD16** *pOut, **HWD16** *pIn, **UBYTE8** ZZTable, **vc1_eBlkType** eBlkType)

Detailed Description

This file provides functions that perform de-zigzagging operations on transform coefficient arrays.

Function Documentation

void vc1DECZZ_DeZigZagBlock (HWD16 * *pOut*, HWD16 * *pIn*, UBYTE8 *ZZTable*, vc1_eBlkType *eBlkType*)

Apply de-zigzag to input to produce block, depending on block type.

Remarks:

No remarks

Parameters:

pOut - array in which to store de-zigzagged data
pIn - input zigzagged data
ZZTable - zigzag table number
eBlkType - the block type

Outputs:

- *pOut* - newly de-zigzagged data

9.19 vc1derivemv.h File Reference

Functions

- void **vc1DERIVEMV_DeriveMV** (**vc1_sPosition** *pPosition)
- void **vc1DERIVEMV_DirectMV** (**vc1_sPosition** *pPosition)
- void **vc1DERIVEMV_DeriveChromaMV** (**vc1_sMV** *pMV, **vc1_sPosition** *pPosition, int eBlk, int Backwards)
- void **vc1DERIVEMV_DecideChromaBlockType** (**vc1_sPosition** *pPosition)
- void **vc1DERIVEMV_StoreMotionVectors** (**vc1_sPosition** *pPosition)

Detailed Description

This file defines the functions used to derive chroma motion vectors and direct motion vectors.

Function Documentation

void vc1DERIVEMV_DecideChromaBlockType (vc1_sPosition * *pPosition*)

Decide if a chroma block is inter or intra, depending on the luma blocks in the same macroblock.

Parameters:

pPosition - pointer to macroblock position structure

Outputs: Sets Cb and Cr block types for current macroblock

void vc1DERIVEMV_DeriveChromaMV (vc1_sMV * *pMV*, vc1_sPosition * *pPosition*, int *eBlk*, int *Backwards*)

Derive a chroma motion vector.

Remarks:

eBlk is only used for interlaced frame chroma motion vectors.

Parameters:

pMV - pointer to a struct which will contain the new MV
pPosition - pointer to current position
eBlk - luma block number from which chroma mv is derived
Backwards - 0=forwards 1=backwards

Outputs:

- *pMV* - updated with new motion vector

void vc1DERIVEMV_DeriveMV (vc1_sPosition * *pPosition*)

Derive chroma and direct motion vectors.

Remarks:

Reconstructs forward and/or backward motion vectors depending on the MBType. This function must be called exactly once for each macroblock as it also manages the motion vector history for the reference pictures.

Parameters:

pPosition - pointer to current position structure

Outputs:

- *pPosition* - current macroblock updated with derived motion vectors

void vc1DERIVEMV_DirectMV (vc1_sPosition * *pPosition*)

Calculate and apply the direct motion vector for a B macroblock.

Remarks:

This function is only called for B pictures.

Parameters:

pPosition - pointer to current position structure

Outputs:

- *pPosition* is updated as follows:
  - If current MB is forward, fills in backward MV
  - If current MB is backward, fills in forward MV
  - If current MB is direct, fills in both MV

void vc1DERIVEMV_StoreMotionVectors (vc1_sPosition * *pPosition*)

Store the luma motion vectors to the history buffer.

Remarks:

Stores the block type (intra/inter) with a defined vc1_MVIntra value.

Parameters:

pPosition - pointer to macroblock position structure

Outputs:

- *pPosition* - motion vector history buffer updated

9.20 vc1gentab.h File Reference

Data Structures

- struct **vc1_sAspectRatio**
- struct **vc1GENTAB_sMVRangeParams**

Functions

- int **vc1GENTAB_ChooseZigZagTableSet** (**vc1_sPosition** *pPosition)

Detailed Description

This file contains generic tables.

Function Documentation

int vc1GENTAB_ChooseZigZagTableSet (vc1_sPosition * *pPosition*)

Choose a zigzag table set, given the profile, picture type and format.

Remarks:

Selects the table set from vc1GENTAB_pZigZagTables.

Parameters:

pPosition - pointer to struct providing picture type information

Returns:

Zigzag table index.

9.21 vc1hrd.h File Reference

Functions

- void **vc1HRD_Init** (**vc1_sHrdState** *pHrdState, const **vc1_sHrdState** *pHrdInitialState)
- **vc1_eResult vc1HRD_RemoveBits** (**vc1_sHrdState** *pHrdState, HRDVALUE Bits)
- void **vc1HRD_AddBits** (**vc1_sHrdState** *pHrdState, unsigned SecNum, unsigned SecDen)
- HRDVALUE **vc1HRD_MinFullness** (**vc1_sHrdState** *pHrdState)

Detailed Description

This file defines the functions used to implement the Hypothetical Reference Decoder model.

Function Documentation

void vc1HRD_AddBits (vc1_sHrdState * *pHrdState*, unsigned *SecNum*, unsigned *SecDen*)

Add the number of bits that have arrived in a given time.

Parameters:

pHrdState - Pointer to Hypothetical reference decoder state
SecNum - Seconds Numerator (frame rate denominator)
SecDen - Seconds Denominator (frame rate numerator)

Outputs:

- *pHrdState* - Updated for the extra bits arriving in this time

void vc1HRD_Init (vc1_sHrdState * *pHrdState*, const vc1_sHrdState * *pHrdInitialState*)

Initialize the HRD state.

Parameters:

pHrdState - Pointer to Hypothetical reference decoder state
pHrdInitialState - Pointer to HRD information in sequence header

HRDVALUE vc1HRD_MinFullness (vc1_sHrdState * *pHrdState*)

Calculate the minimum buffer fullness.

Remarks:

If the next frame occupies more bits than the minimum buffer fullness then the decoder will suffer a buffer underflow error.

Parameters:

pHrdState - Pointer to Hypothetical reference decoder state

Returns:

Minimum buffer fullness

vc1_eResult vc1HRD_RemoveBits (vc1_sHrdState * *pHrdState*, HRDVALUE *Bits*)

Remove bits from the hypothetical reference coder buffers.

Remarks:

When decoding a frame, call this function to remove the number of bits in the frame from the HRD model.

Parameters:

pHrdState - Pointer to Hypothetical reference decoder state
Bits - Number of bits to remove

Returns:

- `vc1_ResultOK` - if the call succeeded
- `vc1_ResultHrdUnderflow` - if there were not enough bits to remove

9.22 vc1interp.h File Reference

Data Structures

- struct **vc1INTERP_sBicubicFilterParams**

Functions

- void **vc1INTERP_InterpPatchQuarterPelBicubic** (**vc1_sComponent** *pC, **vc1_sInterpolate** *pInterp, **WORD32** X, **WORD32** Y)
- void **vc1INTERP_InterpPatchQuarterPelBilinear** (**vc1_sComponent** *pC, **vc1_sInterpolate** *pInterp, **WORD32** X, **WORD32** Y)
- void **vc1INTERP_InterpPatchHalfPelBilinear** (**vc1_sComponent** *pC, **vc1_sInterpolate** *pInterp, **WORD32** X, **WORD32** Y)
- void **vc1INTERP_InterpPatchHalfPelBicubic** (**vc1_sComponent** *pC, **vc1_sInterpolate** *pInterp, **WORD32** X, **WORD32** Y)
- void **vc1INTERP_PredictMB** (**UBYTE8** pPredBlk[6][64], **vc1_sPosition** *pPosition, **FLAG** RndCtrl)
- void **vc1INTERP_InterlacePredMB** (**UBYTE8** pDestMB[6][64], const **UBYTE8** pSourceMB[6][64])
- void **vc1INTERP_InterlaceDiffMB** (**HWD16** pDestMB[6][64], const **HWD16** pSourceMB[6][64])

Detailed Description

This file defines the block interpolation functions.

Function Documentation

void vc1INTERP_InterlaceDiffMB (HWD16 *pDestMB*[6][64], const HWD16 *pSourceMB*[6][64])

Interlace a differential macroblock.

Remarks:

Interleaves the top blocks with the bottom blocks, and writes out four interlaced blocks.

Parameters:

pDestMB - pointer to initialised buffer to hold the new data
pSourceMB - pointer to the source macroblock data

Outputs:

- *pDest* - updated with new interlaced data

void vc1INTERP_InterlacePredMB (UBYTE8 *pDestMB*[6][64], const UBYTE8 *pSourceMB*[6][64])

Interlace a predicted macroblock.

Remarks:

Interleaves the top blocks with the bottom blocks, and writes out four interlaced blocks.

Parameters:

pDestMB - pointer to initialised buffer to hold the new data
pSourceMB - pointer to the source macroblock data

Outputs:

- *pDest* - updated with new interlaced data

void vc1INTERP_InterpPatchHalfPelBicubic (vc1_sComponent * *pC*, vc1_sInterpolate * *pInterp*, WORD32 *X*, WORD32 *Y*)

Bicubic interpolation of a patch to half pixel position resolution.

Remarks:

This function is implemented using the quarter pixel resolution routines.

Parameters:

pC - pointer to a component into which the filtered patch will be written
pInterp - pointer to structure containing parameters required for operation
X - X position in half pixel units
Y - Y position in half pixel units

Outputs:

- *pC* - updated with the filtered patch

void vc1INTERP_InterpPatchHalfPelBilinear (vc1_sComponent * *pC*, vc1_sInterpolate * *pInterp*, WORD32 *X*, WORD32 *Y*)

Bilinear interpolation of a patch to half pixel position resolution.

Remarks:

This function is implemented using the quarter pixel resolution routines.

Parameters:

pC - pointer to a component into which the filtered patch will be written
pInterp - pointer to structure containing parameters required for operation
X - X position in half pixel units
Y - Y position in half pixel units

Outputs:

- *pC* - updated with the filtered patch

void vc1INTERP_InterpPatchQuarterPelBicubic (vc1_sComponent * *pC*, vc1_sInterpolate * *pInterp*, WORD32 *X*, WORD32 *Y*)

Bicubic interpolation of a patch to quarter pixel position resolution.

Parameters:

pC - pointer to a component into which the filtered patch will be written
pInterp - pointer to structure containing parameters required for operation
X - X position in quarter pixel units
Y - Y position in quarter pixel units

Outputs:

- *pC* - updated with the filtered patch

void vc1INTERP_InterpPatchQuarterPelBilinear (vc1_sComponent * *pC*, vc1_sInterpolate * *pInterp*, WORD32 *X*, WORD32 *Y*)

Bilinear interpolation of a patch to quarter pixel position resolution

Parameters:

pC - pointer to a component into which the filtered patch will be written
pInterp - pointer to structure containing parameters required for operation
X - X position in half pixel units
Y - Y position in half pixel units

Outputs:

- *pC* - updated with the filtered patch

void vc1INTERP_PredictMB (UBYTE8 *pPredBlk*[6][64], vc1_sPosition * *pPosition*, FLAG *RndCtrl*)

Predict macroblock from a reference picture.

Remarks:

Uses the macroblock type and motion vectors to form a prediction.

Parameters:

pPredBlk - pointer to an array into which the filtered blocks will be written
pPosition - pointer to a position structure indicating the current macroblock
RndCtrl - rounding control flag from picture layer

Outputs:

- *pPredBlk* - updated with the filtered blocks

9.23 vc1iquant.h File Reference

Functions

- void **vc1IQUANT_ChooseQuantizer** (**vc1_sQuant** *pQuant, **vc1_sPosition** *pPos, **vc1_eQuantMode** eQuantMode, int AltQuant)
- void **vc1IQUANT_GetQuantizer** (**vc1_sQuant** *pQuant, int PQINDEX, **vc1_eQuantizer** eQuantizer)
- **HWD16 vc1IQUANT_InverseDCQuantize** (**WORD32** DCCoeffQ, **vc1_sQuant** *pQuant)
- void **vc1IQUANT_InverseACQuantize** (**HWD16** *pOut, const **HWD16** *TCoefs, **vc1_sQuant** *pQuant, int Intra)

Detailed Description

This file defines the inverse quantization functions.

Function Documentation

void vc1IQUANT_ChooseQuantizer (vc1_sQuant * *pQuant*, vc1_sPosition * *pPos*, vc1_eQuantMode *eQuantMode*, int *AltQuant*)

Calculates quantizer according to quantization mode.

Remarks:

This does not handle the quantization modes:

- vc1_QuantModeMBDual
- vc1_QuantModeMBAny

Parameters:

pQuant - Pointer to picture quantizer (PQUANT, HALFQP, NONUNIFORM)
pPos - Pointer to position and current macroblock
eQuantMode - Quantization mode
AltQuant - ALTPQUANT value

Outputs:

- *pQuant* - Updated to macroblock quantization value

void vc1IQUANT_GetQuantizer (vc1_sQuant * *pQuant*, int *PQINDEX*, vc1_eQuantizer *eQuantizer*)

Converts PQINDEX to a PQUANT and NonUniform according to current Quantizer mode.

Parameters:

pQuant - Pointer to **vc1_sQuant** (*p.47*) structure to fill
PQINDEX - PQINDEX in the range 1 to 31
eQuantizer - Quantizer mode, one of:

- vc1_QuantizerImplicit
- vc1_QuantizerUniform
- vc1_QuantizerNonUniform

Outputs:

- *pQuant* - Updated with new quantisation level and quantiser

void vc1IQUANT_InverseACQuantize (HWD16 * *pOut*, const HWD16 * *TCoefs*, vc1_sQuant * *pQuant*, int *Intra*)

Inverse quantise a group of AC coefficients, given a quantiser, and quantisation level

Remarks:

Parameters:

TCoefs - pointer to a 64-element array of transform coefficients
pQuant - pointer to the **vc1_sQuant** (*p.47*) structure for this block
Intra - 0=Inter 1=Intra

Outputs:

- *pOut* - pointer to a 64-element array of transform coefficients, with dequantised AC coefficients

Returns:

None

HWD16 vc1IQUANT_InverseDCQuantize (WORD32 *DCCoeffQ*, vc1_sQuant * *pQuant*)

Inverse quantise a DC coefficient, given a quantiser and quantisation level.

Remarks:

This function uses **vc1PREDDCAC_DCStepSize()** (*p.120*) from the DC/AC prediction module.

Parameters:

DCCoeffQ - quantised DC coefficient
pQuant - pointer to **vc1_sQuant** (*p.47*) structure

Returns:

Dequantised DC coefficient

9.24 vc1itrans.h File Reference

Functions

- void **vc1ITRANS_InverseTransformBlock** (**HWD16** *pOut, **HWD16** *pIn, **vc1_eBlkType** eType)

Detailed Description

This file defines the inverse transform functions.

Function Documentation

void vc1ITRANS_InverseTransformBlock (HWD16 * *pOut*, HWD16 * *pIn*, vc1_eBlkType *eType*)

2D inverse transform an 8x8 block of coefficients.

Parameters:

pIn - pointer to the input 64-element array
eType - tranform type (8x8, 8x4, 4x8, 4x4)
pOut - pointer to the output 64-element array

Outputs:

- *pOut* - array filled

9.25 vc1pred.h File Reference

Functions

- **vc1_sMB** * **vc1PRED_pLeftMB** (**vc1_sPosition** *pPos)
- **vc1_sMB** * **vc1PRED_pTopMB** (**vc1_sPosition** *pPos)
- **vc1_sMB** * **vc1PRED_pTopLeftMB** (**vc1_sPosition** *pPos)
- **vc1_sBlk** * **vc1PRED_pLeftBlk** (**vc1_sPosition** *pPos, int Blk)
- **vc1_sBlk** * **vc1PRED_pTopBlk** (**vc1_sPosition** *pPos, int Blk)
- **vc1_sBlk** * **vc1PRED_pTopLeftBlk** (**vc1_sPosition** *pPos, int Blk)
- **vc1_sBlk** * **vc1PRED_pB4MVBlk** (**vc1_sPosition** *pPos, int Blk)
- **vc1_sBlk** * **vc1PRED_pB1MVBlk** (**vc1_sPosition** *pPos)

Detailed Description

This file contains routines for use by prediction functions.

Function Documentation

vc1_sBlk* vc1PRED_pB1MVBlk (vc1_sPosition * *pPos*)

Find the block for predictor B in 1MV prediction.

Remarks:

This is often the top right block but not always.

Parameters:

pPos - pointer to position of the 1MV macroblock

Returns:

Pointer to the B motion predictor block or NULL if the block is in the first row

vc1_sBlk* vc1PRED_pB4MVBlk (vc1_sPosition * *pPos*, int *Blk*)

Find the block for predictor B in 4MV prediction.

Remarks:

This is often the top right block but not always.

Parameters:

pPos - pointer to position of current macroblock
Blk - Block within the macroblock

Returns:

Pointer to the B motion predictor block or NULL if the block is in the first row

vc1_sBlk* vc1PRED_pLeftBlk (vc1_sPosition * *pPos*, int *Blk*)

Find the block to the left of the current block.

Parameters:

pPos - pointer to position of current macroblock
Blk - Block within the macroblock

Returns:

Pointer to the left block or NULL if the block is in the first column.

vc1_sMB* vc1PRED_pLeftMB (vc1_sPosition * *pPos*)

Find the macroblock to the left of the current macroblock.

Parameters:

pPos - pointer to position of current macroblock

Returns:

Pointer to the left macroblock or NULL if the macroblock is in the first column.

vc1_sBlk* vc1PRED_pTopBlk (vc1_sPosition * *pPos*, int *Blk*)

Find the block to the top of the current block.

Parameters:

pPos - pointer to position of current macroblock
Blk - Block within the macroblock

Returns:

Pointer to the top block or NULL if the block is in the first row.

vc1_sBlk* vc1PRED_pTopLeftBlk (vc1_sPosition * *pPos*, int *Blk*)

Find the block to the top left of the current block.

Parameters:

pPos - pointer to position of current macroblock
Blk - Block within the macroblock

Returns:

Pointer to the top left block or NULL if the block is in the first row or first column.

vc1_sMB* vc1PRED_pTopLeftMB (vc1_sPosition * *pPos*)

Find the macroblock to the top left of the current macroblock.

Parameters:

pPos - pointer to position of current macroblock

Returns:

Pointer to the top left macroblock or NULL if the macroblock is in the first slice row.

vc1_sMB* vc1PRED_pTopMB (vc1_sPosition * *pPos*)

Find the macroblock to the top of the current macroblock.

Parameters:

pPos - pointer to position of current macroblock

Returns:

Pointer to the top macroblock or NULL if the macroblock is in the first slice row.

9.26 vc1predcbp.h File Reference

Functions

- int **vc1PREDCBP_ApplyCBPCYPred** (**vc1_sPosition** *pPos, int CBPCY)
- int **vc1PREDCBP_CBPCYDifferential** (**vc1_sPosition** *pPos, int CBPCY)

Detailed Description

This file defines the functions for predicting CBPY.

Function Documentation

int vc1PREDCBP_ApplyCBPCYPred (vc1_sPosition * *pPos*, int *CBPCY*)

Generate coded block pattern from prediction and differential.

Parameters:

pPos - pointer to position of current macroblock
CBPCY - coded block pattern differential bit5=Y0 bit0=Cr

Returns:

The absolute CBPCY (prediction applied).

int vc1PREDCBP_CBPCYDifferential (vc1_sPosition * *pPos*, int *CBPCY*)

Generates the coded block pattern differential.

Parameters:

pPos - pointer to position of current macroblock
CBPCY - absolute coded block patternl bit5=Y0 bit0=Cr

Returns:

The differential CBPCY (prediction subtracted).

9.27 vc1preddcac.h File Reference

Functions

- int **vc1PREDDCAC_DCDefault** (int MQuant, int Bias)
- int **vc1PREDDCAC_ACPREDPresent** (**vc1_sPosition** *pPos)
- **vc1_eBlkType vc1PREDDCAC_PredictDCAC** (**HWD16** *pPred, **vc1_sPosition** *pPos, int Blk, int DCDefault)
- unsigned int **vc1PREDDCAC_DCStepSize** (unsigned int Quant)
- void **vc1PREDDCAC_CopyDCAC** (**vc1_sBlk** *pBlk, **HWD16** *pData)

Detailed Description

This file defines the AC and DC quantized coefficient prediction functions.

Function Documentation

int vc1PREDDCAC_ACPREDPresent (vc1_sPosition * *pPos*)

Determine whether the ACPRED flag is present for a macroblock.

Remarks:

The ACPRED flag is present if the macroblock contains an intra block with an intra neighbour to the top or left.

Parameters:

pPos - Position of the current macroblock

Returns:

TRUE if ACPRED is present, otherwise FALSE.

void vc1PREDDCAC_CopyDCAC (vc1_sBlk * *pBlk*, HWD16 * *pData*)

Copy the DC, Top and Left AC coefficients from a block to the block state structure.

Parameters:

pData - Pointer to transformed quantised coefficients
pBlk - Pointer to the block to update

Outputs:

- **pBlk* - Block updated

int vc1PREDDCAC_DCDefault (int *MQuant*, int *Bias*)

Calculate the default DC predictor for a block.

Remarks:

In general the default is round((8*Bias)/DCStepSize).

Parameters:

MQuant - Macroblock quantizer
Bias - Bias applied to DC coefficients (0 or 128)

Returns:

The calculated DC predictor.

unsigned int vc1PREDDCAC_DCStepSize (unsigned int *Quant*)

Calculate DCStepSize from the quantization step as described in the standard.

Remarks:

This function is only called for an intra 8x8 block.

Parameters:

Quant - quantisation step

Returns:

The calculated DCStepSize.

vc1_eBlkType vc1PREDDCAC_PredictDCAC (HWD16 * *pPred*, vc1_sPosition * *pPos*, int *Blk*, int *DCDefault*)

Predict the DC and AC values of a block from preceding blocks.

Remarks:

This function is only called for an intra 8x8 block.

Parameters:

pPred - Pointer to buffer for predicted DC and AC values
pPos - Pointer to the macroblock being predicted
Blk - Block number within the macroblock
DCDefault - Default DC predictor

- For Rule A it is 0
- For Rule B it is ((1024 + (DCStepSize>>1)) / DCStepSize)

Outputs:

- *pPred* - Pointer to the predicted output of 1 DC and then 7 AC values

Returns:

- `vc1_BlkIntra` - No AC values predicted
- `vc1_BlkIntraLeft` - Predicted left 7 AC values
- `vc1_BlkIntraTop` - Predicted top 7 AC values

9.28 vc1predmv.h File Reference

Functions

- int **vc1PREDMV_PredictProgressiveMV** (**vc1_sMV** *pPred, **vc1_sPosition** *pPos, int Blk, int eHybridPred, int Back)
- int **vc1PREDMV_PredictInterlacedFieldMV** (**vc1_sMV** *pPred, **vc1_sPosition** *pPos, int Blk, int eHybridPred, int PredFlag, int Back)
- void **vc1PREDMV_PredictInterlacedFrameMV** (**vc1_sMV** *pPred, **vc1_sPosition** *pPos, int Blk, int Back)

Detailed Description

This file defines the motion vector prediction functions.

Function Documentation

int vc1PREDMV_PredictInterlacedFieldMV (vc1_sMV * *pPred*, vc1_sPosition * *pPos*, int *Blk*, int *eHybridPred*, int *PredFlag*, int *Back*)

Predict the X and Y motion vectors for an interlace field block.

Remarks:

For a 1MV macroblock Blk is ignored. For a 4MV macroblock this function can be called to predict any of the four MVs.

Parameters:

pPred - Pointer to the predicted output of X, Y
pPos - Pointer to the macroblock being predicted
Blk - Block number within the macroblock (for 4MV)
eHybridPred - Hybrid prediction direction
PredFlag - 0=use dominant predictor 1=use non-dominant predictor
Back - 0=predict forward motion 1=predict backward motion

Outputs:

- *pPred* - predicted output values written

Returns:

TRUE if Hybrid prediction applied, otherwise FALSE.

void vc1PREDMV_PredictInterlacedFrameMV (vc1_sMV * *pPred*, vc1_sPosition * *pPos*, int *Blk*, int *Back*)

Predict the X and Y motion vectors for an interlace frame block.

Remarks:

- For a 1MV Frame MB Blk must be 0.
- For a 2MV Field MB Blk can be 0 (top field) or 2 (bottom field).
- For a 4MV Frame MB Blk can be 0 to 3 (top left/right, bottom left/right block).
- For a 4MV Field MB Blk can be 0 to 3 (top left/right, bottom left/right field).

Parameters:

pPred - Pointer to the predicted output of X, Y
pPos - Pointer to the macroblock being predicted
Blk - Block number within the macroblock
Back - 0=predict forward motion 1=predict backward motion

Outputs:

- *pPred* - predicted output values written

int vc1PREDMV_PredictProgressiveMV (vc1_sMV * *pPred*, vc1_sPosition * *pPos*, int *Blk*, int *eHybridPred*, int *Back*)

Predict the X and Y motion vectors for a progressive block.

Remarks:

For a 1MV macroblock Blk is ignored. For a 4MV macroblock this function can be called to predict any of the four MVs.

Parameters:

pPred - Pointer to the predicted output of X, Y
pPos - Pointer to the macroblock being predicted
Blk - Block number within the macroblock (for 4MV)
eHybridPred - Hybrid prediction direction
Back - 0=predict forward motion 1=predict backward motion

Outputs:

- *pPred* - predicted output values written

Returns:

TRUE if Hybrid prediction applied, otherwise FALSE.

9.29 vc1recon.h File Reference

Functions

- void **vc1RECON_ReconstructMB** (const **HWD16** pTCoefs[6][64], const **UBYTE8** pPredBlk[6][64], **vc1_sReferencePicture** *pRefPic, **vc1_sPosition** *pPosition, **vc1_eProfile** eProfile)

Detailed Description

This file defines the block reconstruction functions.

Function Documentation

void vc1RECON_ReconstructMB (const HWD16 *pTCoefs*[6][64], const UBYTE8 *pPredBlk*[6][64], vc1_sReferencePicture * *pRefPic*, vc1_sPosition * *pPosition*, vc1_eProfile *eProfile*)

Reconstruct a macroblock from transform coefficients, and write into a picture.

Remarks:

This function includes dequantise, inverse transform, overlap smooth and clamping.

Parameters:

pTCoefs - pointer to 6 8x8 arrays of transform coefficients
pPredBlk - pointer to 6 8x8 predicted blocks
pRefPic - pointer to picture structure into which blocks will be written
pPosition - pointer to position structure representing current MB
eProfile - profile (simple/main/advanced) of stream

Outputs:

- *pRefPic* - updated with newly reconstructed blocks

9.30 vc1scalemv.h File Reference

Functions

- void **vc1SCALEMV_ScaleMV** (int *pX, int *pY, const **vc1_sScaleMV** *pSMV, int Opposite)

Detailed Description

This file defines the functions that scale motion vectors for field interlaced motion vector prediction.

Function Documentation

void vc1SCALEMV_ScaleMV (int * *pX*, int * *pY*, const vc1_sScaleMV * *pSMV*, int *Opposite*)

Scale an interlaced motion vector from same to opposite field.

Parameters:

pX - pointer to X of 1/4 pel motion vector
pY - pointer to Y of 1/4 pel motion vector
pSMV - pointer to motion vector scaling parameter
Opposite - 0 = Scale for same field, 1 = Scale for opposite field

Outputs:

- *pX* - pointer to scaled X
- *pY* - pointer to scaled Y

9.31 vc1smooth.h File Reference

Functions

- void **vc1SMOOTH_OverlapSmoothHorizMB** (**vc1_sReferencePicture** *pRefPic, **vc1_sPosition** *pPosition)
- void **vc1SMOOTH_OverlapSmoothMB** (**vc1_sReferencePicture** *pRefPic, **vc1_sPosition** *pPosition, **HWD16** pData[6][64])

Detailed Description

This file defines the overlap smoothing filter functions.

Function Documentation

void vc1SMOOTH_OverlapSmoothHorizMB (vc1_sReferencePicture * *pRefPic*, vc1_sPosition * *pPosition*)

Horizontal overlap smooth the top edges of blocks in the current macroblock.

Remarks:

This function is only called when OverlapFilter is enabled for the macroblock. This function copies the bottom block rows into the macroblock state, then filters over horizontal edges as required.

Parameters:

pRefPic - Pointer to output reference picture
pPosition - Pointer to position of current macroblock

Outputs:

- *pRefPic* - Updated with smoothed pixels

void vc1SMOOTH_OverlapSmoothMB (vc1_sReferencePicture * *pRefPic*, vc1_sPosition * *pPosition*, HWD16 *pData*[6][64])

Perform overlap smoothing on this and the previous macroblock.

Remarks:

This function must always be called even if overlap smoothing is off for the current macroblock as it may be on for the previous macroblock. If overlap smoothing is on for the current macroblock it smooths left vertical edges. If overlap smoothing is on for the previous macroblock it smooths top horizontal edges.

Parameters:

pRefPic - Pointer to output reference picture

pPosition - Pointer to position of current macroblock
pData - Coefficients for current macroblock

Outputs:

- *pRefPic* - Updated with smoothed pixels

9.32 vc1tools.h File Reference

Functions

- int **vc1TOOLS_Median3** (int a, int b, int c)
- int **vc1TOOLS_Median4** (int a, int b, int c, int d)
- void **vc1TOOLS_RangeReduce** (**UBYTE8** *pData, int Bpl, int Width, int Height, int Scale)
- void **vc1TOOLS_GetPictureDestination** (**vc1_sComponent** *pC, **vc1_sReferencePicture** *pRefPic, **vc1_sPosition** *pPosition, int eBlk)
- int **vc1TOOLS_InitReferencePicture** (**vc1_sReferencePicture** *pRef, int Width, int Height, **vc1_eProfile** eProfile, int Interlaced)
- void **vc1TOOLS_NewReference** (**vc1_sPosition** *pPos)
- void **vc1TOOLS_CopyReference** (**vc1_sReferencePicture** *pOut, **vc1_sReferencePicture** *pIn)
- void **vc1TOOLS_ICPadReferencePicture** (**vc1_sPosition** *pPos, **vc1_sIntensityComp** pIntensityComp[2])

Detailed Description

This file defines utility functions.

Function Documentation

void vc1TOOLS_CopyReference (vc1_sReferencePicture * *pOut*, vc1_sReferencePicture * *pIn*)

Copy image data from one reference picture to another.

Remarks:

This function is called when dealing with a skipped frame.

Parameters:

pOut - Pointer to destination reference picture
pIn - Pointer to source reference picture

void vc1TOOLS_GetPictureDestination (vc1_sComponent * *pC*, vc1_sReferencePicture * *pRefPic*, vc1_sPosition * *pPosition*, int *eBlk*)

Obtain a pointer into a picture at a block location.

Parameters:

pC - pointer to initialised component structure
pRefPic - pointer to a reference picture structure
pPosition - pointer to a position structure indicating the current MB
eBlk - the block number

Outputs:

- *pC* - pointer to filled out component structure indicating block address and bytes per line of component plane

void vc1TOOLS_ICPadReferencePicture (vc1_sPosition * *pPos*, vc1_sIntensityComp *pIntensityComp*[2])

Intensity compensate and pad reference pictures ready for motion compensation.

Parameters:

pPos - pointer to details of current image and reference images
pIntensityComp - pointer to the intensity compensation values to apply (one per field)

int vc1TOOLS_InitReferencePicture (vc1_sReferencePicture * *pRef*, int *Width*, int *Height*, vc1_eProfile *eProfile*, int *Interlaced*)

Initialise a reference picture structure.

Remarks:

This function may be called purely to work out the number of bytes required for the reference picture structure and contents. In this case pRef will be NULL. If pRef is not NULL then the structure is also initialized.

Parameters:

pRef - Pointer to the reference picture structure or NULL
Width - Reference image width
Height - Reference image height (x2 if interlaced)
eProfile - Profile to support
Interlaced - TRUE if interlaced images are supported

Returns:

Number of bytes required by the reference picture.

int vc1TOOLS_Median3 (int *a*, int *b*, int *c*)

Take the median value of three inputs.

Parameters:

a - first value
b - second value
c - third value

Returns:

Median of a, b and c.

int vc1TOOLS_Median4 (int *a*, int *b*, int *c*, int *d*)

Take the median value of four inputs.

Parameters:

a - first value
b - second value
c - third value
d - fourth value

Returns:

Average of the two middle values.

void vc1TOOLS_NewReference (vc1_sPosition * *pPos*)

Switches the old and new reference buffers.

Parameters:

pPos - pointer to position structure with:

- pReferenceOld = reference to discard
- pReferenceNew = reference to make old
- RangeRed = is the picture range reduced

void vc1TOOLS_RangeReduce (UBYTE8 * *pData*, int *Bpl*, int *Width*, int *Height*, int *Scale*)

Range reduce an area.

Parameters:

pData - pointer to top left of area
Bpl - bytes per line
Width - area width
Height - area height
Scale - scale factor times 8

Outputs:

- *pData* - area range reduced

9.33 vc1types.h File Reference

Data Structures

- struct **vc1_sBFraction**
- struct **vc1_sBlk**
- struct **vc1_sBlkInter**
- struct **vc1_sBlkIntra**
- struct **vc1_sComponent**
- struct **vc1_sField**
- struct **vc1_sHrdState**
- struct **vc1_sImagePosition**
- struct **vc1_sIntensityComp**
- struct **vc1_sInterpolate**
- struct **vc1_sLeakyBucket**
- struct **vc1_sLevelLimit**
- struct **vc1_sMB**
- struct **vc1_sMotion**
- struct **vc1_sMotionHist**
- struct **vc1_sMV**
- struct **vc1_sPanScanParams**
- struct **vc1_sPanScanWindow**
- struct **vc1_sPicture**
- struct **vc1_sPosition**
- struct **vc1_sQuant**
- struct **vc1_sRectangle**
- struct **vc1_sReferencePicture**
- struct **vc1_sScaleMV**
- struct **vc1_sSequenceLayer**

Typedefs

- typedef signed char **BYTE8**
- typedef unsigned char **UBYTE8**
- typedef signed short **HWD16**
- typedef unsigned short **UHWD16**
- typedef signed int **WORD32**
- typedef unsigned int **UWORD32**
- typedef unsigned long long **ULLONG64**
- typedef signed long long **LLONG64**
- typedef **UBYTE8 FLAG**
- typedef **HWD16 vc1_tRunLevel**

Detailed Description

This file contains all the generic type definitions, enumerations, and defines.

Typedef Documentation

typedef signed char BYTE8

Signed 8 bit type

typedef UBYTE8 FLAG

Boolean type

typedef signed short HWD16

Signed 16 bit type

typedef signed long long LLONG64

Signed 64 bit type

typedef unsigned char UBYTE8

Unsigned 8 bit type

typedef unsigned short UHWD16

Unsigned 16 bit type

typedef unsigned long long ULLONG64

Unsigned 64 bit type

typedef unsigned int UWORD32

Unsigned 32 bit type

typedef HWD16 vc1_tRunLevel

Packed Run Level information

typedef signed int WORD32

Signed 32 bit type

Enumeration Type Documentation

enum vc1_eACPred

Enumeration values:

vc1_ACPredOff AC prediction off
vc1_ACPredOn AC prediction on
vc1_ACPredAbsent No blocks can be predicted

enum vc1_eBitplaneCodingMode

Bitplane coding mode.

Enumeration values:

vc1_BitplaneCodingModeNorm2 Normal-2 bitplane coding
vc1_BitplaneCodingModeNorm6 Normal-6 bitplane coding
vc1_BitplaneCodingModeRowskip Rowskip bitplane coding
vc1_BitplaneCodingModeColskip Colskip bitplane coding
vc1_BitplaneCodingModeDiff2 Diff-2 bitplane coding
vc1_BitplaneCodingModeDiff6 Diff-6 bitplane coding
vc1_BitplaneCodingModeRaw Raw (No) bitplane coding

enum vc1_eBlkNumber

Block number

Enumeration values:

vc1_BlkY0 Top left Y block in MB
vc1_BlkY1 Top right Y block in MB
vc1_BlkY2 Bottom left Y block in MB
vc1_BlkY3 Bottom right Y block in MB
vc1_BlkCb Cb block in MB
vc1_BlkCr Cr block in MB
VC1_BLOCKS_PER_MB Number of blocks per MB

enum vc1_eBlkType

Block Types

Enumeration values:

vc1_BlkInter8x8 Inter coded block, 8px wide, 8px high subblocks
vc1_BlkInter8x4 Inter coded block, 8px wide, 4px high subblocks
vc1_BlkInter4x8 Inter coded block, 4px wide, 8px high subblocks
vc1_BlkInter4x4 Inter coded block, 4px wide, 4px high subblocks
vc1_BlkInterAny Inter coded block, transform not yet chosen
vc1_BlkIntra Intra coded block, no AC prediction
vc1_BlkIntraTop Intra coded block, AC prediction of TOP values
vc1_BlkIntraLeft Intra coded block, AC prediction of LEFT values

enum vc1_eHybridPred

Enumeration values:

vc1_HybridLeft Predict from Left
vc1_HybridTop Predict from Top
vc1_HybridNone No hybrid prediction specified

enum vc1_eLevel

Profile level enumeration

Enumeration values:

vc1_LevelLow Simple/Main profile low level
vc1_LevelMedium Simple/Main profile medium level
vc1_LevelHigh Simple/Main profile high level
vc1_LevelL0 Advanced profile level 0
vc1_LevelL1 Advanced profile level 1
vc1_LevelL2 Advanced profile level 2
vc1_LevelL3 Advanced profile level 3
vc1_LevelL4 Advanced profile level 4
vc1_LevelUnknown Unknown profile

enum vc1_eMBType

Macroblock type bitmap

Enumeration values:

vc1_MBIntra Intra (no motion vectors)
vc1_MB1MV One motion vector
vc1_MB2MV Two motion vectors
vc1_MB4MV Four motion vectors
vc1_MBMVMask Mask of MV information
vc1_MBDirect Direct macroblock
vc1_MBForward Forward prediction
vc1_MBBackward Backward prediction

vc1_MBInterp Forward and backward
vc1_MBDirMask Mask of direction information
vc1_MBFieldMV MV's apply to Fields
vc1_MBSwitchMV Bottom field different direction to Top
vc1_MBFieldTX Field transform

enum vc1_eMVMode

Motion vector mode enumeration

Enumeration values:

vc1_MVMode1MVHalfPelBilinear 1MV 0.50 pel bilinear
vc1_MVMode1MVHalfPel 1MV 0.50 pel bicubic
vc1_MVMode1MV 1MV 0.25 pel bicubic
vc1_MVModeMixedMV MixedMV 0.25 pel bicubic
vc1_MVModeIntensityCompensation Variable length code escape flag

enum vc1_ePadMode

Enumeration defining padding modes supported.

Enumeration values:

vc1_PadSimple Simple or main profile - pad from macroblock edge
vc1_PadAdvancedProgressive Advanced profile progressive - pad from image edge
vc1_PadAdvancedInterlaced Advanced profile interlaced field padding

enum vc1_ePictureFormat

Format of a Picture.

Enumeration values:

vc1_ProgressiveFrame Picture is a progressive frame
vc1_InterlacedFrame Picture is an interlaced frame
vc1_InterlacedField Picture is two interlaced fields
vc1_PictureFormatNone Picture format not yet set

enum vc1_ePictureRes

Scaling to be applied to decoded picture before display

Enumeration values:

vc1_PictureRes1x1 No scaling
vc1_PictureRes2x1 Scale horizontally
vc1_PictureRes1x2 Scale vertically
vc1_PictureRes2x2 Scale horizontally and vertically

enum vc1_ePictureType

Picture type enumeration

Enumeration values:

vc1_PictureTypeI I Picture / Field - can be used as a reference
vc1_PictureTypeP P Picture / Field - can be used as a reference
vc1_PictureTypeB B Picture / Field
vc1_PictureTypeBI BI Picture / Field
vc1_PictureTypeSkipped Skipped Frame

enum vc1_eProfile

Bitstream profile enumeration

Enumeration values:

vc1_ProfileSimple Simple profile
vc1_ProfileMain Main profile
vc1_ProfileReserved Reserved
vc1_ProfileAdvanced Advanced profile

enum vc1_eQuantizer

Quantizer mode enumeration

Enumeration values:

vc1_QuantizerImplicit Quantizer implied by quantizer step size
vc1_QuantizerExplicit Quantizer explicitly signaled
vc1_QuantizerNonUniform Non-uniform quantizer
vc1_QuantizerUniform Uniform quantizer

enum vc1_eQuantMode

Per macroblock quantizer step size enumeration

Enumeration values:

vc1_QuantModeDefault All macroblocks use PQUANT
vc1_QuantModeAllEdges Edge macroblocks use ALTPQUANT
vc1_QuantModeLeftTop Left/Top macroblocks use ALTPQUANT
vc1_QuantModeTopRight Top/Right macroblocks use ALTPQUANT
vc1_QuantModeRightBottom Right/Bottom macroblocks use ALTPQUANT
vc1_QuantModeBottomLeft Bottom/Left macroblocks use ALTPQUANT
vc1_QuantModeLeft Left macroblocks use ALTPQUANT
vc1_QuantModeTop Top macroblocks use ALTPQUANT
vc1_QuantModeRight Right macroblocks use ALTPQUANT
vc1_QuantModeBottom Bottom macroblocks use ALTPQUANT
vc1_QuantModeMBDual PQUANT/ALTPQUANT selected on macroblock basis
vc1_QuantModeMBAny Any QUANT selected on a macroblock basis

enum vc1_eResult

Function return codes.

Enumeration values:

vc1_ResultFatal Fatal condition detected
vc1_ResultWarn Continuable fault detected
vc1_ResultOK Function completed successfully
vc1_ResultNoFrame No frame to display (due to buffering)
vc1_ResultSlice More slice data required to complete decode
vc1_ResultField Second field data required to complete decode
vc1_ResultInvalidParameter Option value not accepted
vc1_ResultBadFile File will not open
vc1_ResultBadLine Line in option file not parseable
vc1_ResultBadType Invalid type for parameter
vc1_ResultNoMemory malloc() failed
vc1_ResultNoData Failed to read picture data
vc1_ResultBufferExhausted No more data to read from buffer
vc1_ResultBadImageSize Size breaks restrictions set by the standard
vc1_ResultImageTooBig Size bigger than profile limits
vc1_ResultImageSizeChanged Size changed in simple or main profile
vc1_ResultUnsupportedTransform Invalid transform
vc1_ResultACRunLevelDecodeFailed Decoder failed to read the AC coef run levels
vc1_ResultNoStartCode Start code not found in file input data

enum vc1_eSBP

Subblock pattern enumeration for TTMB and TTBLK

Enumeration values:

vc1_SBP8x8 8x8 transform, coded
vc1_SBP8x4Bottom 8x4 transform, bottom subblock coded
vc1_SBP8x4Top 8x4 transform, top subblock coded
vc1_SBP8x4Both 8x4 transform, both subblocks coded
vc1_SBP4x8Right 4x8 transform, right subblock coded
vc1_SBP4x8Left 4x8 transform, left subblock coded
vc1_SBP4x8Both 4x8 transform, both subblocks coded
vc1_SBP4x4 4x4 transform, subblock pattern separate
vc1_SBP8x8MB 8x8 transform, coded, whole MB
vc1_SBP8x4BottomMB 8x4 transform, bottom subblock coded, whole MB
vc1_SBP8x4TopMB 8x4 transform, top subblock coded, whole MB
vc1_SBP8x4BothMB 8x4 transform, both subblocks coded, whole MB
vc1_SBP4x8RightMB 4x8 transform, right subblock coded, whole MB
vc1_SBP4x8LeftMB 4x8 transform, left subblock coded, whole MB
vc1_SBP4x8BothMB 4x8 transform, both subblocks coded, whole MB
vc1_SBP4x4MB 4x4 transform, subblocks pattern separate, whole MB
vc1_SBPMBLevel MB level threshold

9.34 vc1zztab.h File Reference

Variables

- const BYTE8 vc1_Inv_Intra_Normal_Scan [64]
- const BYTE8 vc1_Inv_Intra_Horizontal_Scan [64]
- const BYTE8 vc1_Inv_Intra_Vertical_Scan [64]
- const BYTE8 vc1_Inv_Inter_8x8_Scan_Simple_Main_Profiles_Progressive_Advanced_Profile [64]
- const BYTE8 vc1_Inv_Inter_8x4_Scan_Simple_Main_Profiles [32]
- const BYTE8 vc1_Inv_Inter_4x8_Scan_Simple_Main_Profiles [32]
- const BYTE8 vc1_Inv_Inter_4x4_Scan_Simple_Main_Profiles_Progressive_Advanced_Profile [16]
- const BYTE8 vc1_Inv_Progressive_Inter_8x4_Scan_Advanced_Profile [32]
- const BYTE8 vc1_Inv_Progressive_Inter_4x8_Scan_Advanced_Profile [32]
- const BYTE8 vc1_Inv_Interlace_Inter_8x8_Scan_Advanced_Profile [64]
- const BYTE8 vc1_Inv_Interlace_Inter_8x4_Scan_Advanced_Profile [32]
- const BYTE8 vc1_Inv_Interlace_Inter_4x8_Scan_Advanced_Profile [32]
- const BYTE8 vc1_Inv_Interlace_Inter_4x4_Scan_Advanced_Profile [16]

Detailed Description

This file contains the zigzag tables