INTERNATIONAL STANDARD

**ISO/IEC 13818-2**

Second edition
2000-12-15

# Information technology — Generic coding of moving pictures and associated audio information: Video

*Technologies de l'information — Codage générique des images animées et du son associé: Données vidéo*

Reference number
ISO/IEC 13818-2:2000(E)

© ISO/IEC 2000



SAMSUNG-1016

**PDF disclaimer**

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2000

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

# CONTENTS


| | | Page |
|---|---|---|
| | Intro. 1 Purpose | vi |
| | Intro. 2 Application | vi |
| | Intro. 3 Profiles and levels | vi |
| | Intro. 4 The scalable and the non-scalable syntax | vii |
| 1 | Scope | 1 |
| 2 | Normative references | 1 |
| 3 | Definitions | 2 |
| 4 | Abbreviations and symbols | 7 |
| | 4.1 Arithmetic operators | 7 |
| | 4.2 Logical operators | 8 |
| | 4.3 Relational operators | 8 |
| | 4.4 Bitwise operators | 8 |
| | 4.5 Assignment | 8 |
| | 4.6 Mnemonics | 8 |
| | 4.7 Constants | 9 |
| 5 | Conventions | 9 |
| | 5.1 Method of describing bitstream syntax | 9 |
| | 5.2 Definition of functions | 10 |
| | 5.3 Reserved, forbidden and marker_bit | 10 |
| | 5.4 Arithmetic precision | 11 |
| 6 | Video bitstream syntax and semantics | 11 |
| | 6.1 Structure of coded video data | 11 |
| | 6.2 Video bitstream syntax | 21 |
| | 6.3 Video bitstream semantics | 36 |
| 7 | The video decoding process | 61 |
| | 7.1 Higher syntactic structures | 61 |
| | 7.2 Variable length decoding | 62 |
| | 7.3 Inverse scan | 64 |
| | 7.4 Inverse quantisation | 66 |
| | 7.5 Inverse DCT | 69 |
| | 7.6 Motion compensation | 69 |
| | 7.7 Spatial scalability | 83 |
| | 7.8 SNR scalability | 92 |
| | 7.9 Temporal scalability | 99 |
| | 7.10 Data partitioning | 102 |
| | 7.11 Hybrid scalability | 103 |
| | 7.12 Output of the decoding process | 104 |
| 8 | Profiles and levels | 106 |
| | 8.1 ISO/IEC 11172-2 compatibility | 109 |
| | 8.2 Relationship between defined profiles | 109 |
| | 8.3 Relationship between defined levels | 111 |
| | 8.4 Scalable layers | 111 |
| | 8.5 Parameter values for defined profiles, levels and layers | 114 |
| | 8.6 Compatibility requirements on decoders | 115 |
| 9 | Registration of Copyright Identifiers | 117 |
| | 9.1 General | 117 |
| | 9.2 Implementation of a Registration Authority (RA) | 118 |


© ISO/IEC 2000 – All rights reserved

*Page*

Annex A – Inverse discrete transform ..... 119

Annex B – Variable length code tables ..... 121
B.1 Macroblock addressing ..... 121
B.2 Macroblock type ..... 122
B.3 Macroblock pattern ..... 127
B.4 Motion vectors ..... 128
B.5 DCT coefficients ..... 129

Annex C – Video buffering verifier ..... 138

Annex D – Features supported by the algorithm ..... 143
D.1 Overview ..... 143
D.2 Video formats ..... 143
D.3 Picture quality ..... 144
D.4 Data rate control ..... 144
D.5 Low delay mode ..... 144
D.6 Random access/channel hopping ..... 145
D.7 Scalability ..... 145
D.8 Compatibility ..... 151
D.9 Differences between this Specification and ISO/IEC 11172-2 ..... 151
D.10 Complexity ..... 154
D.11 Editing encoded bitstreams ..... 154
D.12 Trick modes ..... 154
D.13 Error resilience ..... 155
D.14 Concatenated sequences ..... 162

Annex E – Profile and level restrictions ..... 163
E.1 Syntax element restrictions in profiles ..... 163
E.2 Permissible layer combinations ..... 175

Annex F – Bibliography ..... 197

Annex G – Registration Procedure ..... 198
G.1 Procedure for the request of a Registered Identifier (RID) ..... 198
G.2 Responsibilities of the Registration Authority ..... 198
G.3 Responsibilities of parties requesting an RID ..... 198
G.4 Appeal procedure for denied applications ..... 199

Annex H – Registration Application Form ..... 200
H.1 Contact information of organization requesting a Registered Identifier (RID) ..... 200
H.2 Statement of an intention to apply the assigned RID ..... 200
H.3 Date of intended implementation of the RID ..... 200
H.4 Authorized representative ..... 200
H.5 For official use only of the Registration Authority ..... 200

Annex J – 4:2:2 Profile test results ..... 202
J.1 Introduction ..... 202

Annex K – Patents ..... 207

 © ISO/IEC 2000 – All rights reserved

## Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 13818 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 13818-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*, in collaboration with ITU-T. The identical text is published as ITU-T Rec. H.262.

This second edition cancels and replaces the first edition (ISO/IEC 13818-2:1996), which has been technically revised.

ISO/IEC 13818 consists of the following parts, under the general title *Information technology — Generic coding of moving pictures and associated audio information*:

— *Part 1: Systems*

— *Part 2: Video*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Software simulation*

— *Part 6: Extensions for DSM-CC*

— *Part 7: Advanced Audio Coding (AAC)*

— *Part 9: Extension for real time interface for systems decoders*

— *Part 10: Conformance extensions for Digital Storage Media Command and Control (DSM-CC)*

Annexes A, B and C form a normative part of this part of ISO/IEC 13818. Annexes D to K are for information only.

© ISO/IEC 2000 – All rights reserved

**Introduction**

## Intro. 1 Purpose

This Part of this Recommendation | International Standard was developed in response to the growing need for a generic coding method of moving pictures and of associated sound for various applications such as digital storage media, television broadcasting and communication. The use of this Specification means that motion video can be manipulated as a form of computer data and can be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

## Intro. 2 Application

The applications of this Specification cover, but are not limited to, such areas as listed below:

| | |
|---|---|
| BSS | Broadcasting Satellite Service (to the home) |
| CATV | Cable TV Distribution on optical networks, copper, etc. |
| CDAD | Cable Digital Audio Distribution |
| DSB | Digital Sound Broadcasting (terrestrial and satellite broadcasting) |
| DTTB | Digital Terrestrial Television Broadcasting |
| EC | Electronic Cinema |
| ENG | Electronic News Gathering (including SNG, Satellite News Gathering) |
| FSS | Fixed Satellite Service (e.g. to head ends) |
| HTT | Home Television Theatre |
| IPC | Interpersonal Communications (videoconferencing, videophone, etc.) |
| ISM | Interactive Storage Media (optical disks, etc.) |
| MMM | Multimedia Mailing |
| NCA | News and Current Affairs |
| NDB | Networked Database Services (via ATM, etc.) |
| RVS | Remote Video Surveillance |
| SSM | Serial Storage Media (digital VTR, etc.) |

## Intro. 3 Profiles and levels

This Specification is intended to be generic in the sense that it serves a wide range of applications, bitrates, resolutions, qualities and services. Applications should cover, among other things, digital storage media, television broadcasting and communications. In the course of creating this Specification, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and they have been integrated into a single syntax. Hence, this Specification will facilitate the bitstream interchange among different applications.

Considering the practicality of implementing the full syntax of this Specification, however, a limited number of subsets of the syntax are also stipulated by means of "profile" and "level". These and other related terms are formally defined in clause 3.

A "profile" is a defined subset of the entire bitstream syntax that is defined by this Specification. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by parameters in the bitstream. For instance, it is possible to specify frame sizes as large as (approximately) $2^{14}$ samples wide by $2^{14}$ lines high. It is currently neither practical nor economic to implement a decoder capable of dealing with all possible frame sizes.

In order to deal with this problem, "levels" are defined within each profile. A level is a defined set of constraints imposed on parameters in the bitstream. These constraints may be simple limits on numbers. Alternatively they may take the form of constraints on arithmetic combinations of the parameters (e.g. frame width multiplied by frame height multiplied by frame rate).

Bitstreams complying with this Specification use a common syntax. In order to achieve a subset of the complete syntax, flags and parameters are included in the bitstream that signal the presence or otherwise of syntactic elements that occur later in the bitstream. In order to specify constraints on the syntax (and hence define a profile), it is thus only necessary to constrain the values of these flags and parameters that specify the presence of later syntactic elements.

 © ISO/IEC 2000 – All rights reserved

## Intro. 4 The scalable and the non-scalable syntax

The full syntax can be divided into two major categories: One is the non-scalable syntax, which is structured as a super set of the syntax defined in ISO/IEC 11172-2. The main feature of the non-scalable syntax is the extra compression tools for interlaced video signals. The second is the scalable syntax, the key property of which is to enable the reconstruction of useful video from pieces of a total bitstream. This is achieved by structuring the total bitstream in two or more layers, starting from a standalone base layer and adding a number of enhancement layers. The base layer can use the non-scalable syntax, or in some situations conform to the ISO/IEC 11172-2 syntax.

### Intro. 4.1 Overview of the non-scalable syntax

The coded representation defined in the non-scalable syntax achieves a high compression ratio while preserving good image quality. The algorithm is not lossless as the exact sample values are not preserved during coding. Obtaining good image quality at the bitrates of interest demands very high compression, which is not achievable with intra picture coding alone. The need for random access, however, is best satisfied with pure intra picture coding. The choice of the techniques is based on the need to balance a high image quality and compression ratio with the requirement to make random access to the coded bitstream.

A number of techniques are used to achieve high compression. The algorithm first uses block-based motion compensation to reduce the temporal redundancy. Motion compensation is used both for causal prediction of the current picture from a previous picture, and for non-causal, interpolative prediction from past and future pictures. Motion vectors are defined for each 16-sample by 16-line region of the picture. The prediction error, is further compressed using the Discrete Cosine Transform (DCT) to remove spatial correlation before it is quantised in an irreversible process that discards the less important information. Finally, the motion vectors are combined with the quantised DCT information, and encoded using variable length codes.

#### Intro. 4.1.1 Temporal processing

Because of the conflicting requirements of random access and highly efficient compression, three main picture types are defined. Intra Coded Pictures (I-Pictures) are coded without reference to other pictures. They provide access points to the coded sequence where decoding can begin, but are coded with only moderate compression. Predictive Coded Pictures (P-Pictures) are coded more efficiently using motion compensated prediction from a past intra or predictive coded picture and are generally used as a reference for further prediction. Bidirectionally-predictive Coded Pictures (B-Pictures) provide the highest degree of compression but require both past and future reference pictures for motion compensation. Bidirectionally-predictive coded pictures are never used as references for prediction (except in the case that the resulting picture is used as a reference in a spatially scalable enhancement layer). The organisation of the three picture types in a sequence is very flexible. The choice is left to the encoder and will depend on the requirements of the application. Figure Intro. 1 illustrates an example of the relationship among the three different picture types.

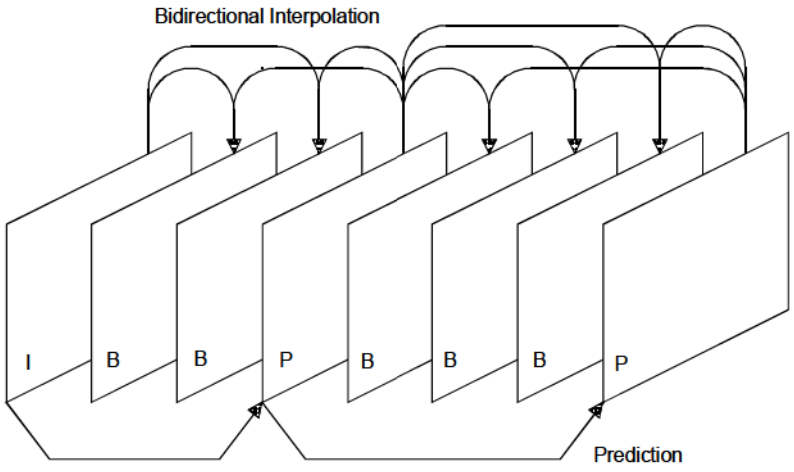



**Figure Intro. 1 – Example of temporal picture structure**

© ISO/IEC 2000 – All rights reserved

### Intro. 4.1.2 Coding interlaced video

Each frame of interlaced video consists of two fields which are separated by one field-period. The Specification allows either the frame to be encoded as picture or the two fields to be encoded as two pictures. Frame encoding or field encoding can be adaptively selected on a frame-by-frame basis. Frame encoding is typically preferred when the video scene contains significant detail with limited motion. Field encoding, in which the second field can be predicted from the first, works better when there is fast movement.

### Intro. 4.1.3 Motion representation – Macroblocks

As in ISO/IEC 11172-2, the choice of 16 by 16 macroblocks for the motion-compensation unit is a result of the trade-off between the coding gain provided by using motion information and the overhead needed to represent it. Each macroblock can be temporally predicted in one of a number of different ways. For example, in frame encoding, the prediction from the previous reference frame can itself be either frame-based or field-based. Depending on the type of the macroblock, motion vector information and other side information is encoded with the compressed prediction error in each macroblock. The motion vectors are encoded differentially with respect to the last encoded motion vectors using variable length codes. The maximum length of the motion vectors that may be represented can be programmed, on a picture-by-picture basis, so that the most demanding applications can be met without compromising the performance of the system in more normal situations.

It is the responsibility of the encoder to calculate appropriate motion vectors. This Specification does not specify how this should be done.

### Intro. 4.1.4 Spatial redundancy reduction

Both source pictures and prediction errors have high spatial redundancy. This Specification uses a block-based DCT method with visually weighted quantisation and run-length coding. After motion compensated prediction or interpolation, the resulting prediction error is split into 8 by 8 blocks. These are transformed into the DCT domain where they are weighted before being quantised. After quantisation many of the DCT coefficients are zero in value and so two-dimensional run-length and variable length coding is used to encode the remaining DCT coefficients efficiently.

### Intro. 4.1.5 Chrominance formats

In addition to the 4:2:0 format supported in ISO/IEC 11172-2 this Specification supports 4:2:2 and 4:4:4 chrominance formats.

### Intro. 4.2 Scalable extensions

The scalability tools in this Specification are designed to support applications beyond that supported by single layer video. Among the noteworthy applications areas addressed are video telecommunications, video on Asynchronous Transfer Mode (ATM) networks, interworking of video standards, video service hierarchies with multiple spatial, temporal and quality resolutions, HDTV with embedded TV, systems allowing migration to higher temporal resolution HDTV, etc. Although a simple solution to scalable video is the simulcast technique which is based on transmission/storage of multiple independently coded reproductions of video, a more efficient alternative is scalable video coding, in which the bandwidth allocated to a given reproduction of video can be partially re-utilised in coding of the next reproduction of video. In scalable video coding, it is assumed that given a coded bitstream, decoders of various complexities can decode and display appropriate reproductions of coded video. A scalable video encoder is likely to have increased complexity when compared to a single layer encoder. However, this Recommendation | International Standard provides several different forms of scalabilities that address non-overlapping applications with corresponding complexities. The basic scalability tools offered are:

- data partitioning;
- SNR scalability;
- spatial scalability; and
- temporal scalability.

Moreover, combinations of these basic scalability tools are also supported and are referred to as *hybrid scalability*. In the case of basic scalability, two layers of video referred to as the *lower layer* and the *enhancement layer* are allowed, whereas in hybrid scalability up to three layers are supported. Tables Intro. 1 to Intro. 3 provide a few example applications of various scalabilities.

 © ISO/IEC 2000 – All rights reserved

**Table Intro. 1 – Applications of SNR scalability**

| Lower layer | Enhancement layer | Application |
|---|---|---|
| Recommendation ITU-R BT.601 | Same resolution and format as lower layer | Two quality service for Standard TV (SDTV) |
| High Definition | Same resolution and format as lower layer | Two quality service for HDTV |
| 4:2:0 high definition | 4:2:2 chroma simulcast | Video production / distribution |

**Table Intro. 2 – Applications of spatial scalability**

| Base | Enhancement | Application |
|---|---|---|
| Progressive (30 Hz) | Progressive (30 Hz) | Compatibility or scalability CIF/SCIF |
| Interlace (30 Hz) | Interlace (30 Hz) | HDTV/SDTV scalability |
| Progressive (30 Hz) | Interlace (30 Hz) | ISO/IEC 11172-2/compatibility with this Specification |
| Interlace (30 Hz) | Progressive (60 Hz) | Migration to high resolution progressive HDTV |

**Table Intro. 3 – Applications of temporal scalability**

| Base | Enhancement | Higher | Application |
|---|---|---|---|
| Progressive (30 Hz) | Progressive (30 Hz) | Progressive (60 Hz) | Migration to high resolution progressive HDTV |
| Interlace (30 Hz) | Interlace (30 Hz) | Progressive (60 Hz) | Migration to high resolution progressive HDTV |

### Intro. 4.2.1 Spatial scalable extension

Spatial scalability is a tool intended for use in video applications involving telecommunications, interworking of video standards, video database browsing, interworking of HDTV and TV, etc., i.e. video systems with the primary common feature that a minimum of two layers of spatial resolution are necessary. Spatial scalability involves generating two spatial resolution video layers from a single video source such that the lower layer is coded by itself to provide the basic spatial resolution and the enhancement layer employs the spatially interpolated lower layer and carries the full spatial resolution of the input video source. The lower and the enhancement layers may either both use the coding tools in this Specification, or the ISO/IEC 11172-2 Standard for the lower layer and this Specification for the enhancement layer. The latter case achieves a further advantage by facilitating interworking between video coding standards. Moreover, spatial scalability offers flexibility in choice of video formats to be employed in each layer. An additional advantage of spatial scalability is its ability to provide resilience to transmission errors as the more important data of the lower layer can be sent over channel with better error performance, while the less critical enhancement layer data can be sent over a channel with poor error performance.

### Intro. 4.2.2 SNR scalable extension

SNR scalability is a tool intended for use in video applications involving telecommunications, video services with multiple qualities, standard TV and HDTV, i.e. video systems with the primary common feature that a minimum of two layers of video quality are necessary. SNR scalability involves generating two video layers of same spatial resolution but different video qualities from a single video source such that the lower layer is coded by itself to provide the basic video quality and the enhancement layer is coded to enhance the lower layer. The enhancement layer when added back to the

© ISO/IEC 2000 – All rights reserved

lower layer regenerates a higher quality reproduction of the input video. The lower and the enhancement layers may either use this Specification or ISO/IEC 11172-2 Standard for the lower layer and this Specification for the enhancement layer. An additional advantage of SNR scalability is its ability to provide high degree of resilience to transmission errors as the more important data of the lower layer can be sent over channel with better error performance, while the less critical enhancement layer data can be sent over a channel with poor error performance.

### Intro. 4.2.3 Temporal scalable extension

Temporal scalability is a tool intended for use in a range of diverse video applications from telecommunications to HDTV for which migration to higher temporal resolution systems from that of lower temporal resolution systems may be necessary. In many cases, the lower temporal resolution video systems may be either the existing systems or the less expensive early generation systems, with the motivation of introducing more sophisticated systems gradually. Temporal scalability involves partitioning of video frames into layers, whereas the lower layer is coded by itself to provide the basic temporal rate and the enhancement layer is coded with temporal prediction with respect to the lower layer, these layers when decoded and temporal multiplexed to yield full temporal resolution of the video source. The lower temporal resolution systems may only decode the lower layer to provide basic temporal resolution, whereas more sophisticated systems of the future may decode both layers and provide high temporal resolution video while maintaining interworking with earlier generation systems. An additional advantage of temporal scalability is its ability to provide resilience to transmission errors as the more important data of the lower layer can be sent over channel with better error performance, while the less critical enhancement layer can be sent over a channel with poor error performance.

### Intro. 4.2.4 Data partitioning extension

Data partitioning is a tool intended for use when two channels are available for transmission and/or storage of a video bitstream, as may be the case in ATM networks, terrestrial broadcast, magnetic media, etc. The bitstream is partitioned between these channels such that more critical parts of the bitstream (such as headers, motion vectors, low frequency DCT coefficients) are transmitted in the channel with the better error performance, and less critical data (such as higher frequency DCT coefficients) is transmitted in the channel with poor error performance. Thus, degradation to channel errors are minimised since the critical parts of a bitstream are better protected. Data from neither channel may be decoded on a decoder that is not intended for decoding data partitioned bitstreams.

 © ISO/IEC 2000 – All rights reserved

**INTERNATIONAL STANDARD**

**ITU-T RECOMMENDATION**

# INFORMATION TECHNOLOGY – GENERIC CODING OF MOVING PICTURES AND ASSOCIATED AUDIO INFORMATION: VIDEO

## 1 Scope

This Recommendation | International Standard specifies the coded representation of picture information for digital storage media and digital video communication and specifies the decoding process. The representation supports constant bitrate transmission, variable bitrate transmission, random access, channel hopping, scalable decoding, bitstream editing, as well as special functions such as fast forward playback, fast reverse playback, slow motion, pause and still pictures. This Recommendation | International Standard is forward compatible with ISO/IEC 11172-2 and upward or downward compatible with EDTV, HDTV, SDTV formats.

This Recommendation | International Standard is primarily applicable to digital storage media, video broadcast and communication. The storage media may be directly connected to the decoder, or via communications means such as busses, LANs, or telecommunications links.

## 2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of ITU maintains a list of currently valid ITU-T Recommendations.

- Recommendation ITU-R BT.601-3 (1992), *Encoding parameters of digital television for studios*.
- Recommendation ITU-R BR.648 (1986), *Digital recording of audio signals*.
- Report ITU-R 955-2 (1990), *Satellite sound broadcasting with portable receivers and receivers in automobiles*.
- ISO/IEC 11172-1:1993, *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1 : Systems.*
- ISO/IEC 11172-2:1993, *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2 : Video.*
- ISO/IEC 11172-3:1993, *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3 : Audio.*
- IEEE 1180:1990, Standard Specifications for the Implementations of 8 by 8 Inverse Discrete Cosine Transform.
- IEC 60908 (1999), *Audio recording – Compact disc digital audio system*.
- IEC 60461 (1986), *Time and control code for video tape recorders.*
- ITU-T Recommendation H.261 (1993), *Video codec for audiovisual services at p × 64 kbit/s.*
- ITU-T Recommendation H.320 (1999), *Narrow-band visual telephone systems and terminal equipment.*
- CCITT Recommendation T.81 (1992) ISO/IEC 10918-1:1994, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines.* (JPEG.)

## 3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

**3.1 AC coefficient**: Any DCT coefficient for which the frequency in one or both dimensions is non-zero.

**3.2 big picture**: A coded picture that would cause VBV buffer underflow as defined in C.7. Big pictures can only occur in sequences where low_delay is equal to 1. "Skipped picture" is a term that is sometimes used to describe the same concept.

**3.3 B-field picture**: A field structure B-Picture.

**3.4 B-frame picture**: A frame structure B-Picture.

**3.5 B-picture; bidirectionally predictive-coded picture**: A picture that is coded using motion compensated prediction from past and/or future reference fields or frames.

**3.6 backward compatibility**: A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard.

**3.7 backward motion vector**: A motion vector that is used for motion compensation from a reference frame or reference field at a later time in display order.

**3.8 backward prediction**: Prediction from the future reference frame (field).

**3.9 base layer**: First, independently decodable layer of a scalable hierarchy.

**3.10 bitstream; stream**: An ordered series of bits that forms the coded representation of the data.

**3.11 bitrate**: The rate at which the coded bitstream is delivered from the storage medium to the input of a decoder.

**3.12 block**: An 8-row by 8-column matrix of samples, or 64 DCT coefficients (source, quantised or dequantised).

**3.13 bottom field**: One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.

**3.14 byte aligned**: A bit in a coded bitstream is byte-aligned if its position is a multiple of 8 bits from the first bit in the stream.

**3.15 byte**: Sequence of 8 bits.

**3.16 channel**: A digital medium that stores or transports a bitstream constructed according to ITU-T Rec. H.262 | ISO/IEC 13818-2.

**3.17 chrominance format**: Defines the number of chrominance blocks in a macroblock.

**3.18 chroma simulcast**: A type of scalability (which is a subset of SNR scalability) where the enhancement layer(s) contain only coded refinement data for the DC coefficients, and all the data for the AC coefficients, of the chrominance components.

**3.19 chrominance component**: A matrix, block or single sample representing one of the two colour difference signals related to the primary colours in the manner defined in the bitstream. The symbols used for the chrominance signals are Cr and Cb.

**3.20 coded B-frame**: A B-frame picture or a pair of B-field pictures.

**3.21 coded frame**: A coded frame is a coded I-frame, a coded P-frame or a coded B-frame.

**3.22 coded I-frame**: An I-frame picture or a pair of field pictures, where the first field picture is an I-picture and the second field picture is an I-picture or a P-picture.

**3.23 coded P-frame**: A P-frame picture or a pair of P-field pictures.

**3.24 coded picture**: A coded picture is made of a picture header, the optional extensions immediately following it, and the following picture data. A coded picture may be a coded frame or a coded field.

**3.25 coded video bitstream**: A coded representation of a series of one or more pictures as defined in ITU-T Rec. H.262 | ISO/IEC 13818-2.

**3.26 coded order**: The order in which the pictures are transmitted and decoded. This order is not necessarily the same as the display order.

**3.27 coded representation**: A data element as represented in its encoded form.

**3.28 coding parameters**: The set of user-definable parameters that characterise a coded video bitstream. Bitstreams are characterised by coding parameters. Decoders are characterised by the bitstreams that they are capable of decoding.

**3.29 component**: A matrix, block or single sample from one of the three matrices (luminance and two chrominance) that make up a picture.

**3.30 compression**: Reduction in the number of bits used to represent an item of data.

**3.31 constant bitrate coded video**: A coded video bitstream with a constant bitrate.

**3.32 constant bitrate**: Operation where the bitrate is constant from start to finish of the coded bitstream.

**3.33 data element**: An item of data as represented before encoding and after decoding.

**3.34 data partitioning**: A method for dividing a bitstream into two separate bitstreams for error resilience purposes. The two bitstreams have to be recombined before decoding.

**3.35 D-Picture**: A type of picture that shall not be used except in ISO/IEC 11172-2.

**3.36 DC coefficient**: The DCT coefficient for which the frequency is zero in both dimensions.

**3.37 DCT coefficient**: The amplitude of a specific cosine basis function.

**3.38 decoder input buffer**: The First-In First-Out (FIFO) buffer specified in the video buffering verifier.

**3.39 decoder**: An embodiment of a decoding process.

**3.40 decoding (process)**: The process defined in ITU-T Rec. H.262 | ISO/IEC 13818-2 that reads an input coded bitstream and produces decoded pictures.

**3.41 dequantisation**: The process of rescaling the quantised DCT coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse DCT.

**3.42 digital storage media (DSM)**: A digital storage or transmission device or system.

**3.43 discrete cosine transform (DCT)**: Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation. The inverse DCT is defined in Annex A of ITU-T Rec. H.262 | ISO/IEC 13818-2.

**3.44 display aspect ratio**: The ratio height/width (in spatial measurement units such as centimeters) of the intended display.

**3.45 display order**: The order in which the decoded pictures are displayed. Normally this is the same order in which they were presented at the input of the encoder.

**3.46 display process**: The (non-normative) process by which reconstructed frames are displayed.

**3.47 dual-prime prediction**: A prediction mode in which two forward field-based predictions are averaged. The predicted block size is $16 \times 16$ luminance samples.

**3.48 editing**: The process by which one or more coded bitstreams are manipulated to produce a new coded bitstream. Conforming edited bitstreams must meet the requirements defined in ITU-T Rec. H.262 | ISO/IEC 13818-2.

**3.49 encoder**: An embodiment of an encoding process.

**3.50 encoding (process)**: A process, not specified in ITU-T Rec. H.262 | ISO/IEC 13818-2, that reads a stream of input pictures and produces a valid coded bitstream as defined in ITU-T Rec. H.262 | ISO/IEC 13818-2.

**3.51 enhancement layer**: A relative reference to a layer (above the base layer) in a scalable hierarchy. For all forms of scalability, its decoding process can be described by reference to the lower layer decoding process and the appropriate additional decoding process for the enhancement layer itself.

**3.52 fast forward playback**: The process of displaying a sequence, or parts of a sequence, of pictures in display-order faster than real-time.

**3.53 fast reverse playback**: The process of displaying the picture sequence in the reverse of display order faster than real-time.

**3.54 field**: For an interlaced video signal, a "field" is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.

**3.55 field-based prediction**: A prediction mode using only one field of the reference frame. The predicted block size is 16 × 16 luminance samples.

**3.56 field period**: The reciprocal of twice the frame rate.

**3.57 field picture; field structure picture**: A field structure picture is a coded picture with picture_structure is equal to "Top field" or "Bottom field".

**3.58 flag**: A one bit integer variable which may take one of only two values (zero and one).

**3.59 forbidden**: The term "forbidden" when used in the clauses defining the coded bitstream indicates that the value shall never be used. This is usually to avoid emulation of start codes.

**3.60 forced updating**: The process by which macroblocks are intra-coded from time-to-time to ensure that mismatch errors between the inverse DCT processes in encoders and decoders cannot build up excessively.

**3.61 forward compatibility**: A newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstreams of the older coding standard.

**3.62 forward motion vector**: A motion vector that is used for motion compensation from a reference frame or reference field at an earlier time in display order.

**3.63 forward prediction**: Prediction from the past reference frame (field).

**3.64 frame**: A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame. For interlaced video, a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other.

**3.65 frame-based prediction**: A prediction mode using both fields of the reference frame.

**3.66 frame period**: The reciprocal of the frame rate.

**3.67 frame picture; frame structure picture**: A frame structure picture is a coded picture with picture_structure is equal to "Frame".

**3.68 frame rate**: The rate at which frames are output from the decoding process.

**3.69 future reference frame (field)**: A future reference frame (field) is a reference frame (field) that occurs at a later time than the current picture in display order.

**3.70 frame re-ordering**: The process of re-ordering the reconstructed frames when the coded order is different from the display order. Frame re-ordering occurs when B-frames are present in a bitstream. There is no frame re-ordering when decoding low delay bitstreams.

**3.71 group of pictures**: A notion defined only in ISO/IEC 11172-2 (MPEG-1 Video). In ITU-T Rec. H.262 | ISO/IEC 13818-2, a similar functionality can be achieved by the mean of inserting group of pictures headers.

**3.72 header**: A block of data in the coded bitstream containing the coded representation of a number of data elements pertaining to the coded data that follow the header in the bitstream.

**3.73 hybrid scalability**: Hybrid scalability is the combination of two (or more) types of scalability.

**3.74 interlace**: The property of conventional television frames where alternating lines of the frame represent different instances in time. In an interlaced frame, one of the field is meant to be displayed first. This field is called the first field. The first field can be the top field or the bottom field of the frame.

**3.75 I-field picture**: A field structure I-Picture.

**3.76 I-frame picture**: A frame structure I-Picture.

**3.77 I-picture; intra-coded picture**: A picture coded using information only from itself.

**3.78 intra coding**: Coding of a macroblock or picture that uses information only from that macroblock or picture.

**3.78.1 Inverse DCT, IDCT**: Inverse discrete cosine transform, as defined in Annex A.

**3.79 level**: A defined set of constraints on the values which may be taken by the parameters of ITU-T Rec. H.262 | ISO/IEC 13818-2 within a particular profile. A profile may contain one or more levels. In a different context, level is the absolute value of a non-zero coefficient (see "run").

**3.80 layer**: In a scalable hierarchy denotes one out of the ordered set of bitstreams and (the result of) its associated decoding process (implicitly including decoding of **all** layers below this layer).

**3.81 layer bitstream**: A single bitstream associated to a specific layer (always used in conjunction with layer qualifiers, e. g. "enhancement layer bitstream").

**3.82 lower layer**: A relative reference to the layer immediately below a given enhancement layer (implicitly including decoding of **all** layers below this enhancement layer).

**3.83 luminance component**: A matrix, block or single sample representing a monochrome representation of the signal and related to the primary colours in the manner defined in the bitstream. The symbol used for luminance is Y.

**3.84 Mbit**: 1 000 000 bits.

**3.85 macroblock**: The four 8 by 8 blocks of luminance data and the two (for 4:2:0 chrominance format), four (for 4:2:2 chrominance format) or eight (for 4:4:4 chrominance format) corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the sample data and sometimes to the coded representation of the sample values and other data elements defined in the macroblock header of the syntax defined in ITU-T Rec. H.262 | ISO/IEC 13818-2. The usage is clear from the context.

**3.86 motion compensation**: The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past and/or future reference frames or reference fields containing previously decoded sample values that are used to form the prediction error.

**3.87 motion estimation**: The process of estimating motion vectors during the encoding process.

**3.88 motion vector**: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture or field to the coordinates in a reference frame or reference field.

**3.89 non-intra coding**: Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.

**3.90 opposite parity**: The opposite parity of top is bottom, and vice versa.

**3.91 P-field picture**: A field structure P-Picture.

**3.92 P-frame picture**: A frame structure P-Picture.

**3.93 P-picture; predictive-coded picture**: A picture that is coded using motion compensated prediction from past reference fields or frame.

**3.94 parameter**: A variable within the syntax of ITU-T Rec. H.262 | ISO/IEC 13818-2 which may take one of a range of values. A variable which can take one of only two values is called a flag.

**3.95 parity (of field)**: The parity of a field can be top or bottom.

**3.96 past reference frame (field)**: A past reference frame (field) is a reference frame (field) that occurs at an earlier time than the current picture in display order.

**3.97 picture**: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. A "coded picture" is defined in 3.21 of ITU-T Rec. H.262 | ISO/IEC 13818-2. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

**3.98 picture data**: In the VBV operations, picture data is defined as all the bits of the coded picture, all the header(s) and user data immediately preceding it if any (including any stuffing between them) and all the stuffing following it, up to (but not including) the next start code, except in the case where the next start code is an end of sequence code, in which case it is included in the picture data.

**3.99 prediction**: The use of a predictor to provide an estimate of the sample value or data element currently being decoded.

**3.100 prediction error**: The difference between the actual value of a sample or data element and its predictor.

**3.101 predictor**: A linear combination of previously decoded sample values or data elements.

**3.102 profile**: A defined subset of the syntax of ITU-T Rec. H.262 | ISO/IEC 13818-2.

NOTE – In ITU-T Rec. H.262 | ISO/IEC 13818-2, the word "profile" is used as defined above. It should not be confused with other definitions of "profile" and in particular it does not have the meaning that is defined by JTC1/SGFS.

**3.103 progressive**: The property of film frames where all the samples of the frame represent the same instances in time.

**3.104 quantisation matrix**: A set of sixty-four 8-bit values used by the dequantiser.

**3.105 quantised DCT coefficients**: DCT coefficients before dequantisation. A variable length coded representation of quantised DCT coefficients is transmitted as part of the coded video bitstream.

**3.106 quantiser scale**: A scale factor coded in the bitstream and used by the decoding process to scale the dequantisation.

**3.107 random access**: The process of beginning to read and decode the coded bitstream at an arbitrary point.

**3.108 reconstructed frame**: A reconstructed frame consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. A reconstructed frame is obtained by decoding a coded frame.

**3.109 reconstructed picture**: A reconstructed picture is obtained by decoding a coded picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a reconstructed frame (when decoding a field picture). If the coded picture is a field picture, then the reconstructed picture is the top field or the bottom field of the reconstructed frame.

**3.110 reference field**: A reference field is one field of a reconstructed frame. Reference fields are used for forward and backward prediction when P-pictures and B-pictures are decoded. Note that when field P-pictures are decoded, prediction of the second field P-picture of a coded frame uses the first reconstructed field of the same coded frame as a reference field.

**3.111 reference frame**: A reference frame is a reconstructed frame that was coded in the form of a coded I-frame or a coded P-frame. Reference frames are used for forward and backward prediction when P-pictures and B-pictures are decoded.

**3.112 re-ordering delay**: A delay in the decoding process that is caused by frame re-ordering.

**3.113 reserved**: The term "reserved" when used in the clauses defining the coded bitstream, indicates that the value may be used in the future for ITU-T | ISO/IEC defined extensions.

**3.114 sample aspect ratio (SAR)**: This specifies the relative distance between samples. It is defined (for the purposes of ITU-T Rec. H.262 | ISO/IEC 13818-2), as the vertical displacement of the lines of luminance samples in a frame divided by the horizontal displacement of the luminance samples. Thus, its units are (metres per line) ÷ (metres per sample).

**3.115 scalable hierarchy**: Coded video data consisting of an ordered set of more than one video bitstream.

**3.116 scalability**: Scalability is the ability of a decoder to decode an ordered set of bitstreams to produce a reconstructed sequence. Moreover, useful video is output when subsets are decoded. The minimum subset that can thus be decoded is the first bitstream in the set which is called the base layer. Each of the other bitstreams in the set is called an enhancement layer. When addressing a specific enhancement layer, "lower layer" refer to the bitstream which precedes the enhancement layer.

**3.117 side information**: Information in the bitstream necessary for controlling the decoder.

**3.118 16 × 8 prediction**: A prediction mode similar to field-based prediction but where the predicted block size is 16 × 8 luminance samples.

**3.119 run**: The number of zero coefficients preceding a non-zero coefficient, in the scan order. The absolute value of the non-zero coefficient is called "level".

**3.120 saturation**: Limiting a value that exceeds a defined range by setting its value to the maximum or minimum of the range as appropriate.

**3.121 skipped macroblock**: A macroblock for which no data is encoded.

**3.122 slice**: A consecutive series of macroblocks which are all located in the same horizontal row of macroblocks.

**3.123 SNR scalability**: A type of scalability where the enhancement layer(s) contain only coded refinement data for the DCT coefficients of the lower layer.

**3.124 source**; **input**: Term used to describe the video material or some of its attributes before encoding.

**3.125 spatial prediction**: Prediction derived from a decoded frame of the lower layer decoder used in spatial scalability.

**3.126 spatial scalability**: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer without using motion vectors. The layers can have different frame sizes, frame rates or chrominance formats.

**3.127 start codes (system and video)**: 32-bit codes embedded in that coded bitstream that are unique. They are used for several purposes including identifying some of the structures in the coding syntax.

**3.128 stuffing (bits)**; **stuffing (bytes)**: Code-words that may be inserted into the coded bitstream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate.

**3.129 temporal prediction**: Prediction derived from reference frames or fields other than those defined as spatial prediction.

**3.130 temporal scalability**: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer using motion vectors. The layers have identical frame size, and chrominance formats, but can have different frame rates.

**3.131 top field**: One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.

**3.132 top layer**: The topmost layer (with the highest layer_id) of a scalable hierarchy.

**3.133 variable bitrate**: Operation where the bitrate varies with time during the decoding of a coded bitstream.

**3.134 variable length coding (VLC)**: A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.

**3.135 video buffering verifier (VBV)**: A hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

**3.136 video sequence**: The highest syntactic structure of coded video bitstreams. It contains a series of one or more coded frames.

**3.137 xxx profile decoder**: Decoder able to decode one or a scalable hierarchy of bitstreams of which the top layer conforms to the specifications of the xxx profile (with xxx being any of the defined Profile names).

**3.138 xxx profile scalable hierarchy**: Set of bitstreams of which the top layer conforms to the specifications of the xxx profile.

**3.139 xxx profile bitstream**: A bitstream of a scalable hierarchy with a profile indication corresponding to xxx. Note that this bitstream is only decodable together with all its lower layer bitstreams (unless it is a base layer bitstream).

**3.140 zigzag scanning order**: A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.

## 4 Abbreviations and symbols

The mathematical operators used to describe this Specification are similar to those used in the C programming language. However, integer divisions with truncation and rounding are specifically defined. Numbering and counting loops generally begin from zero.

### 4.1 Arithmetic operators

| | |
|---|---|
| + | Addition |
| – | Subtraction (as a binary operator) or negation (as a unary operator) |
| ++ | Increment, i.e. *x*++ is equivalent to x = x + 1 |
| –– | Decrement, i.e. *x*–– is equivalent to x = x – 1 |
| × <br> * | Multiplication |

^ Power

/ Integer division with truncation of the result toward zero. For example, 7/4 and –7/–4 are truncated to 1 and –7/4 and 7/–4 are truncated to –1.

// Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and –3//2 is rounded to –2.

DIV Integer division with truncation of the result toward minus infinity. For example 3 DIV 2 is rounded to 1, and –3 DIV 2 is rounded to –2.

÷ Used to denote division in mathematical equations where no truncation or rounding is intended.

% Modulus operator. Defined only for positive numbers.

Sign( ) $\text{Sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x == 0 \\ -1 & x < 0 \end{cases}$

Abs( ) $\text{Abs}(x) = \begin{cases} x & x >= 0 \\ -x & x < 0 \end{cases}$

$\sum_{i=a}^{i<b} f(i)$ The summation of the $f(i)$ with $i$ taking integral values from $a$ up to, but not including $b$.

## 4.2 Logical operators

|| Logical OR

&& Logical AND

! Logical NOT

## 4.3 Relational operators

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

== Equal to

!= Not equal to

max [, … ,] The maximum value in the argument list

min [, … ,] The minimum value in the argument list

## 4.4 Bitwise operators

& AND

| OR

>> Shift right with sign extension

<< Shift left with zero fill

## 4.5 Assignment

= Assignment operator

## 4.6 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bitstream.

**bslbf** Bit string, left bit first, where "left" is the order in which bit strings are written in this Specification. Bit strings are generally written as a string of 1s and 0s within single quote marks, e.g. '1000 0001'. Blanks within a bit string are for ease of reading and have no significance. For convenience, large strings are occasionally written in hexadecimal; in this case, conversion to a binary in the conventional manner will yield the value of the bit string. Thus, the left most hexadecimal digit is first and in each hexadecimal digit the most significant of the four bits is first.

**uimsbf** Unsigned integer, most significant bit first.

**simsbf** Signed integer, in twos complement format, most significant (sign) bit first.

**vlclbf** Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written. The byte order of multibyte words is the most significant byte first.

In the clauses describing the syntax, any syntactic element that can only take positive or unsigned values (such as a flag that can be equal to 0 or 1) is described with the mnemonic 'uimsbf'. If the syntactic element can have a negative value, it is described with the mnemonic 'simsbf'. If the syntactic element has a constant value (e.g. marker_bit) then it is described with the mnemonic 'bslbf'. If the syntactic element represents a variable-length code, it is described with the mnemonic 'vlclbf'.

### 4.7 Constants

$\pi$ 3,141 592 653 58...

$e$ 2,718 281 828 45...

## 5 Conventions

### 5.1 Method of describing bitstream syntax

The bitstream retrieved by the decoder is described in 6.2. Each data item in the bitstream is in bold type. It is described by its name, its length in bits, and a mnemonic for its type and order of transmission.

The action caused by a decoded data element in a bitstream depends on the value of that data element and on data elements previously decoded. The decoding of the data elements and definition of the state variables used in their decoding are described in 6.3. The following constructs are used to express the conditions when data elements are present, and are in normal type:

| | |
|---|---|
| while ( condition ) {<br>  **data_element**<br>  . . .<br>} | If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true. |
| do {<br>  **data_element**<br>  . . .<br>} while ( condition ) | The data element always occurs at least once.<br><br>The data element is repeated until the condition is not true. |
| if ( condition ) {<br>  **data_element**<br>  . . .<br>} else {<br>  **data_element**<br>  . . .<br>} | If the condition is true, then the first group of data elements occurs next in the data stream.<br><br>If the condition is not true, then the second group of data elements occurs next in the data stream. |
| for ( i = m; i < n; i++) {<br>  **data_element**<br>  . . .<br>} | The group of data elements occurs (m – n) times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to zero for the first occurrence, incremented by one for the second occurrence, and so forth. |
| /* comment ... */ | Explanatory comment that may be deleted entirely without in any way altering the syntax. |

This syntax uses the 'C-code' convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true and a variable or expression evaluating to a zero value is equivalent to a condition that is false. In many cases a literal string is used in a condition. For example:

if ( scalable_mode == "spatial scalability" ) …

In such cases the literal string is that used to describe the value of the bitstream element in 6.3. In this example, we see that "spatial scalability" is defined in Table 6-10 to be represented by the two bit binary number '01'.

As noted, the group of data elements may contain nested conditional constructs. For compactness, the {} are omitted when only one data element follows:

**data_element [n]** data_element [n] is the n + 1th element of an array of data.

**data_element [m][n]** data_element [m][n] is the m + 1, n + 1th element of a two-dimensional array of data.

**data_element [l][m][n]** data_element [l][m][n] is the l + 1, m + 1, n + 1th element of a three-dimensional array of data.

While the syntax is expressed in procedural terms, it should not be assumed that 6.2 implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bitstream. Actual decoders must include means to look for start codes in order to begin decoding correctly, and to identify errors, erasures or insertions while decoding. The methods to identify these situations, and the actions to be taken, are not standardised.

## 5.2 Definition of functions

Several utility functions for picture coding algorithm are defined as follows.

### 5.2.1 Definition of bytealigned() function

The function bytealigned () returns 1 if the current position is on a byte boundary, that is the next bit in the bitstream is the first bit in a byte. Otherwise it returns 0.

### 5.2.2 Definition of nextbits() function

The function nextbits () permits comparison of a bit string with the next bits to be decoded in the bitstream.

### 5.2.3 Definition of next_start_code() function

The next_start_code() function removes any zero bit and zero byte stuffing and locates the next start code.

| next_start_code() { | No. of bits | Mnemonic |
|---|---|---|
| while ( !bytealigned() ) | | |
| **zero_bit** | 1 | '0' |
| while ( nextbits() != '0000 0000 0000 0000 0000 0001' ) | | |
| **zero_byte** | 8 | '0000 0000' |
| } | | |

This function checks whether the current position is byte aligned. If it is not, zero stuffing bits are present. After that any number of zero stuffing bytes may be present before the start code. Therefore, start codes are always byte aligned and may be preceded by any number of zero stuffing bits.

## 5.3 Reserved, forbidden and marker_bit

The terms "reserved" and "forbidden" are used in the description of some values of several fields in the coded bitstream.

The term "reserved" indicates that the value may be used in the future for ITU-T | ISO/IEC defined extensions.

The term "forbidden" indicates a value that shall never be used (usually in order to avoid emulation of start codes).

The term "marker_bit" indicates a one bit integer in which the value zero is forbidden (and it therefore shall have the value '1'). These marker bits are introduced at several points in the syntax to avoid start code emulation.

### 5.4 Arithmetic precision

In order to reduce discrepancies between implementations of this Specification, the following rules for arithmetic operations are specified:

a) Where arithmetic precision is not specified, such as in the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values.

b) Where ranges of values are given by a colon, the end points are included if a bracket is present, and excluded if the 'less than' (<) and 'greater than' (>) characters are used. For example, [a : b > means from *a* to *b*, including *a* but excluding *b*.

## 6 Video bitstream syntax and semantics

### 6.1 Structure of coded video data

Coded video data consists of an ordered set of video bitstreams, called layers. If there is only one layer, the coded video data is called non-scalable video bitstream. If there are two layers or more, the coded video data is called a scalable hierarchy.

The first layer (of the ordered set) is called base layer, and it can always be decoded independently. See 7.1 to 7.6 and 7.12 for a description of the decoding process for the base layer, except in the case of Data partitioning, described in 7.10.

Other layers are called enhancement layers, and can only be decoded together with all the lower layers (previous layers in the ordered set), starting with the base layer. See 7.7 to 7.11 for a description of the decoding process for scalable hierarchy.

See Rec. ITU-T H.222.0 | ISO/IEC 13818-1 for a description of the way layers may be multiplexed together.

The base layer of a scalable hierarchy may conform to this Specification or to other standards such as ISO/IEC 11172-2. See details in 7.7 to 7.11. Enhancement layers shall conform to this Specification.

In all cases apart from Data partitioning, the base layer does not contain a sequence_scalable_extension(). Enhancement layers always contain sequence_scalable_extension().

In general, the video bitstream can be thought of as a syntactic hierarchy in which syntactic structures contain one or more subordinate structures. For instance, the structure "picture_data()" contains one or more of the syntactic structure "slice()" which in turn contains one or more of the structure "macroblock()".

This structure is very similar to that used in ISO/IEC 11172-2.

#### 6.1.1 Video sequence

The highest syntactic structure of the coded video bitstream is the video sequence.

A video sequence commences with a sequence header which may optionally be followed by a group of pictures header and then by one or more coded frames. The order of the coded frames in the coded bitstream is the order in which the decoder processes them, but not necessarily in the correct order for display. The video sequence is terminated by a sequence_end_code. At various points in the video sequence, a particular coded frame may be preceded by either a repeat sequence header or a group of pictures header or both. (In the case that both a repeat sequence header and a group of pictures header immediately precede a particular picture, the group of pictures header shall follow the repeat sequence header.)

##### 6.1.1.1 Progressive and interlaced sequences

This Specification deals with coding of both progressive and interlaced sequences.

The output of the decoding process, for interlaced sequences, consists of a series of reconstructed fields that are separated in time by a field period. The two fields of a frame may be coded separately (field-pictures). Alternatively the two fields may be coded together as a frame (frame-pictures). Both frame pictures and field pictures may be used in a single video sequence.

In progressive sequences each picture in the sequence shall be a frame picture. The sequence, at the output of the decoding process, consists of a series of reconstructed frames that are separated in time by a frame period.

#### 6.1.1.2 Frame

A frame consists of three rectangular matrices of integers: a luminance matrix (Y), and two chrominance matrices (Cb and Cr).

The relationship between these Y, Cb and Cr components and the primary (analogue) Red, Green and Blue Signals ($E'_R$, $E'_G$ and $E'_B$), the chromaticity of these primaries and the transfer characteristics of the source frame may be specified in the bitstream (or specified by some other means). This information does not affect the decoding process.

#### 6.1.1.3 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

#### 6.1.1.4 Picture

A coded picture is made of a picture header, the optional extensions immediately following it and the following picture data. A coded picture may be a coded frame or a coded field.

An I-frame picture or a pair of field pictures, where the first field picture is an I-picture and the second field picture is an I-picture or a P-picture, is called a coded I-frame.

A P-frame picture or a pair of P-field pictures is called a coded P-frame.

A B-frame picture or a pair of B-field pictures is called a coded B-frame.

A coded I-frame, a coded P-frame or a coded B-frame is called a coded frame.

A reconstructed picture is obtained by decoding a coded picture, i.e. a picture header, the optional extensions immediately following it, and the picture data. A coded picture may be a frame picture or a field picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a reconstructed frame (when decoding a field picture).

##### 6.1.1.4.1 Field pictures

If field pictures are used, then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field pictures that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first picture of the coded frame is a P-field picture, then the second picture of the coded frame shall also be a P-field picture. Similarly when the first picture of the coded frame is a B-field picture the second picture of the coded frame shall also be a B-field picture.

When the first picture of the coded frame is a I-field picture, then the second picture of the frame shall be either an I-field picture or a P-field picture. If the second picture is a P-field picture, then certain restrictions apply (see 7.6.3.5).

##### 6.1.1.4.2 Frame pictures

When coding interlaced sequences using frame pictures, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-picture.

#### 6.1.1.5 Picture types

There are three types of pictures that use different coding methods:

- an **Intra-coded (I) picture** is coded using information only from itself;
- a **Predictive-coded (P) picture** is a picture which is coded using motion compensated prediction from a past reference frame or past reference field;
- a **Bidirectionally predictive-coded (B) picture** is a picture which is coded using motion compensated prediction from a past and/or future reference frame(s).

#### 6.1.1.6 Sequence header

A video sequence header commences with a sequence_header_code and is followed by a series of data elements. In this Specification, sequence_header() shall be followed by sequence_extension() which includes further parameters beyond those used by ISO/IEC 11172-2. When sequence_extension() is present, the syntax and semantics defined in ISO/IEC 11172-2 do not apply, and the present Specification applies.

In repeat sequence headers all of the data elements with the permitted exception of those defining the quantisation matrices (load_intra_quantiser_matrix, load_non_intra_quantiser_matrix and optionally intra_quantiser_matrix and non_intra_quantiser_matrix) shall have the same values as in the first sequence header. The quantisation matrices may be redefined each time that a sequence header occurs in the bitstream (note that quantisation matrices may also be updated using quant_matrix_extension()).

All of the data elements in the sequence_extension() that follows a repeat sequence_header() shall have the same values as in the first sequence_extension().

If a sequence_scalable_extension() occurs after the first sequence_header() all subsequent sequence headers shall be followed by sequence_scalable_extension() in which all data elements are the same as in the first sequence_scalable_extension(). Conversely if no sequence_scalable_extension() occurs between the first sequence_header() and the first picture_header(), then sequence_scalable_extension() shall not occur in the bitstream.

If a sequence_display_extension() occurs after the first sequence_header() all subsequent sequence headers shall be followed by sequence_display_extension() in which all data elements are the same as in the first sequence_display_extension(). Conversely if no sequence_display_extension() occurs between the first sequence_header() and the first picture_header(), then sequence_display_extension() shall not occur in the bitstream.

Repeating the sequence header allows the data elements of the initial sequence header to be repeated in order that random access into the video sequence is possible.

In the coded bitstream, the first picture following a sequence header or a repeated sequence header shall be either an I-picture or a P-picture, but not a B-picture. In the case that an interlaced frame is coded as two separate field pictures, a repeat sequence header shall not precede the second of these two field pictures.

If a bitstream is edited so that all of the data preceding any of the repeat sequence headers is removed (or alternatively random access is made to that sequence header), then the resulting bitstream shall be a legal bitstream that complies with this Specification. In the case that the first picture of the resulting bitstream is a P-picture, it is possible that it will contain non-intra macroblocks. Since the reference picture(s) required by the decoding process are not available, the reconstructed picture may not be fully defined. The time taken to fully refresh the entire frame depends on the refresh techniques employed.

#### 6.1.1.7 I-pictures and group of pictures header

I-pictures are intended to assist random access into the sequence. Applications requiring random access, fast-forward playback, or fast reverse playback may use I-pictures relatively frequently.

I-pictures may also be used at scene cuts or other cases where motion compensation is ineffective.

Group of picture header is an optional header that can be used immediately before a coded I-frame to indicate to the decoder if the first consecutive B-pictures immediately following the coded I-frame can be reconstructed properly in the case of a random access. In effect, if the preceding reference frame is not available, those B-pictures, if any, cannot be reconstructed properly unless they only use backward prediction or intra coding. This is more precisely defined in the clause describing closed_gop and broken_link. A group of picture header also contains a time code information that is not used by the decoding process.

In the coded bitstream, the first coded frame following a group of pictures header shall be a coded I-frame.

#### 6.1.1.8 4:2:0 format

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in both horizontal and vertical dimensions. The Y-matrix shall have an even number of lines and samples.

> NOTE – When interlaced frames are coded as field pictures, the picture reconstructed from each of these field pictures shall have a Y-matrix with half the number of lines as the corresponding frame. Thus, the total number of lines in the Y-matrix of an entire frame shall be divisible by four.

The luminance and chrominance samples are positioned as shown in Figure 6-1.

In order to further specify the organisation, Figures 6-2 and 6-3 show the vertical and temporal positioning of the samples in an interlaced frame. Figure 6-4 shows the vertical and temporal positioning of the samples in a progressive frame.

In each field of an interlaced frame, the chrominance samples do not lie (vertically) midway between the luminance samples of the field, this is so that the spatial location of the chrominance samples in the frame is the same whether the frame is represented as a single frame-picture or two field-pictures.

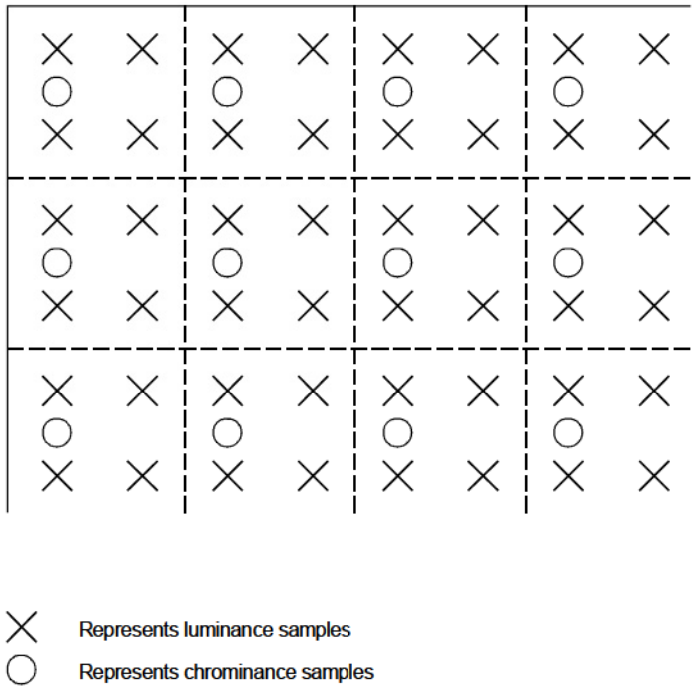



**Figure 6-1 – The position of luminance and chrominance samples – 4:2:0 data**

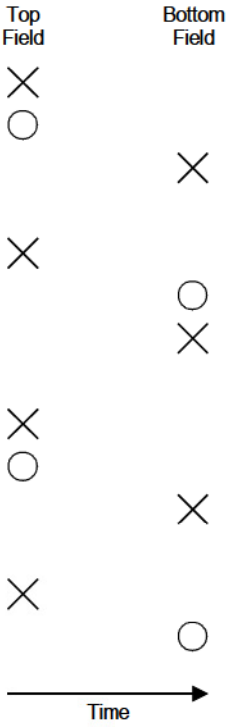



**Figure 6-2 – Vertical and temporal positions of samples in an interlaced frame with top_field_first = 1**

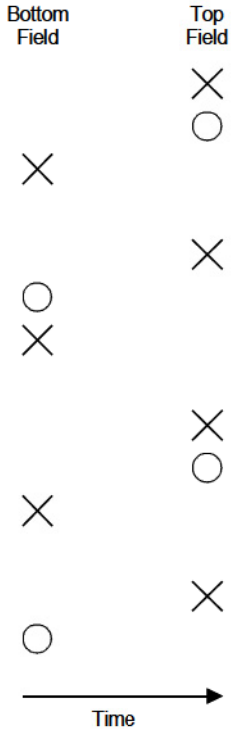



**Figure 6-3 – Vertical and temporal positions of samples in an interlaced frame with top_field_first = 0**

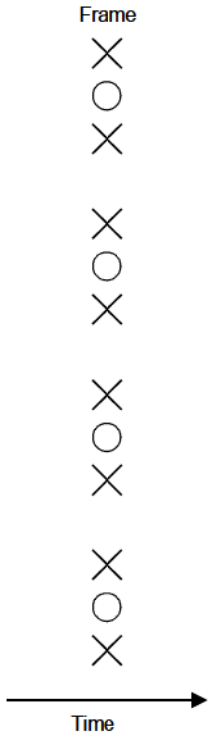



**Figure 6-4 – Vertical and temporal positions of samples in a progressive frame**

#### 6.1.1.9 4:2:2 format

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in the horizontal dimension and the same size as the Y-matrix in the vertical dimension. The Y-matrix shall have an even number of samples.

NOTE – When interlaced frames are coded as field pictures, the picture reconstructed from each of these field pictures shall have a Y-matrix with half the number of lines as the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figure 6-5.

In order to clarify the organisation, Figure 6-6 shows the (vertical) positioning of the samples when the frame is separated into two fields.

#### 6.1.1.10 4:4:4 format

In this format the Cb and Cr matrices shall be the same size as the Y-matrix in the horizontal and the vertical dimensions.

NOTE – When interlaced frames are coded as field pictures, the picture reconstructed from each of these field pictures shall have a Y-matrix with half the number of lines as the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figures 6-6 and 6-7.

#### 6.1.1.11 Frame re-ordering

When the sequence contains coded B-frames, the number of consecutive coded B-frames is variable and unbounded. The first coded frame after a sequence header shall not be a B-frame.

A sequence may contain no coded P-frames. A sequence may also contain no coded I-frames in which case some care is required at the start of the sequence and within the sequence to effect both random access and error recovery.

The order of the coded frames in the bitstream, also called coded order, is the order in which a decoder reconstructs them. The order of the reconstructed frames at the output of the decoding process, also called the display order, is not always the same as the coded order and this subclause defines the rules of frame re-ordering that shall happen within the decoding process.

When the sequence contains no coded B-frames, the coded order is the same as the display order. This is true in particular always when low_delay is one.

When B-frames are present in the sequence, re-ordering is performed according to the following rules:

- If the current frame in coded order is a B-frame, the output frame is the frame reconstructed from that B-frame.
- If the current frame in coded order is a I-frame or P-frame, the output frame is the frame reconstructed from the previous I-frame or P-frame if one exists. If none exists, at the start of the sequence, no frame is output.

The frame reconstructed from the final I-frame or P-frame is output immediately after the frame reconstructed when the last coded frame in the sequence was removed from the VBV buffer.

The following is an example of frames taken from the beginning of a video sequence. In this example there are two coded B-frames between successive coded P-frames and also two coded B-frames between successive coded I- and P-frames and all pictures are frame-pictures. Frame '1I' is used to form a prediction for frame '4P'. Frames '4P' and '1I' are both used to form predictions for frames '2B' and '3B'. Therefore the order of coded frames in the coded sequence shall be '1I', '4P', '2B', '3B'. However, the decoder shall display them in the order '1I', '2B', '3B', '4P'.

At the encoder input:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | B | B | P | B | B | P | B | B | I | B | B | P |

At the encoder output, in the coded bitstream, and at the decoder input:

| 1 | 4 | 2 | 3 | 7 | 5 | 6 | 10 | 8 | 9 | 13 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | P | B | B | P | B | B | I | B | B | P | B | B |

At the decoder output:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

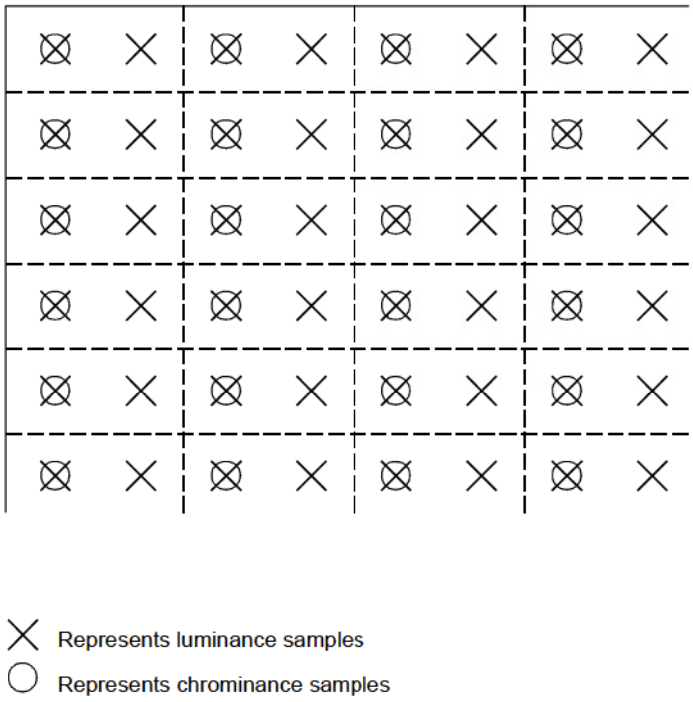



**Figure 6-5 – The position of luminance and chrominance samples – 4:2:2 data**




**Figure 6-6 – Vertical positions of samples with 4:2:2 and 4:4:4 data**

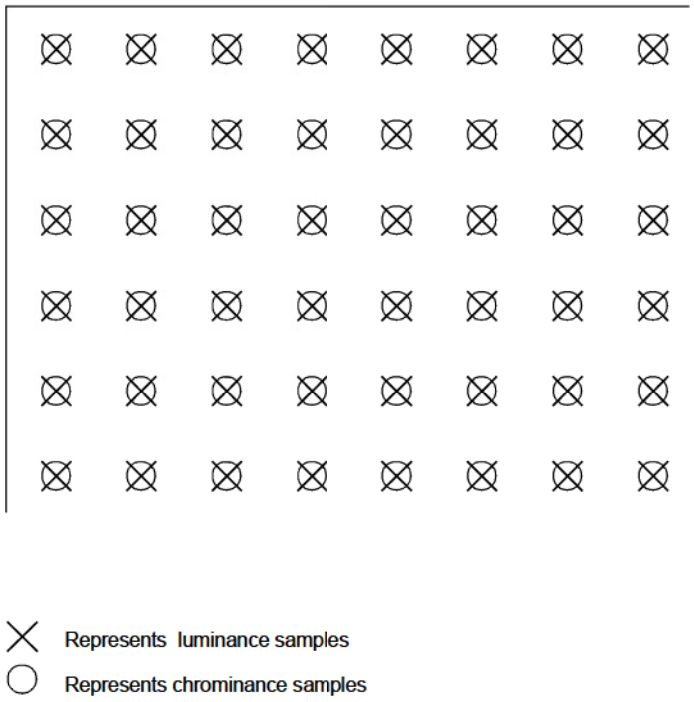



**Figure 6-7 – The position of luminance and chrominance samples – 4:4:4 data**

### 6.1.2 Slice

A **slice** is a series of an arbitrary number of consecutive macroblocks. The first and last macroblocks of a slice shall not be skipped macroblocks. Every slice shall contain at least one macroblock. Slices shall not overlap. The position of slices may change from picture to picture.

The first and last macroblock of a slice shall be in the same horizontal row of macroblocks.

Slices shall occur in the bitstream in the order in which they are encountered, starting at the upper-left of the picture and proceeding by raster-scan order from left to right and top to bottom (illustrated in Figures 6-8 and 6-9 as alphabetical order).

#### 6.1.2.1 The general slice structure

In the most general case it is not necessary for the slices to cover the entire picture. Figure 6-8 shows this case. Those areas that are not enclosed in a slice are not encoded and no information is encoded for such areas (in the specific picture).

If the slices do not cover the entire picture, then it is a requirement that if the picture is subsequently used to form predictions, then predictions shall only be made from those regions of the picture that were enclosed in slices. It is the responsibility of the encoder to ensure this.

This Specification does not define what action a decoder shall take in the regions between the slices.

#### 6.1.2.2 Restricted slice structure

In certain defined levels of defined profiles a restricted slice structure illustrated in Figure 6-9 shall be used. In this case every macroblock in the picture shall be enclosed in a slice.

Where a defined level of a defined profile requires that the slice structure obeys the restrictions detailed in this subclause, the term "restricted slice structure" may be used.

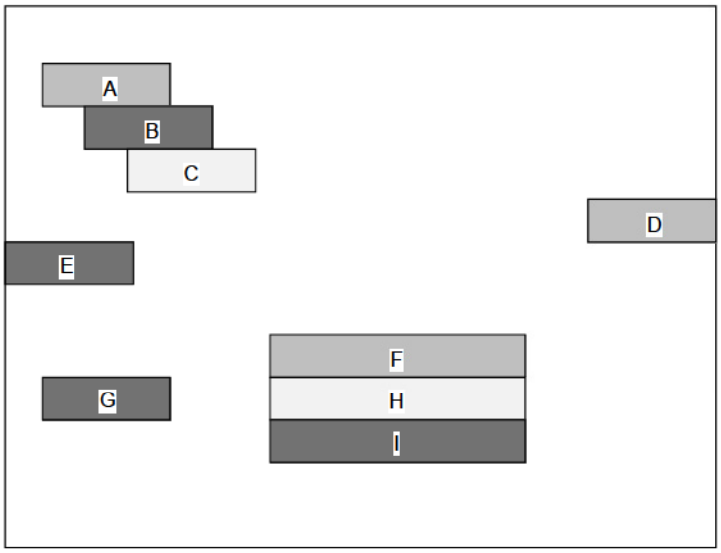



**Figure 6-8 – The most general slice structure**

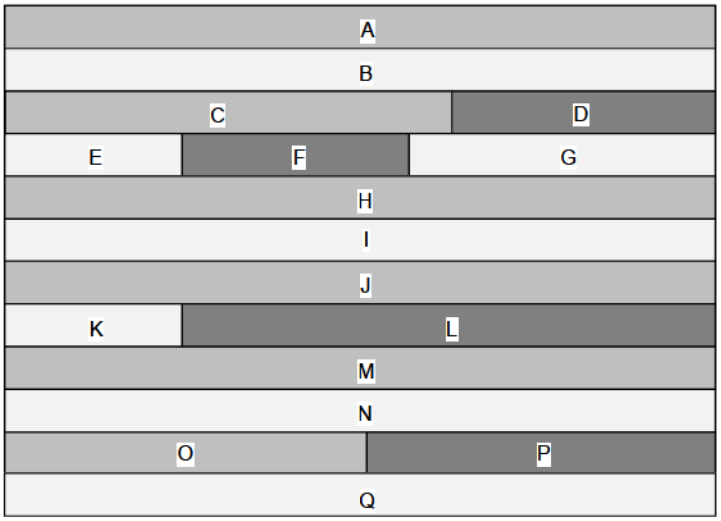



**Figure 6-9 – Restricted slice structure**

### 6.1.3 Macroblock

A **macroblock** contains a section of the luminance component and the spatially corresponding chrominance components. The term macroblock can either refer to source and decoded data or to the corresponding coded data elements. A skipped macroblock is one for which no information is transmitted (see 7.6.6). There are three chrominance formats for a macroblock, namely, 4:2:0, 4:2:2 and 4:4:4 formats. The orders of blocks in a macroblock shall be different for each different chrominance format and are illustrated below.

A 4:2:0 Macroblock consists of 6 blocks. This structure holds 4 Y, 1 Cb and 1 Cr Blocks and the block order is depicted in Figure 6-10.

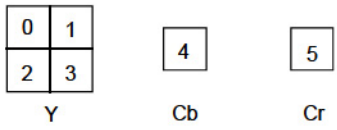



**Figure 6-10 – 4:2:0 Macroblock structure**

A 4:2:2 Macroblock consists of 8 blocks. This structure holds 4 Y, 2 Cb and 2 Cr Blocks and the block order is depicted in Figure 6-11.

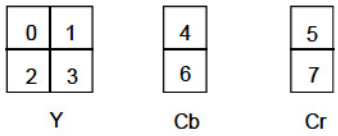



**Figure 6-11 – 4:2:2 Macroblock structure**

A 4:4:4 Macroblock consists of 12 blocks. This structure holds 4 Y, 4 Cb and 4 Cr Blocks and the block order is depicted in Figure 6-12.

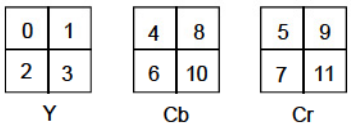



**Figure 6-12 – 4:4:4 Macroblock structure**

In frame pictures, where both frame and field DCT coding may be used, the internal organisation within the macroblock is different in each case.

- In the case of frame DCT coding, each block shall be composed of lines from the two fields alternately. This is illustrated in Figure 6-13.
- In the case of field DCT coding, each block shall be composed of lines from only one of the two fields. This is illustrated in Figure 6-14.

In the case of chrominance blocks the structure depends upon the chrominance format that is being used. In the case of 4:2:2 and 4:4:4 formats (where there are two blocks in the vertical dimension of the macroblock) the chrominance blocks are treated in exactly the same manner as the luminance blocks. However, in the 4:2:0 format the chrominance blocks shall always be organised in frame structure for the purposes of DCT coding. It should, however, be noted that field based predictions may be made for these blocks which will, in the general case, require that predictions for $8 \times 4$ regions (after half-sample filtering) must be made.

In field pictures, each picture only contains lines from one of the fields. In this case each block consists of lines taken from successive lines in the picture as illustrated by Figure 6-13.

### 6.1.4 Block

The term "**block**" can refer either to source and reconstructed data or to the DCT coefficients or to the corresponding coded data elements.

When "block" refers to source and reconstructed data it refers to an orthogonal section of a luminance or chrominance component with the same number of lines and samples. There are 8 lines and 8 samples in the block.

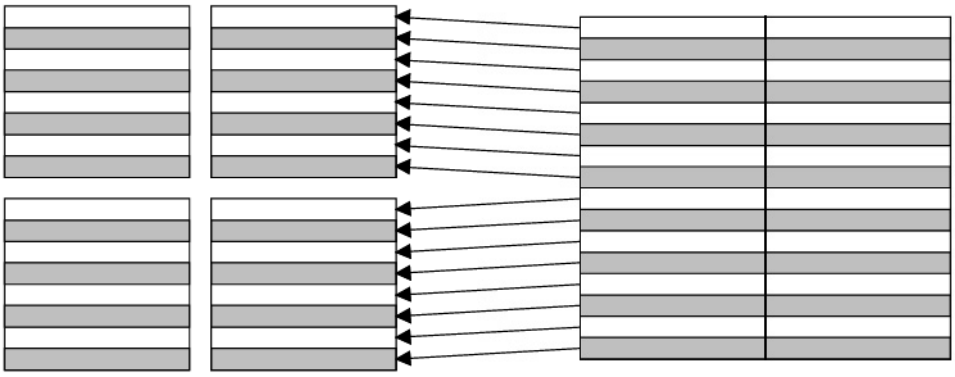


**Figure 6-13 – Luminance macroblock structure in frame DCT coding**

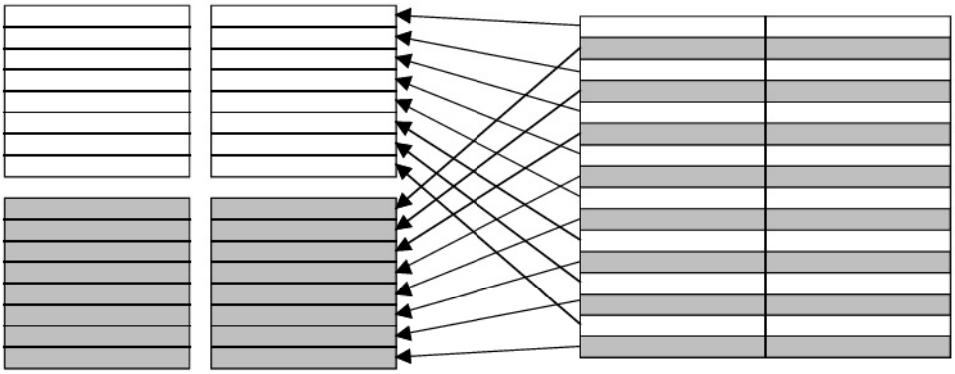


**Figure 6-14 – Luminance macroblock structure in field DCT coding**

## 6.2 Video bitstream syntax

### 6.2.1 Start codes

Start codes are specific bit patterns that do not otherwise occur in the video stream.

Each start code consists of a start code prefix followed by a start code value. The start code prefix is a string of twenty three bits with the value zero followed by a single bit with the value one. The start code prefix is thus the bit string '0000 0000 0000 0000 0000 0001'.

The start code value is an eight bit integer which identifies the type of start code. Most types of start code have just one start code value. However, slice_start_code is represented by many start code values, in this case the start code value is the slice_vertical_position for the slice.

All start codes shall be byte aligned. This shall be achieved by inserting bits with the value zero before the start code prefix such that the first bit of the start code prefix is the first (most significant) bit of a byte.

Table 6-1 defines the start code values for the start codes used in the video bitstream.

The use of the start codes is defined in the following syntax description with the exception of the sequence_error_code. The sequence_error_code has been allocated for use by a media interface to indicate where uncorrectable errors have been detected.

**Table 6-1 – Start code values**

| Name | Start code value (hexadecimal) |
|---|---|
| picture_start_code | 00 |
| slice_start_code | 01 through AF |
| reserved | B0 |
| reserved | B1 |
| user_data_start_code | B2 |
| sequence_header_code | B3 |
| sequence_error_code | B4 |
| extension_start_code | B5 |
| reserved | B6 |
| sequence_end_code | B7 |
| group_start_code | B8 |
| system start codes (Note) | B9 through FF |
| NOTE – System start codes are defined in Part 1 of this Specification. | |

### 6.2.2 Video Sequence

| video_sequence() { | No. of bits | Mnemonic |
|---|---|---|
| next_start_code() | | |
| sequence_header() | | |
| if ( nextbits() == extension_start_code ) { | | |
| sequence_extension() | | |
| do { | | |
| extension_and_user_data( 0 ) | | |
| do { | | |
| if (nextbits() == group_start_code) { | | |
| group_of_pictures_header() | | |
| extension_and_user_data( 1 ) | | |
| } | | |
| picture_header() | | |
| picture_coding_extension() | | |
| extensions_and_user_data( 2 ) | | |
| picture_data() | | |
| } while ( (nextbits() == picture_start_code) \|\| | | |
| (nextbits() == group_start_code) ) | | |
| if ( nextbits() != sequence_end_code ) { | | |
| sequence_header() | | |
| sequence_extension() | | |
| } | | |
| } while ( nextbits() != sequence_end_code ) | | |
| } else { | | |
| /* ISO/IEC 11172-2 */ | | |
| } | | |
| **sequence_end_code** | 32 | bslbf |
| } | | |

#### 6.2.2.1 Sequence header

| sequence_header() { | No. of bits | Mnemonic |
|---|---|---|
| **sequence_header_code** | 32 | bslbf |
| **horizontal_size_value** | 12 | uimsbf |
| **vertical_size_value** | 12 | uimsbf |
| **aspect_ratio_information** | 4 | uimsbf |
| **frame_rate_code** | 4 | uimsbf |
| **bit_rate_value** | 18 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **vbv_buffer_size_value** | 10 | uimsbf |
| **constrained_parameters_flag** | 1 | bslbf |
| **load_intra_quantiser_matrix** | 1 | uimsbf |
| if ( load_intra_quantiser_matrix ) | | |
| **intra_quantiser_matrix[64]** | 8*64 | uimsbf |
| **load_non_intra_quantiser_matrix** | 1 | uimsbf |
| if ( load_non_intra_quantiser_matrix ) | | |
| **non_intra_quantiser_matrix[64]** | 8*64 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.2.2 Extension and user data

| extension_and_user_data( i ) { | No. of bits | Mnemonic |
|---|---|---|
| while ( ( nextbits()== extension_start_code ) \|\| | | |
| ( nextbits()== user_data_start_code ) ) { | | |
| if ( ( i != 1) && ( nextbits()== extension_start_code ) ) | | |
| extension_data( i ) | | |
| if ( nextbits()== user_data_start_code ) | | |
| user_data() | | |
| } | | |
| } | | |

#### 6.2.2.2.1 Extension data

| extension_data( i ) { | No. of bits | Mnemonic |
|---|---|---|
| while ( nextbits()== extension_start_code ) { | | |
| **extension_start_code** | 32 | bslbf |
| if (i == 0) { /* follows sequence_extension() */ | | |
| if ( nextbits()== "Sequence Display Extension ID" ) | | |
| sequence_display_extension() | | |
| else if ( nextbits() | | |
| == "Sequence Scalable Extension ID" ) | | |
| sequence_scalable_extension() | | |
| } | | |
| /* NOTE – i never takes the value 1 because extension_data() | | |
| never follows a group_of_pictures_header() */ | | |
| if (i == 2) { /* follows picture_coding_extension() */ | | |
| if ( nextbits() == "Quant Matrix Extension ID" ) | | |
| quant_matrix_extension() | | |
| else if ( nextbits() == "Copyright Extension ID" ) | | |
| copyright_extension() | | |
| else if ( nextbits() == "Picture Display Extension ID" ) | | |
| picture_display_extension() | | |
| else if ( nextbits() | | |
| == "Picture Spatial Scalable Extension ID" ) | | |
| picture_spatial_scalable_extension() | | |
| else if ( nextbits() | | |
| == "Picture Temporal Scalable Extension ID" ) | | |
| picture_temporal_scalable_extension() | | |
| else if ( nextbits() | | |
| == "Camera Parameters Extension ID" ) | | |
| camera_parameters_extension() | | |
| else if ( nextbits() | | |
| == "ITU-T Extension ID" ) | | |
| ITU-T_extension() | | |
| } | | |
| } | | |
| } | | |

#### 6.2.2.2.2 User data

| user_data() { | No. of bits | Mnemonic |
|---|---|---|
| **user_data_start_code** | 32 | bslbf |
| while( nextbits() != '0000 0000 0000 0000 0000 0001' ) { | | |
| **user_data** | 8 | uimsbf |
| } | | |
| next_start_code() | | |
| } | | |

#### 6.2.2.3 Sequence extension

| sequence_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code** | 32 | bslbf |
| **extension_start_code_identifier** | 4 | uimsbf |
| **profile_and_level_indication** | 8 | uimsbf |
| **progressive_sequence** | 1 | uimsbf |
| **chroma_format** | 2 | uimsbf |
| **horizontal_size_extension** | 2 | uimsbf |
| **vertical_size_extension** | 2 | uimsbf |
| **bit_rate_extension** | 12 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **vbv_buffer_size_extension** | 8 | uimsbf |
| **low_delay** | 1 | uimsbf |
| **frame_rate_extension_n** | 2 | uimsbf |
| **frame_rate_extension_d** | 5 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.2.4 Sequence display extension

| sequence_display_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **video_format** | 3 | uimsbf |
| **colour_description** | 1 | uimsbf |
| if ( colour_description ) { | | |
| **colour_primaries** | 8 | uimsbf |
| **transfer_characteristics** | 8 | uimsbf |
| **matrix_coefficients** | 8 | uimsbf |
| } | | |
| **display_horizontal_size** | 14 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **display_vertical_size** | 14 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.2.5 Sequence scalable extension

| sequence_scalable_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **scalable_mode** | 2 | uimsbf |
| **layer_id** | 4 | uimsbf |
| if (scalable_mode == "spatial scalability") { | | |
| **lower_layer_prediction_horizontal_size** | 14 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **lower_layer_prediction_vertical_size** | 14 | uimsbf |
| **horizontal_subsampling_factor_m** | 5 | uimsbf |
| **horizontal_subsampling_factor_n** | 5 | uimsbf |
| **vertical_subsampling_factor_m** | 5 | uimsbf |
| **vertical_subsampling_factor_n** | 5 | uimsbf |
| } | | |
| if ( scalable_mode == "temporal scalability" ) { | | |
| **picture_mux_enable** | 1 | uimsbf |
| if ( picture_mux_enable ) | | |
| **mux_to_progressive_sequence** | 1 | uimsbf |
| **picture_mux_order** | 3 | uimsbf |
| **picture_mux_factor** | 3 | uimsbf |
| } | | |
| next_start_code() | | |
| } | | |

#### 6.2.2.6 Group of pictures header

| group_of_pictures_header() { | No. of bits | Mnemonic |
|---|---|---|
| **group_start_code** | 32 | bslbf |
| **time_code** | 25 | uimsbf |
| **closed_gop** | 1 | uimsbf |
| **broken_link** | 1 | uimsbf |
| next_start_code() | | |
| } | | |

### 6.2.3 Picture header

| picture_header() { | No. of bits | Mnemonic |
|---|---|---|
| **picture_start_code** | 32 | bslbf |
| **temporal_reference** | 10 | uimsbf |
| **picture_coding_type** | 3 | uimsbf |
| **vbv_delay** | 16 | uimsbf |
| if ( picture_coding_type == 2 \|\| picture_coding_type == 3) { | | |
| **full_pel_forward_vector** | 1 | bslbf |
| **forward_f_code** | 3 | bslbf |
| } | | |
| if ( picture_coding_type == 3 ) { | | |
| **full_pel_backward_vector** | 1 | bslbf |
| **backward_f_code** | 3 | bslbf |
| } | | |
| while ( nextbits() == '1' ) { | | |
| **extra_bit_picture** /* with the value '1' */ | 1 | uimsbf |
| **extra_information_picture** | 8 | uimsbf |
| } | | |
| **extra_bit_picture** /* with the value '0' */ | 1 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.3.1 Picture coding extension

| picture_coding_extension() { | No . of bits | Mnemonic |
|---|---|---|
| **extension_start_code** | 32 | bslbf |
| **extension_start_code_identifier** | 4 | uimsbf |
| **f_code[0][0] /* forward horizontal */** | 4 | uimsbf |
| **f_code[0][1] /* forward vertical */** | 4 | uimsbf |
| **f_code[1][0] /* backward horizontal */** | 4 | uimsbf |
| **f_code[1][1] /* backward vertical */** | 4 | uimsbf |
| **intra_dc_precision** | 2 | uimsbf |
| **picture_structure** | 2 | uimsbf |
| **top_field_first** | 1 | uimsbf |
| **frame_pred_frame_dct** | 1 | uimsbf |
| **concealment_motion_vectors** | 1 | uimsbf |
| **q_scale_type** | 1 | uimsbf |
| **intra_vlc_format** | 1 | uimsbf |
| **alternate_scan** | 1 | uimsbf |
| **repeat_first_field** | 1 | uimsbf |
| **chroma_420_type** | 1 | uimsbf |
| **progressive_frame** | 1 | uimsbf |
| **composite_display_flag** | 1 | uimsbf |
| if ( composite_display_flag ) { | | |
| **v_axis** | 1 | uimsbf |
| **field_sequence** | 3 | uimsbf |
| **sub_carrier** | 1 | uimsbf |
| **burst_amplitude** | 7 | uimsbf |
| **sub_carrier_phase** | 8 | uimsbf |
| } | | |
| next_start_code() | | |
| } | | |

#### 6.2.3.2 Quant matrix extension

| quant_matrix_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **load_intra_quantiser_matrix** | 1 | uimsbf |
| if ( load_intra_quantiser_matrix ) | | |
| **intra_quantiser_matrix[64]** | 8 * 64 | uimsbf |
| **load_non_intra_quantiser_matrix** | 1 | uimsbf |
| if ( load_non_intra_quantiser_matrix ) | | |
| **non_intra_quantiser_matrix[64]** | 8 * 64 | uimsbf |
| **load_chroma_intra_quantiser_matrix** | 1 | uimsbf |
| if ( load_chroma_intra_quantiser_matrix ) | | |
| **chroma_intra_quantiser_matrix[64]** | 8 * 64 | uimsbf |
| **load_chroma_non_intra_quantiser_matrix** | 1 | uimsbf |
| if ( load_chroma_non_intra_quantiser_matrix ) | | |
| **chroma_non_intra_quantiser_matrix[64]** | 8 * 64 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.3.3 Picture display extension

| picture_display_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| for ( i = 0; i < number_of_frame_centre_offsets; i ++ ) { | | |
| **frame_centre_horizontal_offset** | 16 | simsbf |
| **marker_bit** | 1 | bslbf |
| **frame_centre_vertical_offset** | 16 | simsbf |
| **marker_bit** | 1 | bslbf |
| } | | |
| next_start_code() | | |
| } | | |

#### 6.2.3.4 Picture temporal scalable extension

| picture_temporal_scalable_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **reference_select_code** | 2 | uimsbf |
| **forward_temporal_reference** | 10 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **backward_temporal_reference** | 10 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.3.5 Picture spatial scalable extension

| picture_spatial_scalable_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **lower_layer_temporal_reference** | 10 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **lower_layer_horizontal_offset** | 15 | simsbf |
| **marker_bit** | 1 | bslbf |
| **lower_layer_vertical_offset** | 15 | simsbf |
| **spatial_temporal_weight_code_table_index** | 2 | uimsbf |
| **lower_layer_progressive_frame** | 1 | uimsbf |
| **lower_layer_deinterlaced_field_select** | 1 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.3.6 Copyright extension

| copyright_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **copyright_flag** | 1 | uimsbf |
| **copyright_identifier** | 8 | uimsbf |
| **original_or_copy** | 1 | uimsbf |
| **reserved** | 7 | bslbf |
| **marker_bit** | 1 | bslbf |
| **copyright_number_1** | 20 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **copyright_number_2** | 22 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **copyright_number_3** | 22 | uimsbf |
| next_start_code() | | |
| } | | |

#### 6.2.3.7 Picture data

| picture_data() { | No. of bits | Mnemonic |
|---|---|---|
| do { | | |
| slice() | | |
| } while ( nextbits() == slice_start_code ) | | |
| next_start_code() | | |
| } | | |

#### 6.2.3.7.1 Camera parameters extension

| camera_parameters_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| **reserved** | 1 | uimsbf |
| **camera_id** | 7 | simsbf |
| **marker_bit** | 1 | bslbf |
| **height_of_image_device** | 22 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **focal_length** | 22 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **f_number** | 22 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **vertical_angle_of_view** | 22 | uimsbf |
| **marker_bit** | 1 | bslbf |
| **camera_position_x_upper** | 16 | simsbf |
| **marker_bit** | 1 | bslbf |
| **camera_position_x_lower** | 16 | |
| **marker_bit** | 1 | bslbf |
| **camera_position_y_upper** | 16 | simsbf |
| **marker_bit** | 1 | bslbf |
| **camera_position_y_lower** | 16 | |
| **marker_bit** | 1 | bslbf |
| **camera_position_z_upper** | 16 | simsbf |
| **marker_bit** | 1 | bslbf |
| **camera_position_z_lower** | 16 | |
| **marker_bit** | 1 | bslbf |
| **camera_direction_x** | 22 | simsbf |
| **marker_bit** | 1 | bslbf |
| **camera_direction_y** | 22 | simsbf |
| **marker_bit** | 1 | bslbf |
| **camera_direction_z** | 22 | simsbf |
| **marker_bit** | 1 | bslbf |
| **image_plane_vertical_x** | 22 | simsbf |
| **marker_bit** | 1 | bslbf |
| **image_plane_vertical_y** | 22 | simsbf |
| **marker_bit** | 1 | bslbf |
| **image_plane_vertical_z** | 22 | simsbf |
| **marker_bit** | 1 | bslbf |
| **reserved** | 32 | bslbf |
| **next_start_code()** | | |
| } | | |

NOTE – The parameters marked 'reserved' in the camera parameters extension shall be equal to zero. Other values are reserved for future use by ITU-T | ISO/IEC.

#### 6.2.3.7.2 ITU-T extension

| ITU-T_extension() { | No. of bits | Mnemonic |
|---|---|---|
| **extension_start_code_identifier** | 4 | uimsbf |
| while( nextbits() != '0000 0000 0000 0000 0000 0001' ) { | | |
| **ITU-T_data** | 1 | uimsbf |
| } | | |
| **next_start_code()** | | |
| } | | |
| NOTE – The construct with the while-statement prevents start code emulation. | | |

### 6.2.4 Slice

| slice() { | No. of bits | Mnemonic |
|---|---|---|
| **slice_start_code** | 32 | bslbf |
| if (vertical_size > 2800) | | |
| **slice_vertical_position_extension** | 3 | uimsbf |
| if (*<sequence_scalable_extension() is present in the bitstream>*) { | | |
| if (scalable_mode == 'data partitioning') | | |
| **priority_breakpoint** | 7 | uimsbf |
| } | | |
| **quantiser_scale_code** | 5 | uimsbf |
| if (nextbits() == '1') { | | |
| **slice_extension_flag** | 1 | bslbf |
| **intra_slice** | 1 | uimsbf |
| **slice_picture_id_enable** | 1 | uimsbf |
| **slice_picture_id** | 6 | uimbsf |
| while (nextbits() == '1') { | | |
| **extra_bit_slice** /* with the value '1' */ | 1 | uimsbf |
| extra_information_slice | 8 | uimsbf |
| } | | |
| } | | |
| **extra_bit_slice** /* with the value '0' */ | 1 | uimsbf |
| do { | | |
| macroblock() | | |
| } while (nextbits() != '000 0000 0000 0000 0000 0000') | | |
| next_start_code() | | |
| } | | |

### 6.2.5 Macroblock

| macroblock() { | No. of bits | Mnemonic |
|---|---|---|
| while ( nextbits() == '0000 0001 000' ) | | |
| **macroblock_escape** | 11 | bslbf |
| **macroblock_address_increment** | 1-11 | vlclbf |
| macroblock_modes() | | |
| if ( macroblock_quant ) | | |
| **quantiser_scale_code** | 5 | uimsbf |
| if ( macroblock_motion_forward \|\| | | |
| ( macroblock_intra && concealment_motion_vectors) ) | | |
| motion_vectors( 0 ) | | |
| if ( macroblock_motion_backward ) | | |
| motion_vectors( 1 ) | | |
| if ( macroblock_intra && concealment_motion_vectors) | | |
| **marker_bit** | 1 | bslbf |
| if ( macroblock_pattern ) | | |
| coded_block_pattern() | | |
| for ( i = 0; i < block_count; i + + ) { | | |
| block( i ) | | |
| } | | |
| } | | |

#### 6.2.5.1 Macroblock modes

| macroblock_modes() { | No. of bits | Mnemonic |
|---|---|---|
| **macroblock_type** | 1-9 | vlclbf |
| if ( ( spatial_temporal_weight_code_flag == 1 ) && | | |
| ( spatial_temporal_weight_code_table_index != '00') ) { | | |
| **spatial_temporal_weight_code** | 2 | uimsbf |
| } | | |
| if ( macroblock_motion_forward \|\| | | |
| macroblock_motion_backward ) { | | |
| if ( picture_structure == 'frame' ) { | | |
| if ( frame_pred_frame_dct == 0 ) | | |
| **frame_motion_type** | 2 | uimsbf |
| } else { | | |
| **field_motion_type** | 2 | uimsbf |
| } | | |
| } | | |
| if ( ( picture_structure == "Frame picture" ) && | | |
| ( frame_pred_frame_dct == 0 ) && | | |
| ( macroblock_intra \|\| macoblock_pattern) ){ | | |
| **dct_type** | 1 | uimsbf |
| } | | |
| } | | |

#### 6.2.5.2 Motion vectors

| motion_vectors ( s ) { | No. of bits | Mnemonic |
|---|---|---|
| if ( motion_vector_count == 1 ) { | | |
| if (( mv_format == field ) && ( dmv != 1)) | | |
| **motion_vertical_field_select[0][s]** | 1 | uimsbf |
| motion_vector( 0, s ) | | |
| } else { | | |
| **motion_vertical_field_select[0][s]** | 1 | uimsbf |
| motion_vector( 0, s ) | | |
| **motion_vertical_field_select[1][s]** | 1 | uimsbf |
| motion_vector(1, s ) | | |
| } | | |
| } | | |

#### 6.2.5.2.1 Motion vector

| motion_vector ( r, s ) { | No. of bits | Mnemonic |
|---|---|---|
| **motion_code[r][s][0]** | 1-11 | vlclbf |
| if ( ( f_code[s][0] != 1) && ( motion_code[r][s][0] != 0 ) ) | | |
| **motion_residual[r][s][0]** | 1-8 | uimsbf |
| if (dmv == 1) | | |
| **dmvector[0]** | 1-2 | vlclbf |
| **motion_code[r][s][1]** | 1-11 | vlclbf |
| if ( ( f_code[s][1] != 1) && ( motion_code[r][s][1] != 0 ) ) | | |
| **motion_residual[r][s][1]** | 1-8 | uimsbf |
| if (dmv == 1) | | |
| **dmvector[1]** | 1-2 | vlclbf |
| } | | |

#### 6.2.5.3 Coded block pattern

| coded_block_pattern () { | No. of bits | Mnemonic |
|---|---|---|
| **coded_block_pattern_420** | 3-9 | vlclbf |
| if ( chroma_format == 4:2:2 ) | | |
| **coded_block_pattern_1** | 2 | uimsbf |
| if ( chroma_format == 4:4:4 ) | | |
| **coded_block_pattern_2** | 6 | uimsbf |
| } | | |

### 6.2.6 Block

The detailed syntax for the terms "First DCT coefficient", "Subsequent DCT coefficient" and "End of Block" is fully described in 7.2.

This subclause does not adequately document the block layer syntax when data partitioning is used. See 7.10.

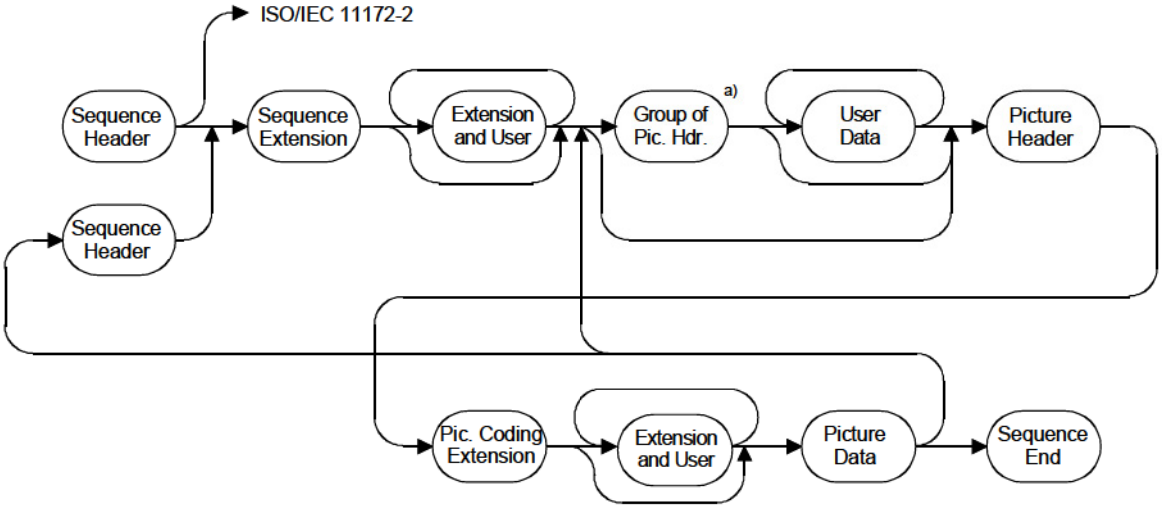
| block( i ) { | No. of bits | Mnemonic |
|---|---|---|
| if ( pattern_code[i] ) { | | |
| if ( macroblock_intra ) { | | |
| if ( i < 4 ) { | | |
| **dct_dc_size_luminance** | 2-9 | vlclbf |
| if(dct_dc_size_luminance != 0) | | |
| **dct_dc_differential** | 1-11 | uimsbf |
| } else { | | |
| **dct_dc_size_chrominance** | 2-10 | vlclbf |
| if(dct_dc_size_chrominance != 0) | | |
| **dct_dc_differential** | 1-11 | uimsbf |
| } | | |
| } else { | | |
| First DCT coefficient | 2-24 | vlclbf |
| } | | |
| while ( nextbits() != End of block ) | | |
| Subsequent DCT coefficients | 3-24 | vlclbf |
| End of block | 2 or 4 | vlclbf |
| } | | |
| } | | |

## 6.3 Video bitstream semantics

### 6.3.1 Semantic rules for higher syntactic structures

This subclause details the rules that govern the way in which the higher level syntactic elements may be combined together to produce a legal bitstream. Subsequent clauses detail the semantic meaning of all fields in the video bitstream.

Figure 6-15 illustrates the high level structure of the video bitstream.




a) After a GOP, the first picture shall be an I-picture.

**Figure 6-15 – High level bitstream organisation**

The following semantic rules apply:

- If the first sequence_header() of the sequence is not followed by sequence_extension(), then the stream shall conform to ISO/IEC 11172-2 and is not documented within this Specification.
- If the first sequence_header() of a sequence is followed by a sequence_extension(), then all subsequent occurrences of sequence_header() shall also be immediately followed by a sequence_extension().
- sequence_extension() shall only occur immediately following a sequence_header().
- Following a sequence_header() there shall be at least one coded picture before a repeat sequence_header() or a sequence_end_code. This implies that sequence_extension() shall not immediately precede a sequence_end_code.
- If sequence_extension() occurs in the bitstream, then each picture_header() shall be followed immediately by a picture_coding_extension().
- sequence_end_code shall be positioned at the end of the bitstream such that, after decoding and frame re-ordering, there shall be no missing frames.
- picture_coding_extension() shall only occur immediately following a picture_header().
- The first coded frame following a group_of_pictures_header() shall be a coded I-frame.

A number of different extensions are defined in addition to sequence_extension() and picture_coding_extension(). The set of allowed extensions is different at each different point in the syntax where extensions are allowed. Table 6-2 defines a four bit extension_start_code_identifier for each extension.

**Table 6-2 – extension_start_code_identifier codes**

| extension_start_code_identifier | Name |
|---|---|
| 0000 | Reserved |
| 0001 | Sequence Extension ID |
| 0010 | Sequence Display Extension ID |
| 0011 | Quant Matrix Extension ID |
| 0100 | Copyright Extension ID |
| 0101 | Sequence Scalable Extension ID |
| 0110 | Reserved |
| 0111 | Picture Display Extension ID |
| 1000 | Picture Coding Extension ID |
| 1001 | Picture Spatial Scalable Extension ID |
| 1010 | Picture Temporal Scalable Extension ID |
| 1011 | Camera Parameters Extension ID |
| 1100 | ITU-T extension ID |
| 1101 | Reserved |
| ... | ... |
| 1111 | Reserved |

At each point where extensions are allowed in the bitstream any number of the extensions from the defined allowable set may be included. However, each type of extension shall not occur more than once.

In the case that a decoder encounters an extension with an extension identification that is described as "reserved" in this Specification, the decoder shall discard all subsequent data until the next start code. This requirement allows future definition of compatible extensions to this Specification.

### 6.3.2 Video sequence

**sequence_end_code** – The sequence_end_code is the bit string '000001B7' in hexadecimal. It terminates a video sequence.

### 6.3.3 Sequence header

**sequence_header_code** – The sequence_header_code is the bit string '000001B3' in hexadecimal. It identifies the beginning of a sequence header.

**horizontal_size_value** – This word forms the 12 least significant bits of horizontal_size.

**vertical_size_value** – This word forms the 12 least significant bits of vertical_size.

**horizontal_size** – The horizontal_size is a 14-bit unsigned integer, the 12 least significant bits are defined in horizontal_size_value, the 2 most significant bits are defined in horizontal_size_extension. The horizontal_size is the width of the displayable part of the luminance component of pictures in samples. The width of the encoded luminance component of pictures in macroblocks, mb_width, is (horizontal_size + 15)/16. The displayable part is left-aligned in the encoded pictures.

In order to avoid start code emulation horizontal_size_value shall not be zero. This precludes values of horizontal_size that are multiples of 4096.

**vertical_size** – The vertical_size is a 14-bit unsigned integer, the 12 least significant bits are defined in vertical_size_value, the 2 most significant bits are defined in vertical_size_extension. The vertical_size is the height of the displayable part of the luminance component of the frame in lines.

In the case that progressive_sequence is '1' the height of the encoded luminance component of frames in macroblocks, mb_height, is (vertical_size + 15)/16.

In the case that progressive_sequence is '0' the height of the encoded luminance component of frame pictures in macroblocks, mb_height, is 2*((vertical_size + 31)/32). The height of the encoded luminance component of field pictures in macroblocks, mb_height, is ((vertical_size + 31)/32).

The displayable part is top-aligned in the encoded pictures.

In order to avoid start code emulation vertical_size_value shall not be zero. This precludes values of vertical_size that are multiples of 4096.

**aspect_ratio_information** – This is a four-bit integer defined in Table 6-3.

**Table 6-3 – aspect_ratio_information**

| aspect_ratio_information | Sample Aspect Ratio | DAR |
|---|---|---|
| 0000 | Forbidden | Forbidden |
| 0001 | 1.0 (Square Sample) | – |
| 0010 | – | 3 ÷ 4 |
| 0011 | – | 9 ÷ 16 |
| 0100 | – | 1 ÷ 2.21 |
| 0101 | – | Reserved |
| … | | … |
| 1111 | – | Reserved |

aspect_ratio_information either specifies that the "Sample Aspect Ratio" (SAR) of the reconstructed frame is 1,0 (square samples) or alternatively it gives the "Display Aspect Ratio" (DAR).

- If sequence_display_extension() is not present, then it is intended that the entire reconstructed frame is intended to be mapped to the entire active region of the display. The sample aspect ratio may be calculated as follows:

$$SAR = DAR \times \frac{horizontal\_size}{vertical\_size}$$

NOTE 1 – In this case horizontal_size and vertical_size are constrained by the SAR of the source and the DAR selected.

- If sequence_display_extension() is present then the sample aspect ratio may be calculated as follows:

$$SAR = DAR \times \frac{display\_horizontal\_size}{display\_vertical\_size}$$

**frame_rate_code** – This is a four-bit integer used to define frame_rate_value as shown in Table 6-4. frame_rate may be derived from frame_rate_value, frame_rate_extension_n and frame_rate_extension_d as follows:

$$frame\_rate = frame\_rate\_value * (frame\_rate\_extension\_n + 1) \div (frame\_rate\_extension\_d + 1)$$

When an entry for the frame rate exists directly in Table 6-4, frame_rate_extension_n and frame_rate_extension_d shall be zero. (frame_rate_extension_n + 1) and (frame_rate_extension_d + 1) shall not have a common divisor greater than one.

**Table 6-4 – frame_rate_value**

| frame_rate_code | frame_rate_value |
|---|---|
| 0000 | Forbidden |
| 0001 | 24 000 ÷ 1001 (23.976…) |
| 0010 | 24 |
| 0011 | 25 |
| 0100 | 30 000 ÷ 1001 (29.97…) |
| 0101 | 30 |
| 0110 | 50 |
| 0111 | 60 000 ÷ 1001 (59.94…) |
| 1000 | 60 |
| 1001 | Reserved |
| … | … |
| 1111 | Reserved |

If progressive_sequence is '1' the period between two successive frames at the output of the decoding process is the reciprocal of the frame_rate. See Figure 7-18.

If progressive_sequence is '0' the period between two successive fields at the output of the decoding process is half of the reciprocal of the frame_rate. See Figure 7-20.

The frame_rate signalled in the enhancement layer of temporal scalability is the combined frame rate after the temporal re-multiplex operation if picture_mux_enable in the sequence_scalable_extension() is set to '1'.

**bit_rate_value** – The lower 18 bits of bit_rate.

**bit_rate** – This is a 30-bit integer. The lower 18 bits of the integer are in bit_rate_value and the upper 12 bits are in bit_rate_extension. bit_rate is measured in units of 400 bits/second, rounded upwards. The value zero is forbidden.

The bitrate specified bounds the maximum rate of operation of the VBV as defined in C.3.

The VBV operates in one of two modes depending on the coded values in vbv_delay. In all cases (both constant and variable bitrate operation) the bitrate specified shall be the upper bound of the rate at which the coded data is supplied to the input of the VBV.

NOTE 2 – Since constant bitrate operation is simply a special case of variable bitrate operation there is no requirement that the value of bit_rate is the actual bitrate at which the data is supplied. However it is recommended in the case of constant bitrate operation that bit_rate should represent the actual bitrate.

**marker_bit** – This is one bit that shall be set to '1'. This bit prevents emulation of start codes.

**vbv_buffer_size_value** – the lower 10 bits of vbv_buffer_size.

**vbv_buffer_size** – vbv_buffer_size is an 18-bit integer. The lower 10 bits of the integer are in vbv_buffer_size_value and the upper 8 bits are in vbv_buffer_size_extension. The integer defines the size of the VBV (Video Buffering Verifier, see Annex C) buffer needed to decode the sequence. It is defined as:

$$B = 16 * 1024 * vbv\_buffer\_size$$

where B is the minimum VBV buffer size in bits required to decode the sequence (see Annex C).

**constrained_parameters_flag** – This flag (used in ISO/IEC 11172-2) has no meaning in this Specification and shall have the value '0'.

**load_intra_quantiser_matrix** – See 6.3.11 "Quant matrix extension".

**intra_quantiser_matrix** – See 6.3.11 "Quant matrix extension".

**load_non_intra_quantiser_matrix** – See 6.3.11 "Quant matrix extension".

**non_intra_quantiser_matrix** – See 6.3.11 "Quant matrix extension".

### 6.3.4 Extension and user data

**extension_start_code** – The extension_start_code is the bit string '000001B5' in hexadecimal. It identifies the beginning of extensions beyond ISO/IEC 11172-2.

#### 6.3.4.1 User data

**user_data_start_code** – The user_data_start_code is the bit string '000001B2' in hexadecimal. It identifies the beginning of user data. The user data continues until receipt of another start code.

**user_data** – This is an 8 bit integer, an arbitrary number of which may follow one another. User data is defined by users for their specific applications. In the series of consecutive user_data bytes there shall not be a string of 23 or more consecutive zero bits.

### 6.3.5 Sequence extension

**extension_start_code_identifier** – This is a 4-bit integer which identifies the extension. See Table 6-2.

**profile_and_level_indication** – This is an 8-bit integer used to signal the profile and level identification. The meaning of the bits is given in clause 8.

NOTE – In a scalable hierarchy the bitstreams of each layer may set profile_and_level_indication to a different value as specified in clause 8.

**progressive_sequence** – When set to '1' the coded video sequence contains only progressive frame-pictures. When progressive_sequence is set to '0' the coded video sequence may contain both frame-pictures and field-pictures, and frame-picture may be progressive or interlaced frames.

**chroma_format** – This is a two-bit integer indicating the chrominance format as defined in Table 6-5.

**horizontal_size_extension** – This 2-bit integer is the 2 most significant bits from horizontal_size.

**vertical_size_extension** – This 2-bit integer is the 2 most significant bits from vertical_size.

**bit_rate_extension** – This 12-bit integer is the 12 most significant bits from bit_rate.

**vbv_buffer_size_extension** – This 8-bit integer is the 8 most significant bits from vbv_buffer_size.

**Table 6-5 – Meaning of chroma_format**

| chroma_format | Meaning |
|---|---|
| 00 | Reserved |
| 01 | 4:2:0 |
| 10 | 4:2:2 |
| 11 | 4:4:4 |

**low_delay** – This flag, when set to '1', indicates that the sequence does not contain any B-pictures, that the frame re-ordering delay is not present in the VBV description and that the bitstream may contain "big pictures", i.e. that C.7 of the VBV may apply.

When set to '0', it indicates that the sequence may contain B-pictures, that the frame re-ordering delay is present in the VBV description and that bitstream shall not contain big pictures, i.e. C.7 of the VBV does not apply.

This flag is not used during the decoding process and therefore can be ignored by decoders, but it is necessary to define and verify the compliance of low-delay bitstreams.

**frame_rate_extension_n** – This is a 2-bit integer used to determine the frame_rate. See frame_rate_code.

**frame_rate_extension_d** – This is a 5-bit integer used to determine the frame_rate. See frame_rate_code.

### 6.3.6 Sequence display extension

This Specification does not define the display process. The information in this extension does not affect the decoding process and may be ignored by decoders that conform to this Specification.

**video_format** – This is a three bit integer indicating the representation of the pictures before being coded in accordance with this Specification. Its meaning is defined in Table 6-6. If the sequence_display_extension() is not present in the bitstream, then the video format may be assumed to be "Unspecified video format".

**Table 6-6 – Meaning of video_format**

| video_format | Meaning |
|---|---|
| 000 | Component |
| 001 | PAL |
| 010 | NTSC |
| 011 | SECAM |
| 100 | MAC |
| 101 | Unspecified video format |
| 110 | Reserved |
| 111 | Reserved |

**colour_description** – A flag which if set to '1' indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

**colour_primaries** – This 8-bit integer describes the chromaticity coordinates of the source primaries, and is defined in Table 6-7.

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero, the chromaticity is assumed to be implicitly defined by the application.

**transfer_characteristics** – This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is defined in Table 6-8.

**Table 6-7 – Colour Primaries**

| Value | Primaries | | |
|---|---|---|---|
| 0 | (Forbidden) | | |
| 1 | Recommendation ITU-R BT.709 | | |
| | primary | x | y |
| | green | 0.300 | 0.600 |
| | blue | 0.150 | 0.060 |
| | red | 0.640 | 0.330 |
| | white D65 | 0.3127 | 0.3290 |
| 2 | Unspecified Video | | |
| | Image characteristics are unknown | | |
| 3 | Reserved | | |
| 4 | Recommendation ITU-R BT.470-2 System M | | |
| | primary | x | y |
| | green | 0.21 | 0.71 |
| | blue | 0.14 | 0.08 |
| | red | 0.67 | 0.33 |
| | white C | 0.310 | 0.316 |
| 5 | Recommendation ITU-R BT.470-2 System B, G | | |
| | primary | x | y |
| | green | 0.29 | 0.60 |
| | blue | 0.15 | 0.06 |
| | red | 0.64 | 0.33 |
| | white D65 | 0.313 | 0.329 |
| 6 | SMPTE 170M | | |
| | primary | x | y |
| | green | 0.310 | 0.595 |
| | blue | 0.155 | 0.070 |
| | red | 0.630 | 0.340 |
| | white D65 | 0.3127 | 0.3290 |
| 7 | SMPTE 240M (1987) | | |
| | primary | x | y |
| | green | 0.310 | 0.595 |
| | blue | 0.155 | 0.070 |
| | red | 0.630 | 0.340 |
| | white D65 | 0.3127 | 0.3291 |
| 8-255 | Reserved | | |

**Table 6-8 – Transfer Characteristics**

| Value | Transfer Characteristic |
|---|---|
| 0 | (Forbidden) |
| 1 | Recommendation ITU-R BT.709<br>$V = 1.099\ L_c^{0.45} - 0.099$<br>for $1 \geq L_c \geq 0.018$<br>$V = 4.500\ L_c$<br>for $0.018 > L_c \geq 0$ |
| 2 | Unspecified Video<br>Image characteristics are unknown |
| 3 | Reserved |
| 4 | Recommendation ITU-R BT.470-2 System M<br>Assumed display gamma 2.2 |
| 5 | Recommendation ITU-R BT.470-2 System B, G<br>Assumed display gamma 2.8 |
| 6 | SMPTE 170M<br>$V = 1.099\ L_c^{0.45} - 0.099$<br>for $1 \geq L_c \geq 0.018$<br>$V = 4.500\ L_c$<br>for $0.018 > L_c \geq 0$ |
| 7 | SMPTE 240M (1987)<br>$V = 1.1115\ L_c^{0.45} - 0.1115$<br>for $L_c \geq 0.0228$<br>$V = 4.0\ L_c$<br>for $0.0228 > L_c$ |
| 8 | Linear transfer characteristics<br>i.e. $V = L_c$ |
| 9-255 | Reserved |

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero, the transfer characteristics are assumed to be implicitly defined by the application.

**matrix_coefficients** – This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, and is defined in Table 6-9.

**Table 6-9 – Matrix Coefficients**

| Value | Matrix |
|---|---|
| 0 | (Forbidden) |
| 1 | Recommendation ITU-R BT.709<br>$E'_Y = 0.7154\ E'_G + 0.0721\ E'_B + 0.2125\ E'_R$<br>$E'_{PB} = -0.386\ E'_G + 0.500\ E'_B - 0.115\ E'_R$<br>$E'_{PR} = -0.454\ E'_G - 0.046\ E'_B + 0.500\ E'_R$ |
| 2 | Unspecified Video<br>Image characteristics are unknown |
| 3 | Reserved |
| 4 | FCC<br>$E'_Y = 0.59\ E'_G + 0.11\ E'_B + 0.30\ E'_R$<br>$E'_{PB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$<br>$E'_{PR} = -0.421\ E'_G - 0.079\ E'_B + 0.500\ E'_R$ |
| 5 | Recommendation ITU-R BT.470-2 System B, G<br>$E'_Y = 0.587\ E'_G + 0.114\ E'_B + 0.299\ E'_R$<br>$E'_{PB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$<br>$E'_{PR} = -0.419\ E'_G - 0.081\ E'_B + 0.500\ E'_R$ |
| 6 | SMPTE 170M<br>$E'_Y = 0.587\ E'_G + 0.114\ E'_B + 0.299\ E'_R$<br>$E'_{PB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$<br>$E'_{PR} = -0.419\ E'_G - 0.081\ E'_B + 0.500\ E'_R$ |
| 7 | SMPTE 240M (1987)<br>$E'_Y = 0.701\ E'_G + 0.087\ E'_B + 0.212\ E'_R$<br>$E'_{PB} = -0.384\ E'_G + 0.500\ E'_B - 0.116\ E'_R$<br>$E'_{PR} = -0.445\ E'_G - 0.055\ E'_B + 0.500\ E'_R$ |
| 8-255 | Reserved |

In Table 6-9:

- $E'_Y$ is analogue with values between 0 and 1;
- $E'_{PB}$ and $E'_{PR}$ are analogue between the values –0.5 and 0.5;
- $E'_R$, $E'_G$ and $E'_B$ are analogue with values between 0 and 1;
- Y, Cb and Cr are related to $E'_Y$, $E'_{PB}$ and $E'_{PR}$ by the following formulae:

$$Y = ( 219 * E'_Y ) + 16$$
$$Cb = ( 224 * E'_{PB} ) + 128$$
$$Cr = ( 224 * E'_{PR} ) + 128$$

NOTE 1 – The decoding process given by this Specification limits output sample values for Y, Cr and Cb to the range [0:255]. Thus, sample values outside the range implied by the above equations may occasionally occur at the output of the decoding process. In particular the sample values 0 and 255 may occur.

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the matrix coefficients are assumed to be implicitly defined by the application.

NOTE 2 – In applications which may have signals with more than one set of colour primaries, transfer characteristics, and/or matrix coefficients, it is recommended to transmit a sequence display extension with colour_description set to one, and to specify the appropriate values for the colorimetry parameters.

**display_horizontal_size** – See display_vertical_size.

**display_vertical_size** – display_horizontal_size and display_vertical_size together define a rectangle which may be considered as the "intended display's" active region. If this rectangle is smaller than the encoded frame size, then the display process may be expected to display only a portion of the encoded frame. Conversely if the display rectangle is larger than the encoded frame size, then the display process may be expected to display the reconstructed frames on a portion of the display device rather than on the whole display device.

display_horizontal_size shall be in the same units as horizontal_size (samples of the encoded frames).

display_vertical_size shall be in the same units as vertical_size (lines of the encoded frames).

display_horizontal_size and display_vertical_size do not affect the decoding process but may be used by the display process that is not standardised in this Specification.

### 6.3.7 Sequence scalable extension

It is a syntactic restriction that if a sequence_scalable_extension() is present in the bitstream following a given sequence_extension(), then sequence_scalable_extension() shall follow every other occurrence of sequence_extension(). Thus a bitstream is either scalable or it is not scalable. It is not possible to mix scalable and non-scalable coding within a sequence.

**scalable_mode** – The scalable_mode indicates the type of scalability used in the video sequence. If no sequence_scalable_extension() is present in the bitstream, then no scalability is used for that sequence. scalable_mode also indicates the macroblock_type tables to be used. However, in the case of spatial scalability if no picture_spatial_scalable_extension() is present for a given picture, then that picture shall be decoded in a non-scalable manner (i.e. as if sequence_scalable_extension() had not been present).

**Table 6-10 – Definition of scalable_mode**

| scalable_mode | Meaning | picture_spatial_scalable-_extension() | macroblock_type tables |
|---|---|---|---|
| sequence_scalable_extension() not present | | | B-2, B-3 and B-4 |
| 00 | Data partitioning | | B-2, B-3 and B-4 |
| 01 | Spatial scalability | Present | B-5, B-6 and B-7 |
| | | Not present | B-2, B-3 and B-4 |
| 10 | SNR scalability | | B-8 |
| 11 | Temporal scalability | | B-2, B-3 and B-4 |

**layer_id** – This is an integer which identifies the layers in a scalable hierarchy. The base layer always has layer_id = 0. However, the base layer of a scalable hierarchy does not carry a sequence_scalable_extension() and hence layer_id, except in the case of data partitioning. Each successive layer has a layer_id which is one greater than the layer for which it is an enhancement.

In the case of data partitioning layer_id shall be zero for partition zero and layer_id shall be one for partition one.

**lower_layer_prediction_horizontal_size** – This is a 14-bit integer indicating the horizontal size of the lower layer frame which is used for prediction. This shall contain the value contained in horizontal_size (horizontal_size_value and horizontal_size_extension) in the lower layer bitstream.

**lower_layer_prediction_vertical_size** – This is a 14-bit integer indicating the vertical size of the lower layer frame which is used for prediction. This shall contain the value contained in vertical_size (vertical_size_value and vertical_size_extension) in the lower layer bitstream.

**horizontal_subsampling_factor_m** – This affects the spatial scalable upsampling process, as defined in 7.7.2. The value zero is forbidden.

**horizontal_subsampling_factor_n** – This affects the spatial scalable upsampling process, as defined in 7.7.2. The value zero is forbidden.

**vertical_subsampling_factor_m** – This affects the spatial scalable upsampling process, as defined in 7.7.2. The value zero is forbidden.

**vertical_subsampling_factor_n** – This affects the spatial scalable upsampling process, as defined in 7.7.2. The value zero is forbidden.

**picture_mux_enable** – If set to 1, picture_mux_order and picture_mux_factor are used for re-multiplexing prior to display.

**mux_to_progressive_sequence** – This flag when set to '1' indicates that the decoded pictures corresponding to the two layers shall be temporally multiplexed to generate a progressive sequence for display. When the temporal multiplexing is intended to generate an interlaced sequence this flag shall be '0'.

**picture_mux_order** – It denotes number of enhancement layer pictures prior to the first base layer picture. It thus assists re-multiplexing of pictures prior to display as it contains information for inverting the demultiplexing performed at the encoder.

**picture_mux_factor** – It denotes number of enhancement layer pictures between consecutive base layer pictures to allow correct re-multiplexing of base and enhancement layers for display. It also assists in re-multiplexing of pictures prior to display as it contains information for inverting the temporal demultiplexing performed at the encoder. The value '000' is reserved.

### 6.3.8 Group of pictures header

**group_start_code** – The group_start_code is the bit string '000001B8' in hexadecimal. It identifies the beginning of a group of pictures header.

**time_code** – This is a 25-bit integer containing the following: drop_frame_flag, time_code_hours, time_code_minutes, marker_bit, time_code_seconds and time_code_pictures as shown in Table 6-11. The parameters correspond to those defined in the IEC standard publication 60461 for "time and control codes for video tape recorders" (see Bibliography, Annex F). The time code refers to the first picture after the group of pictures header that has a temporal_reference of zero. The drop_frame_flag can be set to either '0' or '1'. It may be set to '1' only if the frame rate is 29.97 Hz. If it is '0' then pictures are counted assuming rounding to the nearest integral number of pictures per second, for example 29.97 Hz would be rounded to and counted as 30 Hz. If it is '1' then picture numbers 0 and 1 at the start of each minute, except minutes 0, 10, 20, 30, 40, 50 are omitted from the count.

NOTE – The information carried by time_code plays no part in the decoding process.

**Table 6-11 – time_code**

| time_code | Range of value | No. of bits | Mnemonic |
|---|---|---|---|
| **drop_frame_flag** | | 1 | uimsbf |
| **time_code_hours** | 0 - 23 | 5 | uimsbf |
| **time_code_minutes** | 0 - 59 | 6 | uimsbf |
| **marker_bit** | 1 | 1 | bslbf |
| **time_code_seconds** | 0 - 59 | 6 | uimsbf |
| **time_code_pictures** | 0 - 59 | 6 | uimsbf |

**closed_gop** – This is a one-bit flag which indicates the nature of the predictions used in the first consecutive B-pictures (if any) immediately following the first coded I-frame following the group of picture header.

closed_gop is set to '1' to indicate that these B-pictures have been encoded using only backward prediction or intra coding.

This bit is provided for use during any editing which occurs after encoding. If the previous pictures have been removed by editing, broken_link may be set to '1' so that a decoder may avoid displaying these B-Pictures following the first I-Picture following the group of picture header. However, if the closed_gop bit is set to '1', then the editor may choose not to set the broken_link bit as these B-Pictures can be correctly decoded.

**broken_link** – This is a one-bit flag which shall be set to '0' during encoding. It is set to '1' to indicate that the first consecutive B-Pictures (if any) immediately following the first coded I-frame following the group of picture header may not be correctly decoded because the reference frame which is used for prediction is not available (because of the action of editing).

A decoder may use this flag to avoid displaying frames that cannot be correctly decoded.

### 6.3.9 Picture header

**picture_start_code** – The picture_start_code is a string of 32 bits having the value 00000100 in hexadecimal.

**temporal_reference** – The temporal_reference is a 10-bit unsigned integer associated with each coded picture.

The following simple specification applies only when low_delay is equal to zero.

When a coded frame is in the form of two field pictures, the temporal_reference associated with each picture shall be the same (it is called the temporal_reference of the coded frame). The temporal_reference of each coded frame shall increment by one modulo 1024 when examined in display order at the output of the decoding process, except when a group of pictures header occurs. Among the frames coded after a group of pictures header, the temporal_reference of the coded frame that is displayed first, shall be set to zero.

The following more general specification applies when low_delay is equal to zero or one.

If picture A is not a big picture, i.e. the VBV buffer is only examined once before the coded picture A is removed from the VBV buffer and if N is the temporal_reference of picture A, then the temporal_reference of picture B immediately following picture A in display order is equal to:

- 0 if there is a group of pictures header present between picture A and picture B (in coded order).
- (N + 1) % 1024 if picture B is a frame picture or is the first field of a pair of field pictures.
- N if picture B is the second field of a pair of field pictures.

When low_delay is equal to one, there may be situations where the VBV buffer shall be re-examined several times before removing a coded picture (referred to as a big picture) from the VBV buffer.

If picture A is a big picture and if K is the number of times that the VBV buffer is re-examined as defined in C.7 (K > 0), if N is the temporal_reference of picture A, then the temporal_reference of picture B immediately following picture A in display order is equal to:

- K % 1024 if there is a group of pictures header present between picture A and picture B (in coded order).
- (N + K + 1) % 1024 if picture B is a frame picture or is the first field of a pair of field pictures.
- (N + K) % 1024 if picture B is the second field of a pair of field pictures.

NOTE 1 – If the big picture is the first field of a frame coded with field pictures, then the temporal_reference of the two field pictures of that coded frame are not identical.

**picture_coding_type** – The picture_coding_type identifies whether a picture is an intra-coded picture(I), predictive-coded picture(P) or bidirectionally predictive-coded picture(B). The meaning of picture_coding_type is defined in Table 6-12.

NOTE 2 – Intra-coded pictures with only DC coefficients (D-pictures) that may be used in ISO/IEC 11172-2 are not supported by this Specification.

**Table 6-12 – picture_coding_type**

| picture_coding_type | coding method |
|---|---|
| 000 | Forbidden |
| 001 | intra-coded (I) |
| 010 | predictive-coded (P) |
| 011 | bidirectionally-predictive-coded (B) |
| 100 | Shall not be used<br>(dc intra-coded (D) in ISO/IEC11172-2) |
| 101 | Reserved |
| 110 | Reserved |
| 111 | Reserved |

**vbv_delay** – The vbv_delay is a 16-bit unsigned integer. In all cases other then when vbv_delay has the value hexadecimal FFFF, the value of vbv_delay is the number of periods of a 90 kHz clock derived from the 27 MHz system clock that the VBV shall wait after receiving the final byte of the picture start code before decoding the picture. vbv_delay shall be coded to represent the delay as specified above or it shall be coded with the value hexadecimal FFFF. If any vbv_delay field in a sequence is coded with hexadecimal FFFF, then all of them shall be coded with this value. If vbv_delay takes the value hexadecimal FFFF, input of data to the VBV buffer is defined in C.3.2, otherwise input to the VBV buffer is defined in C.3.1.

If low_delay is equal '1' and if the bitstream contains big pictures, the vbv_delay values encoded in the picture_header() of big pictures may be wrong if not equal to hexadecimal FFFF.

NOTE 3 – There are several ways of calculating vbv_delay in an encoder.

In all cases it may be calculated by noting that the end-to-end delay through the encoder and decoder buffer is constant for all pictures. The encoder is capable of knowing the delay experienced by the relevant picture start code in the encoder buffer and the total end-to-end delay. Therefore, the value encoded in vbv_delay (the decoder buffer delay of the picture start code) is calculated as the total delay less the delay of the corresponding picture start code in the encoder buffer measured in periods of a 90 kHz clock derived from the 27 MHz system clock.

Alternatively, for constant bitrate operation only, vbv_delay may be calculated from the state of the VBV as follows:

$$\text{vbv\_delay}_n = 90\,000 * B_n^* / R$$

where:

$n > 0$

$B_n^*$ = VBV occupancy, measured in bits, immediately before removing picture n from the buffer but after removing any header(s), user data and stuffing that immediately precedes the data elements of picture n.

R = the actual bitrate (i.e. to full accuracy rather than the quantised value given by bit_rate in the sequence header).

An equivalent method of calculating vbv_delay for variable bitrate streams can be derived from the equation in C.3.1. This will be in the form of a recurrence relation for the vbv_delay given the previous vbv_delay, the decoding times of the current and previous pictures, and the number of bytes in the previous picture. This method can be applied if, at the time vbv_delay is encoded, the average bitrate of the transfer of the picture data of the previous picture is known.

**full_pel_forward_vector** – This flag that is used in ISO/IEC 11172-2 is not used by this Specification. It shall have the value '0'.

**forward_f_code** – This 3 bit string (which is used in ISO/IEC 11172-2) is not used by this Specification. It shall have the value '111'.

**full_pel_backward_vector** – This flag that is used in ISO/IEC 11172-2 is not used by this Specification. It shall have the value '0'.

**backward_f_code** – This 3 bit string (which is used in ISO/IEC 11172-2) is not used by this Specification. It shall have the value '111'.

**extra_bit_picture** – A bit indicates the presence of the following extra information. If extra_bit_picture is set to '1', extra_information_picture will follow it. If it is set to '0', there are no data following it. extra_bit_picture shall be set to '0', the value '1' is reserved for possible future extensions defined by ITU-T | ISO/IEC.

**extra_information_picture** – Reserved. A decoder conforming to this Specification that encounters extra_information_picture in a bitstream shall ignore it (i.e. remove from the bitstream and discard). A bitstream conforming to this Specification shall not contain this syntax element.

### 6.3.10 Picture coding extension

**f_code[s][t]** – A 4 bit unsigned integer taking values 1 through 9, or 15. The value zero is forbidden and the values 10 through 14 are reserved. It is used in the decoding of motion vectors, see 7.6.3.1.

In an I-picture in which concealment_motion_vectors is zero f_code[s][t] is not used (since motion vectors are not used) and shall take the value 15 (all ones).

Similarly, in an I-picture or a P-picture f_code[1][t] is not used in the decoding process (since it refers to backwards motion vectors) and shall take the value 15 (all ones).

See Table 7-7 for the meaning of the indices; s and t.

**intra_dc_precision** – This is a 2-bit integer defined in the Table 6-13.

**Table 6-13 – Intra DC precision**

| intra_dc_precision | Precision (bits) |
|---|---|
| 00 | 8 |
| 01 | 9 |
| 10 | 10 |
| 11 | 11 |

The inverse quantisation process for the Intra DC coefficients is modified by this parameter as explained in 7.4.1.

**picture_structure** – This is a 2-bit integer defined in the Table 6-14.

**Table 6-14 – Meaning of picture_structure**

| picture_structure | Meaning |
|---|---|
| 00 | Reserved |
| 01 | Top Field |
| 10 | Bottom Field |
| 11 | Frame picture |

When a frame is encoded in the form of two field pictures both fields must be of the same picture_coding_type, except where the first encoded field is an I-picture in which case the second may be either an I-picture or a P-picture.

The first encoded field of a frame may be a top-field or a bottom field, and the next field must be of opposite parity.

When a frame is encoded in the form of two field pictures, the following syntax elements may be set independently in each field picture:

- f_code[0][0], f_code[0][1];
- f_code[1][0], f_code[1][1];
- intra_dc_precision, concealment_motion_vectors, q_scale_type;
- intra_vlc_format, alternate_scan;
- vbv_delay;
- temporal_reference.

**top_field_first** – The meaning of this element depends upon picture_structure, progressive_sequence and repeat_first_field.

If progressive_sequence is equal to '0', this flag indicates what field of a reconstructed frame is output first by the decoding process.

In a field picture top_field_first shall have the value '0', and the only field output by the decoding process is the decoded field picture.

In a frame picture top_field_first being set to '1' indicates that the top field of the reconstructed frame is the first field output by the decoding process. top_field_first being set to '0' indicates that the bottom field of the reconstructed frame is the first field output by decoding process.

If progressive_sequence is equal to '1', this flag, combined with repeat_first_field, indicates how many times (one, two or three) the reconstructed frame is output by the decoding process.

If repeat_first_field is set to 0, top_field_first shall be set to '0'. In this case the output of the decoding process corresponding to this reconstructed frame consists of one progressive frame.

If top_field_first is set to 0 and repeat_first_field is set to '1', the output of the decoding process corresponding to this reconstructed frame consists of two identical progressive frames.

If top_field_first is set to 1 and repeat_first_field is set to '1', the output of the decoding process corresponding to this reconstructed frame consists of three identical progressive frames.

**frame_pred_frame_dct** – If this flag is set to '1', then only frame-DCT and frame prediction are used. In a field picture it shall be '0'. frame_pred_frame_dct shall be '1' if progressive_sequence is '1'. This flag affects the syntax of the bitstream.

**concealment_motion_vectors** – This flag has the value '1' to indicate that motion vectors are coded in intra macroblocks. This flag has the value '0' to indicate that no motion vectors are coded in intra macroblocks.

**q_scale_type** – This flag affects the inverse quantisation process as described in 7.4.2.2.

**intra_vlc_format** – This flag affects the decoding of transform coefficient data as described in 7.2.2.1.

**alternate_scan** – This flag affects the decoding of transform coefficient data as described in 7.3.

**repeat_first_field** – This flag is applicable only in a frame picture; in a field picture it shall be set to zero and does not affect the decoding process.

If progressive_sequence is equal to 0 and progressive_frame is equal to 0, repeat_first_field shall be zero, and the output of the decoding process corresponding to this reconstructed frame consists of two fields.

If progressive_sequence is equal to 0 and progressive_frame is equal to 1:

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of two fields. The first field (top or bottom field as identified by top_field_first) is followed by the other field.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of three fields. The first field (top or bottom field as identified by top_field_first) is followed by the other field, then the first field is repeated.

If progressive_sequence is equal to 1:

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of one frame.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of two or three frames, depending on the value of top_field_first.

**chroma_420_type** – If chroma_format is "4:2:0", the value of chroma_420_type shall be the same as progressive_frame; else chroma_420_type has no meaning and shall be equal to zero. This flag exists for historical reasons.

**progressive_frame** – If progressive_frame is set to 0 it indicates that the two fields of the frame are interlaced fields in which an interval of time of the field period exists between (corresponding spatial samples) of the two fields. In this case the following restriction applies:

- repeat_first_field shall be zero (two field duration).

If progressive_frame is set to 1 it indicates that the two fields (of the frame) are actually from the same time instant as one another. In this case a number of restrictions to other parameters and flags in the bitstream apply:

- picture_structure shall be "Frame";
- if progressive_sequence is equal to one, then frame_pred_frame_dct shall be 1.

progressive_frame is used when the video sequence is used as the lower layer of a spatial scalable sequence. Here it affects the up-sampling process used in forming a prediction in the enhancement layer from the lower layer.

**composite_display_flag** – This flag is set to 1 to indicate that the following fields that are of use when the input pictures have been coded as (analogue) composite video prior to encoding into a bitstream that complies with this Specification. If it is set to 0, then these parameters do not occur in the bitstream.

The information relates to the picture that immediately follows the extension. In the case that this picture is a frame picture, the information relates to the first field of that frame. The equivalent information for the second field may be derived (there is no way to represent it in the bitstream).

NOTE 1 – The various syntactic elements that are included in the bitstream if composite_display_flag is '1' are not used in the decoding process.

NOTE 2 – repeat_first_field will cause a composite video field to be repeated out of the 4-field or 8-field sequence. It is recommended that repeat_first_field and composite_display_flag are not both set simultaneously.

**v_axis** – A 1-bit integer used only when the bitstream represents a signal that had previously been encoded according to PAL systems. v_axis is set to 1 on a positive sign, v_axis is set to 0 otherwise.

**field_sequence** – A 3-bit integer which defines the number of the field in the eight field sequence used in PAL systems or the four field sequence used in NTSC systems as defined in the Table 6-15.

**Table 6-15 – Definition of field_sequence**

| Field sequence | Frame | Field |
|---|---|---|
| 000 | 1 | 1 |
| 001 | 1 | 2 |
| 010 | 2 | 3 |
| 011 | 2 | 4 |
| 100 | 3 | 5 |
| 101 | 3 | 6 |
| 110 | 4 | 7 |
| 111 | 4 | 8 |

**sub_carrier** – This is a 1-bit integer. Set to 0 means the sub-carrier/line frequency relationship is correct. When set to 1 the relationship is not correct.

**burst_amplitude** – This is a 7-bit integer defining the burst amplitude (for PAL and NTSC only). The amplitude of the sub-carrier burst is quantised as a Recommendation ITU-R BT.601 luminance signal, with the MSB omitted.

**sub_carrier_phase** – This is an 8-bit integer defining the phase of the reference sub-carrier at the field-synchronisation datum with respect, to field start as defined in Recommendation ITU-R BT.470 (see Table 6-16).

**Table 6-16 – Definition of sub_carrier_phase**

| sub_carrier_phase | Phase |
|---|---|
| 0 | ([360° ÷ 256] * 0) |
| 1 | ([360° ÷ 256] * 1) |
| … | … |
| 255 | ([360° ÷ 256] * 255) |

### 6.3.11 Quant matrix extension

Each quantisation matrix has a default set of values. When a sequence_header_code is decoded all matrices shall be reset to their default values. User defined matrices may be downloaded and this can occur in a sequence_header() or in a quant_matrix_extension().

With 4:2:0 data only two matrices are used, one for intra blocks the other for non-intra blocks.

With 4:2:2 or 4:4:4 data four matrices are used. Both an intra and a non-intra matrix are provided for both luminance blocks and for chrominance blocks. Note, however, that it is possible to download the same user defined matrix into both the luminance and chrominance matrix at the same time.

The default matrix for intra blocks (both luminance and chrominance) is:

| | | *u* | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 0 | 8 | 16 | 19 | 22 | 26 | 27 | 29 | 34 |
| | 1 | 16 | 16 | 22 | 24 | 27 | 29 | 34 | 37 |
| | 2 | 19 | 22 | 26 | 27 | 29 | 34 | 34 | 38 |
| | 3 | 22 | 22 | 26 | 27 | 29 | 34 | 37 | 40 |
| | 4 | 22 | 26 | 27 | 29 | 32 | 35 | 40 | 48 |
| | 5 | 26 | 27 | 29 | 32 | 35 | 40 | 48 | 58 |
| | 6 | 26 | 27 | 29 | 34 | 38 | 46 | 56 | 69 |
| *v* | 7 | 27 | 29 | 35 | 38 | 46 | 56 | 69 | 83 |

The default matrix for non-intra blocks (both luminance and chrominance) is:

| *v* \ *u* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 1 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 2 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 3 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 4 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 5 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 6 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 7 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |

**load_intra_quantiser_matrix** – This is a one-bit flag which is set to '1' if intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used.

**intra_quantiser_matrix** – This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.3.1, replace the previous values. The first value shall always be 8 (values 1 to 7 and 9 to 255 are reserved). For all of the 8-bit unsigned integers, the value zero is forbidden. With 4:2:2 and 4:4:4 data the new values shall be used for both the luminance intra matrix and the chrominance intra matrix. However, the chrominance intra matrix may subsequently be loaded with a different matrix.

**load_non_intra_quantiser_matrix** – This is a one-bit flag which is set to '1' if non_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used.

**non_intra_quantiser_matrix** – This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.3.1, replace the previous values. For all the 8-bit unsigned integers, the value zero is forbidden. With 4:2:2 and 4:4:4 data, the new values shall be used for both the luminance non-intra matrix and the chrominance non-intra matrix. However, the chrominance non-intra matrix may subsequently be loaded with a different matrix.

**load_chroma_intra_quantiser_matrix** – This is a one-bit flag which is set to '1' if chroma_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used. If chroma_format is "4:2:0", this flag shall take the value '0'.

**chroma_intra_quantiser_matrix** – This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.3.1, replace the previous values. The first value shall always be 8 (values 1 to 7 and 9 to 255 are reserved). For all of the 8-bit unsigned integers, the value zero is forbidden.

**load_chroma_non_intra_quantiser_matrix** – This is a one-bit flag which is set to '1' if chroma_non_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used. If chroma_format is "4:2:0", this flag shall take the value '0'.

**chroma_non_intra_quantiser_matrix** – This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.3.1, replace the previous values. For all the 8-bit unsigned integers, the value zero is forbidden.

### 6.3.12 Picture display extension

This Specification does not define the display process. The information in this extension does not affect the decoding process and may be ignored by decoders that conform to this Specification.

The picture display extension allows the position of the display rectangle whose size is specified in sequence_display_extension() to be moved on a picture-by-picture basis. One application for this is the implementation of pan-scan.

**frame_centre_horizontal_offset** – This is a 16-bit signed integer giving the horizontal offset in units of 1/16th sample. A positive value shall indicate that the centre of the reconstructed frame lies to the right of the centre of the display rectangle.

**frame_centre_vertical_offset** – This is a 16-bit signed integer giving the vertical offset in units of 1/16th sample. A positive value shall indicate that the centre of the reconstructed frame lies below the centre of the display rectangle.

The dimensions of the display rectangular region are defined in the sequence_display_extension(). The coordinates of the region within the coded picture are defined in the picture_display_extension().

The centre of the reconstructed frame is the centre of the rectangle defined by horizontal_size and vertical_size.

Since (in the case of an interlaced sequence) a coded picture may relate to one, two or three decoded fields, the picture_display_extension() may contain up to three offsets.

The number of frame centre offsets in the picture_display_extension() shall be defined as follows:

```
if ( progressive_sequence == 1) {
    if ( repeat_first_field == '1' ) {
        if ( top_field_first == '1' )
            number_of_frame_centre_offsets = 3
        else
            number_of_frame_centre_offsets = 2
    } else {
        number_of_frame_centre_offsets = 1
    }
} else {
    if (picture_structure == "field") {
        number_of_frame_centre_offsets = 1
    } else {
        if (repeat_first_field == '1' )
            number_of_frame_centre_offsets = 3
        else
            number_of_frame_centre_offsets = 2
    }
}
```

A picture_display_extension() shall not occur unless a sequence_display_extension() followed the previous sequence_header().

In the case that a given picture does not have a picture_display_extension(), then the most recently decoded frame centre offset shall be used. Note that each of the missing frame centre offsets have the same value (even if two or three frame centre offsets would have been contained in the picture_display_extension() had been present). Following a sequence_header() the value zero shall be used for all frame centre offsets until a picture_display_extension() defines non-zero values.

Figure 6-16 illustrates the picture display parameters. As shown, the frame centre offsets contained in the picture_display_extension() shall specify the position of the centre of the reconstructed frame from the centre of the display rectangle.

NOTE 1 – The display rectangle may also be larger than the reconstructed frame.

NOTE 2 – Even in a field picture the frame_centre_vertical_offset still represents the offset of the centre of the frame in 1/16$^{th}$ of a **frame** line (not a line in the field).

NOTE 3 – In the example of Figure 6-16 both frame_centre_horizontal_offset and frame_centre_vertical_offset have negative values.

#### 6.3.12.1 Pan-scan

The frame centre offsets may be used to implement pan-scan in which a rectangular region is defined which may be panned around the entire reconstructed frame.

By way of example only; this facility may be used to identify a 3/4 aspect ratio window in a 9/16 coded picture format. This would allow a decoder to produce usable pictures for a conventional definition television set from an encoded format intended for enhanced definition. The 3/4 aspect ratio region is intended to contain the "most interesting" region of the picture.

The 3/4 region is defined by display_horizontal_size and display_vertical_size. The 9/16 frame size is defined by horizontal_size and vertical_size.

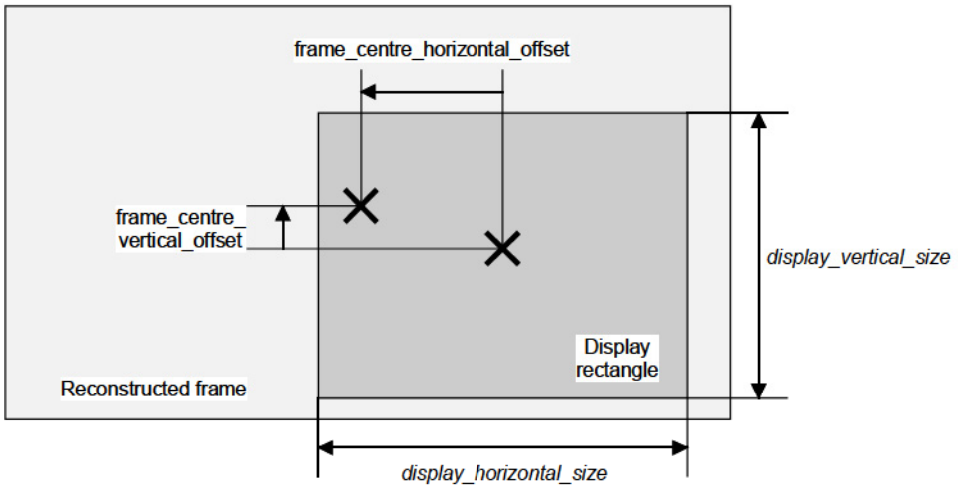



**Figure 6-16 – Frame centre offset parameters**

### 6.3.13 Picture temporal scalable extension

NOTE – See also 7.9.

**reference_select_code** – This is a 2-bit code that identifies reference frames or reference fields for prediction depending on the picture type.

**forward_temporal_reference** – A 10 bit unsigned integer value which indicates temporal reference of the lower layer frame to be used to provide the forward prediction. If the lower layer indicates temporal reference with more than 10 bits, the least significant bits are encoded here. If the lower layer indicates temporal reference with fewer than 10 bits, all bits are encoded here and the more significant bits shall be set to zero.

**backward_temporal_reference** – A 10 bit unsigned integer value which indicates temporal reference of the lower layer frame to be used to provide the backward prediction. If the lower layer indicates temporal reference with more than 10 bits, the least significant bits are encoded here. If the lower layer indicates temporal reference with fewer than 10 bits, all bits are encoded here and the more significant bits shall be set to zero.

### 6.3.14 Picture spatial scalable extension

**lower_layer_temporal_reference** – A 10 bit unsigned integer value which indicates temporal reference of the lower layer frame to be used to provide the prediction. If the lower layer indicates temporal reference with more than 10 bits, the least significant bits are encoded here. If the lower layer indicates temporal reference with fewer than 10 bits, all bits are encoded here and the more significant bits shall be set to zero.

**lower_layer_horizontal_offset** – This 15 bit signed (twos complement) integer specifies the horizontal offset (of the top left hand corner) of the upsampled lower layer frame relative to the enhancement layer picture. It is expressed in units of the enhancement layer picture sample width. If the chrominance format is 4:2:0 or 4:2:2, then this parameter shall be an even number.

**lower_layer_vertical_offset** – This 15 bit signed (twos complement) integer specifies the vertical offset (of the top left hand corner) of the upsampled lower layer picture relative to the enhancement layer picture. It is expressed in units of the enhancement layer picture sample height. If the chrominance format is 4:2:0, then this parameter shall be an even number.

**spatial_temporal_weight_code_table_index** – This 2-bit integer indicates which table of spatial temporal weight codes is to be used as defined in 7.7. Permissible values of spatial_temporal_weight_code_table_index are defined in Table 7-21.

**lower_layer_progressive_frame** – This flag shall be set to 0 if the lower layer frame is interlaced and shall be set to '1' if the lower layer frame is progressive. The use of this flag in the spatial scalable upsampling process is defined in 7.7.

**lower_layer_deinterlaced_field_select** – This flag affects the spatial scalable upsampling process, as defined in 7.7.

### 6.3.15 Copyright extension

**extension_start_code_identifier** – This is a 4-bit integer which identifies the extension (see Table 6-2).

**copyright_flag** – This is a one bit flag. When copyright_flag is set to '1', it indicates that the source video material encoded in all the coded pictures following the copyright extension, in coding order, up to the next copyright extension or end of sequence code, is copyrighted. The copyright_identifier and copyright_number identify the copyrighted work. When copyright_flag is set to '0', it does not indicate whether the source video material encoded in all the coded pictures following the copyright extension, in coding order, is copyrighted or not.

**copyright_identifier** – This is an 8-bit integer given by a Registration Authority as designated by ISO/IEC JTC1/SC29. Value zero indicates that this information is not available. The value of copyright_number shall be zero when copyright_identifier is equal to zero.

When copyright_flag is set to '0', copyright_identifier has no meaning and shall have the value 0.

**original_or_copy** – This is a one bit flag. It is set to '1' to indicate that the material is an original, and set to '0' to indicate that it is a copy.

**reserved** – This is a 7-bit integer, reserved for future extension. It shall have the value zero.

**copyright_number_1** – This is a 20-bit integer, representing bits 44 to 63 of copyright_number.

**copyright_number_2** – This is a 22-bit integer, representing bits 22 to 43 of copyright_number.

**copyright_number_3** – This is a 22-bit integer. representing bits 0 to 21 of copyright_number.

**copyright_number** – This is a 64-bit integer, derived from copyright_number_1, copyright_number_2, and copyright_number_3 as follows:

copyright_number = (copyright_number_1 << 44) + (copyright_number_2 << 22) + copyright_number_3.

The meaning of copyright_number is defined only when copyright_flag is set to '1'. In this case, the value of copyright_number identifies uniquely the copyrighted work marked by the copyrighted extension. The value 0 for copyright_number indicates that the identification number of the copyrighted work is not available.

When copyright_flag is set to '0', copyright_number has no meaning and shall have the value 0.

### 6.3.16 Slice

**slice_start_code** – The slice_start_code is a string of 32-bits. The first 24-bits have the value 000001 in hexadecimal and the last 8-bits are the slice_vertical_position having a value in the range 01 through AF hexadecimal inclusive.

**slice_vertical_position** – This is given by the last eight bits of the slice_start_code. It is an unsigned integer giving the vertical position in macroblock units of the first macroblock in the slice.

In large pictures (when the vertical size of the frame is greater than 2800 lines) the slice vertical position is extended by the **slice_vertical_position_extension**.

The macroblock row may be calculated as follows:

```
if ( vertical_size > 2800 )
        mb_row = (slice_vertical_position_extension << 7) + slice_vertical_position –1;
    else
        mb_row = slice_vertical_position –1;
```

The slice_vertical_position of the first row of macroblocks is one. Some slices may have the same slice_vertical_position, since slices may start and finish anywhere. The maximum value of slice_vertical_position is 175 unless slice_vertical_position_extension is present in which case slice_vertical_position shall be in the range [1:128].

**priority_breakpoint** – This is a 7-bit integer that indicates the point in the syntax where the bitstream shall be partitioned. The allowed values and their semantic interpretation is given in Table 7-30 priority_breakpoint shall take the value zero in partition 1.

**quantiser_scale_code** – A 5 bit unsigned integer in the range 1 to 31 . The decoder shall use this value until another quantiser_scale_code is encountered either in slice() or macroblock(). The value zero is forbidden.

**slice_extension_flag** – This flag shall be set to '1' to indicate the presence of intra_slice, slice_picture_id_enable, slice_picture_id in the bitstream.

**intra_slice** – This flag shall be set to '0' if any of the macroblocks in the slice are non-intra macroblocks. If all of the macroblocks, are intra macroblocks, then intra_slice may be set to '1'.

intra_slice may be omitted from the bitstream (by setting intra_slice_flag to '0') in which case it shall be assumed to have the value zero.

intra_slice is not used by the decoding process. intra_slice is intended to aid a DSM application in performing FF/FR (see D.12).

**slice_picture_id_enable** – This flag controls the semantics of slice_picture_id. If slice_picture_id_enable is set to '0', slice_picture_id is not used by this Specification and shall have the value zero. If slice_picture_id_enable is set to '1', slice_picture_id may have a value different from zero.

slice_picture_id_enable must have the same value in all the slices of a picture. slice_picture_id_enable may be omitted from the bitstream (by setting slice_extension_flag to '0') in which case it shall be assumed to have the value zero.

slice_picture_id_enable is not used by the decoding process.

**slice_picture_id** – This is a 6-bit integer. If slice_picture_id_enable is set to '0', slice_picture_id is not used by this Specification and shall have the value zero. If slice_picture_id_enable is set to '1', slice_picture_id is application defined and may have any value, with the constraint that slice_picture_id shall have the same value in all the slices of a picture.

slice_picture_id is not used by the decoding process. slice_picture_id is intended to aid recovery on severe bursts of errors for certain types of applications. For example, the application may increment slice_picture_id with each transmitted picture, so that in case of severe burst error, when several slices are lost, the decoder can know if the slice following the burst error belongs to the current picture or to another picture, which may be the case if at least a picture header has been lost.

**extra_bit_slice** – This flag indicates the presence of the following extra information. If extra_bit_slice is set to '1', extra_information_slice will follow it. If it is set to '0', there are no data following it. extra_bit_slice shall be set to '0', the value '1' is reserved for possible future extensions defined by ITU-T | ISO/IEC.

**extra_information_slice** – Reserved. A decoder conforming to this Specification that encounters extra_information_slice in a bitstream shall ignore it (i.e. remove from the bitstream and discard). A bitstream conforming to this Specification shall not contain this syntax element.

### 6.3.17 Macroblock

NOTE – "macroblock_stuffing" which is supported in ISO/IEC 11172-2 shall not be used in a bitstream defined by this Specification.

**macroblock_escape** – The macroblock_escape is a fixed bit-string '0000 0001 000' which is used when the difference between macroblock_address and previous_macroblock_address is greater than 33. It causes the value of macroblock_address_increment to be 33 greater than the value that will be decoded by subsequent macroblock_escape and the macroblock_address_increment codewords.

For example, if there are two macroblock_escape codewords preceding the macroblock_address_increment, then 66 is added to the value indicated by macroblock_address_increment.

**macroblock_address_increment** – This is a variable length coded integer coded as per Table B.1 which indicates the difference between macroblock_address and previous_macroblock_address. The maximum value of macroblock_address_increment is 33. Values greater than this can be encoded using the macroblock_escape codeword.

The macroblock_address is a variable defining the absolute position of the current macroblock. The macroblock_address of the top-left macroblock is zero.

The previous_macroblock_address is a variable defining the absolute position of the last non-skipped macroblock (see 7.6.6 for the definition of skipped macroblocks) except at the start of a slice. At the start of a slice previous_macroblock_address is reset as follows:

$$\text{previous\_macroblock\_address} = (\text{mb\_row} * \text{mb\_width}) - 1$$

The horizontal spatial position in macroblock units of a macroblock in the picture (mb_column) can be computed from the macroblock_address as follows:

$$\text{mb\_column} = \text{macroblock\_address} \ \% \ \text{mb\_width}$$

where mb_width is the number of macroblocks in one row of the picture.

Except at the start of a slice, if the value of macroblock_address recovered from macroblock_address_increment and the macroblock_escape codes (if any) differs from the previous_macroblock_address by more than one then some macroblocks have been skipped. It is a requirement that:

- There shall be no skipped macroblocks in I-pictures except when either
  - picture_spatial_scalable_extension() follows the picture_header() of the current picture; or
  - sequence_scalable_extension() is present in the bitstream and scalable_mode = "SNR scalability".
- In a B-picture there shall be no skipped macroblocks immediately following a macroblock in which macroblock_intra is one.

It should be noted that the syntax does not allow the first and last macroblock of a slice to be skipped.

#### 6.3.17.1 Macroblock modes

**macroblock_type** – Variable length coded indicator which indicates the method of coding and content of the macroblock according to the Tables B.2 through B.8, selected by picture_coding_type and scalable_mode.

**macroblock_quant** – Derived from macroblock_type according to the Tables B.2 through B.8. This is set to 1 to indicate that quantiser_scale_code is present in the bitstream.

**macroblock_motion_forward** – Derived from macroblock_type according to the Tables B.2 through B.8. This flag affects the bitstream syntax and is used by the decoding process.

**macroblock_motion_backward** – Derived from macroblock_type according to the Tables B.2 through B.8. This flag affects the bitstream syntax and is used by the decoding process.

**macroblock_pattern** – Derived from macroblock_type according to the Tables B.2 through B.8. This is set to 1 to indicate that coded_block_pattern() is present in the bitstream.

**macroblock_intra** – Derived from macroblock_type according to the Tables B.2 through B.8. This flag affects the bitstream syntax and is used by the decoding process.

**spatial_temporal_weight_code_flag** – Derived from the macroblock_type. This indicates whether the spatial_temporal_weight_code is present in the bitstream.

When spatial_temporal_weight_code_flag is '0' (indicating that spatial_temporal_weight_code is not present in the bitstream) the spatial_temporal_weight_class is derived from Tables B.5 to B.7. When spatial_temporal_weight_code_flag is '1' spatial_temporal_weight_class is derived from Table 7-20.

**spatial_temporal_weight_code** – This is a two bit code which indicates, in the case of spatial scalability, how the spatial and temporal predictions shall be combined to form the prediction for the macroblock. A full description of how to form the spatial scalable prediction is given in 7.7.

**frame_motion_type** – This is a two bit code indicating the macroblock prediction type, defined in Table 6-17.

**Table 6-17 – Meaning of frame_motion_type**

| Code | spatial_temporal_weight_class | Prediction type | motion_vector_count | mv_format | dmv |
|---|---|---|---|---|---|
| 00 | | Reserved | | | |
| 01 | 0, 1 | Field-based | 2 | Field | 0 |
| 01 | 2, 3 | Field-based | 1 | Field | 0 |
| 10 | 0, 1, 2, 3 | Frame-based | 1 | Frame | 0 |
| 11 | 0, 2, 3 | Dual-Prime | 1 | Field | 1 |

If frame_pred_frame_dct is equal to 1 then frame_motion_type is omitted from the bitstream. In this case motion vector decoding and prediction formation shall be performed as if frame_motion_type had indicated "Frame-based prediction".

In the case of intra macroblocks (in a frame picture) when concealment_motion_vectors is equal to 1 frame_motion_type is not present in the bitstream. In this case motion vector decoding and update of the motion vector predictors shall be performed as if frame_motion_type had indicated "Frame-based" (see 7.6.3.9).

**field_motion_type** – This is a two bit code indicating the macroblock prediction type, defined in Table 6-18.

**Table 6-18 – Meaning of field_motion_type**

| Code | spatial_temporal _weight_class | Prediction type | motion_vector _count | mv_format | dmv |
|---|---|---|---|---|---|
| 00 | | Reserved | | | |
| 01 | 0, 1 | Field-based | 1 | Field | 0 |
| 10 | 0, 1 | 16 × 8 MC | 2 | Field | 0 |
| 11 | 0 | Dual-Prime | 1 | Field | 1 |

In the case of intra macroblocks (in a field picture) when concealment_motion_vectors is equal to 1 field_motion_type is not present in the bitstream. In this case, motion vector decoding and update of the motion vector predictors shall be performed as if field_motion_type had indicated "Field-based" (see 7.6.3.9).

**dct_type** – This is a flag indicating whether the macroblock is frame DCT coded or field DCT coded. If this is set to '1', the macroblock is field DCT coded.

In the case that **dct_type** is not present in the bitstream, then the value of dct_type (used in the remainder of the decoding process) shall be derived as shown in Table 6-19.

**Table 6-19 – Value of dct_type if dct_type is not in the bitstream**

| Condition | dct_type |
|---|---|
| picture_structure == "field" | Unused because there is no frame/field distinction in a field picture. |
| frame_pred_frame_dct == 1 | 0 (“frame") |
| !(macroblock_intra \|\| macroblock_pattern) | Unused – Macroblock is not coded |
| macroblock is skipped | Unused – Macroblock is not coded |

#### 6.3.17.2 Motion vectors

motion_vector_count is derived from field_motion_type or frame_motion_type as indicated in Tables 6-17 and 6-18.

mv_format is derived from field_motion_type or frame_motion_type as indicated in the Tables 6-17 and 6-18. mv_format indicates if the motion vector is a field-motion vector or a frame-motion vector. mv_format is used in the syntax of the motion vectors and in the process of motion vector prediction.

dmv is derived from field_motion_type or frame_motion_type as indicated in Tables 6-17 and 6-18.

**motion_vertical_field_select[r][s]** – This flag indicates which reference field shall be used to form the prediction. If motion_vertical_field_select[r][s] is zero, then the top reference field shall be used, if it is one then the bottom reference field shall be used. (See Table 7-7 for the meaning of the indices r and s.)

#### 6.3.17.3 Motion vector

**motion_code[r][s][t]** – This is a variable length code, as defined in Table B.10, which is used in motion vector decoding as described in 7.6.3.1. (See Table 7-7 for the meaning of the indices r, s and t.)

**motion_residual[r][s][t]** – This is an integer which is used in motion vector decoding as described in 7.6.3.1. (See Table 7-7 for the meaning of the indices r, s and t.) The number of bits in the bitstream for motion_residual[r][s][t], r_size, is derived from f_code[s][t] as follows:

$$r\_size = f\_code[s][t] - 1$$

NOTE – The number of bits for both motion_residual[0][s][t] and motion_residual[1][s][t] is denoted by f_code[s][t].

**dmvector[t]** – This is a variable length code, as defined in Table B.11, which is used in motion vector decoding as described in 7.6.3.6. (See Table 7-7 for the meaning of the index t.)

#### 6.3.17.4 Coded block pattern

**coded_block_pattern_420** – A variable length code that is used to derive the variable cbp according to Table B.9.

**coded_block_pattern_1**

**coded_block_pattern_2** – For 4:2:2 and 4:4:4 data the coded block pattern is extended by the addition of either a two bit or six bit fixed length code, coded_block_pattern_1 or coded_block_pattern_2. Then the pattern_code[i] is derived using the following:

```
for (i = 0; i < 12; i++) {
    if (macroblock_intra)
        pattern_code[i] = 1;
    else
        pattern_code[i] = 0;
}
if (macroblock_pattern) {
    for (i = 0; i < 6; i++)
        if ( cbp & (1<<(5 – i)) ) pattern_code[i] = 1;
    if (chroma_format == "4:2:2")
        for (i = 6; i < 8; i++)
            if ( coded_block_pattern_1 & (1<<(7 – i)) ) pattern_code[i] = 1;
    if (chroma_format == "4:4:4")
        for (i = 6; i < 12; i++)
            if ( coded_block_pattern_2 & (1<<(11 – i)) ) pattern_code[i] = 1;
}
```

If pattern_code[i] equals to 1, i = 0 to (block_count-1), then the block number i defined in Figures 6-10, 6-11 and 6-12 is contained in this macroblock.

The number "block_count" which determines the number of blocks in the macroblock is derived from the chrominance format as shown in Table 6-20.

**Table 6-20 – block_count as a function of chroma_format**

| chroma_format | block_count |
|---|---|
| 4:2:0 | 6 |
| 4:2:2 | 8 |
| 4:4:4 | 12 |

#### 6.3.18 Block

The semantics of block() are described in clause 7.

#### 6.3.19 Camera parameters extension

**camera_id** – The number in camera_id identifies a camera.

**height_of_image_device** – This is a 22-bit unsigned integer which specifies the height of image device. Its value shall be measured to a resolution of 0,001 millimeter and having a range of zero to 4 194,303 mm.

**focal_length** – This is a 22-bit unsigned integer which specifies the focal length. Its value shall be measured to a resolution of 0,001 millimeter and having a range of zero to 4 194,303 mm.

**f_number** – This is a 22-bit unsigned integer which specifies the F-number. F-number is defined by (focal_length)/(effective aperture of lens). Its value shall be measured to a resolution of 0,001 and having a range of zero to 4 194,303.

**vertical_angle_of_view** – This is a 22-bit unsigned integer which specifies the vertical angle of the field of view as determined between the top and bottom edges of the image device. Its value shall be measured to a resolution of 0,0001 degree and having a range of zero to 180 degrees.

**camera_position_x_upper, camera_position_y_upper, camera_position_z_upper** – These words constitute the 16 most significant bits of camera_position_x, camera_position_y and camera_position_z respectively.

**camera_position_x_lower, camera_position_y_lower, camera_position_z_lower** – These words constitute the 16 least significant bits of camera_position_x, camera_position_y and camera_position_z respectively.

**camera_position_x, camera_position_y, camera_position_z** – A set of these values specifies the position of the optical principal point of the camera in a user-specified world coordinate system. Each of these values shall be measured to a resolution of 0,001 millimeter and having a range of +2 147 483,647 mm to –2 147 483,648 mm. The camera_position_x is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_x_lower, the 16 most significant bits are defined in camera_position_x_upper. The camera_position_y is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_y_lower, the 16 most significant bits are defined in camera_position_y_upper. The camera_position_z is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_z_lower, the 16 most significant bits are defined in camera_position_z_upper.

**camera_direction_x, camera_direction_y, camera_direction_z** – A set of these values specifies the direction of the camera. The direction of the camera is defined by using the vector from optical principal point to a point which is in front of the camera and is on the optical axis of the camera. Each of these values is a 22-bit signed (two's complement) integer and having a range of +2 097 151 to –2 097 152.

**image_plane_vertical_x, image_plane_vertical_y, image_plane_vertical_z** – A set of these values specifies the upper direction of the camera. The upper direction of the camera is defined by using the vector which is parallel to the side edge of the image device and is from bottom edge to top edge. Each of these values is a 22-bit signed (two's complement) integer and is having a range of +2 097 151 to –2 097 152.

Figure 6-17 explains these terms pictorially.

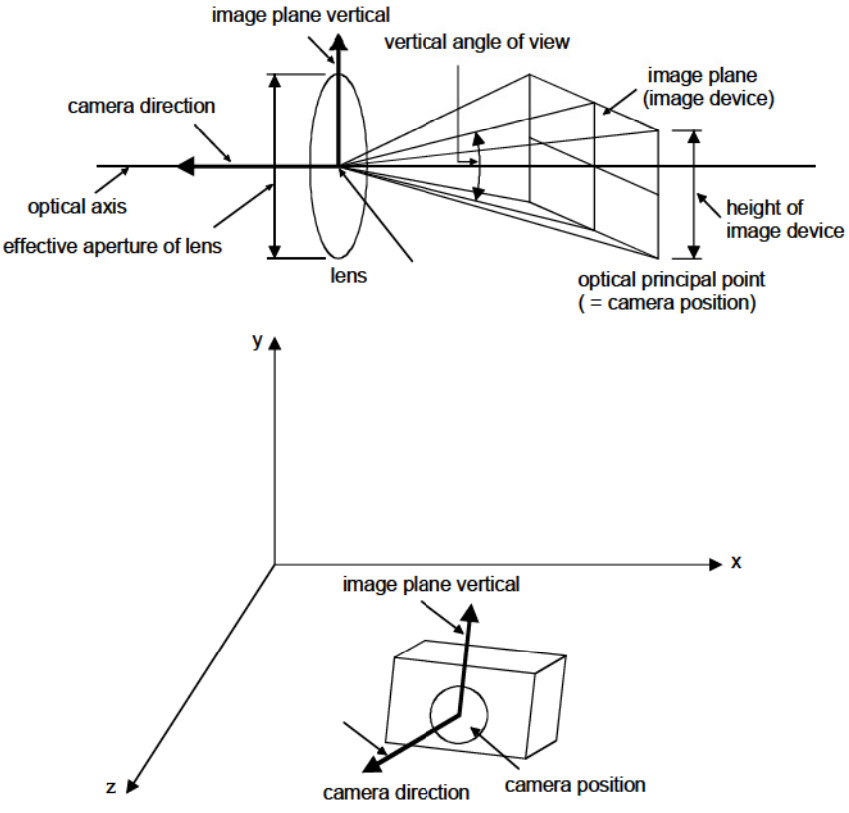



**Figure 6-17 -- Camera parameters**

### 6.3.20 ITU-T extension

The use of this extension is defined in Annex A/H.320.

# 7 The video decoding process

This clause specifies the decoding process that a decoder shall perform to reconstruct frames from the coded bitstream.

The IDCT function f[y][x] used in the decoding process may be any of several approximations of the saturated mathematical integer-number IDCT defined in Annex A. Requirements on the accuracy of the IDCT function used in the decoding process are specified in Annex A.

In 7.1 through 7.6 the simplest decoding process is specified in which no scalability features are used. Subclauses 7.7 to 7.11 specify the decoding process when scalable extensions are used. Subclause 7.12 defines the output of the decoding process.

Figure 7-1 is a diagram of the Video Decoding Process without any scalability. The diagram is simplified for clarity.

NOTE – Throughout this Specification two dimensional arrays are represented as *name*[*q*][*p*] where '*q*' is the index in the vertical dimension and '*p*' the index in the horizontal dimension.

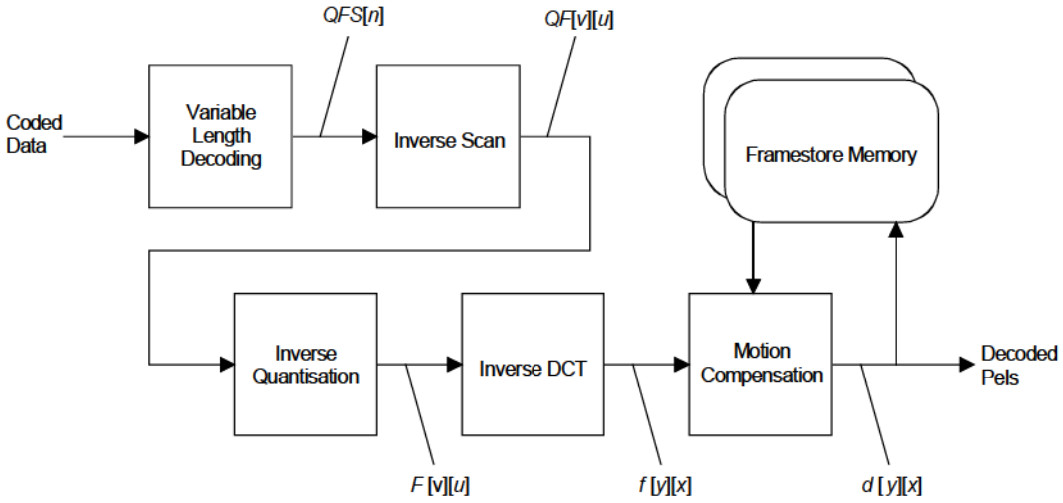



**Figure 7-1 – Simplified Video Decoding Process**

## 7.1 Higher syntactic structures

The various parameters and flags in the bitstream for macroblock() and all syntactic structures above macroblock() shall be interpreted as indicated in clause 6. Many of these parameters and flags affect the decoding process described in the following subclauses. Once all of the macroblocks in a given picture have been processed, the entire picture will have been reconstructed.

Reconstructed fields shall be associated together in pairs to form reconstructed frames. (See "picture_structure" in 6.3.10.)

The sequence of reconstructed frames shall be re-ordered as described in 6.1.1.11.

If progressive_sequence == 1 the reconstructed frames shall be output from the decoding process at regular intervals of the frame period as shown in Figure 7-19.

If progressive_sequence == 0 the reconstructed frames shall be broken into a sequence of fields which shall be output from the decoding process at regular intervals of the field period as shown in Figure 7-20. In the case that a frame picture has repeat_first_field == 1 the first field of the frame shall be repeated after the second field. (See "repeat_first_field" in 6.3.10.)

## 7.2 Variable length decoding

Subclause 7.2.1 specifies the decoding process used for the DC coefficient ($n = 0$) in an intra coded block. ($n$ is the index of the coefficient in the appropriate zigzag scanning order.) Subclause 7.2.2 specifies the decoding process for all other coefficients, AC coefficients ($n > 0$) and DC coefficients in non-intra coded blocks.

Let *cc* denote the colour component. It is related to the block number as specified in Table 7-1. Thus, *cc* is zero for the Y component, one for the Cb component and two for the Cr component.

**Table 7-1 – Definition of *cc*, colour component index**

| Block Number | *cc* | | |
|---|---|---|---|
| | 4:2:0 | 4:2:2 | 4:4:4 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 |
| 5 | 2 | 2 | 2 |
| 6 | | 1 | 1 |
| 7 | | 2 | 2 |
| 8 | | | 1 |
| 9 | | | 2 |
| 10 | | | 1 |
| 11 | | | 2 |

### 7.2.1 DC coefficients in intra blocks

DC coefficients in blocks in intra macroblocks are encoded as a variable length code denoting dct_dc_size as defined in Tables B.12 and B.13. If dct_dc_size is not equal to zero, then this shall be followed by a fixed length code, dct_dc_differential, of dct_dc_size bits. A differential value is first recovered from the coded data which is added to a predictor in order to recover the final decoded coefficient.

If *cc* is zero then Table B.12 shall be used for dct_dc_size. If *cc* is non-zero, then Table B.13 shall be used for dct_dc_size.

Three predictors are maintained, one for each of the colour components, *cc*. Each time a DC coefficient in a block in an intra macroblock is decoded the predictor is added to the differential to recover the actual coefficient. Then the predictor shall be set to the value of the coefficient just decoded. At various times, as described below, the predictors shall be reset. The reset value is derived from the parameter intra_dc_precision as specified in Table 7-2.

**Table 7-2 – Relation between intra_dc_precision and the predictor reset value**

| intra_dc_precision | Bits of precision | reset value |
|---|---|---|
| 0 | 8 | 128 |
| 1 | 9 | 256 |
| 2 | 10 | 512 |
| 3 | 11 | 1024 |

The predictors shall be reset to the reset value at the following times:

- At the start of a slice.
- Whenever a non-intra macroblock is decoded.
- Whenever a macroblock is skipped. i.e. when macroblock_address_increment > 1.

The predictors are denoted *dct_dc_pred*[*cc*].

*QFS*[0] shall be calculated from dct_dc_size and dct_dc_differential by any process equivalent to:

```
if ( dct_dc_size == 0 ) {
    dct_diff = 0;
} else {
    half_range = 2 ^ ( dct_dc_size - 1 );
    if ( dct_dc_differential > = half_range )
        dct_diff = dct_dc_differential;
    else
        dct_diff = (dct_dc_differential + 1) - (2 * half_range);
}
QFS[0] = dct_dc_pred[cc] + dct_diff;
dct_dc_pred[cc] = QFS[0]
```

NOTE 1 – The symbol ^ *denotes power (not XOR).*

NOTE 2 – *dct_diff* and *half_range* are temporary variables which are not used elsewhere in this Specification.

It is a requirement of the bitstream that QFS[0] shall lie in the range:

$$0 \text{ to } ((2^{(8 + \text{intra\_dc\_precision})}) - 1)$$

### 7.2.2 Other coefficients

All coefficients with the exception of the DC intra coefficients shall be encoded using Tables B.14, B.15 and B.16.

In all cases a variable length code shall first be decoded using either Table B.14 or Table B.15. The decoded value of this code denotes one of three courses of action:

1) *End of Block* – In this case there are no more coefficients in the block in which case the remainder of the coefficients in the block (those for which no value has yet been decoded) shall be set to zero. This is denoted by "End of block" in the syntax specification of 6.2.6.

2) A "normal" coefficient in which a value of *run* and *level* is decoded followed by a single bit, *s*, giving the sign of the coefficient *signed_level* is computed from *level* and *s* as shown below. *run* coefficients shall be set to zero and the subsequent coefficient shall have the value *signed_level*.

```
if (s ==0)
        signed_level = level;
    else
        signed_level = (-level);
```

3) An "Escape" coded coefficient. In which the values of *run* and *signed_level* are fixed length coded as described in 7.2.2.3.

#### 7.2.2.1 Table selection

Table 7-3 indicates which Table shall be used for decoding the DCT coefficients.

**Table 7-3 – Selection of DCT coefficient VLC tables**

| intra_vlc_format | 0 | 1 |
|---|---|---|
| **intra blocks (macroblock_intra = 1)** | B.14 | B.15 |
| **non-intra blocks (macroblock_intra = 0)** | B.14 | B.14 |

#### 7.2.2.2 First coefficient of a non-intra block

In the case of the first coefficient of a non-intra block (a block in a non-intra macroblock) Table B.14 is modified as indicated by Notes 2 and 3 at the foot of that Table.

This modification only affects the entry that represents run = 0, level = ± 1. Since it is not possible to encode an End of block as the first coefficient of a block (the block would be "not coded" in this case) no possibility for ambiguity exists.

The positions in the syntax that use this modified Table are denoted by "First DCT coefficient" in the syntax specification of 6.2.6. The remainder of the coefficients are denoted by "Subsequent DCT coefficients".

NOTE – In the case that Table B.14 is used for an intra block, the first coefficient shall be coded as specified in 7.2.1. Table B.14 shall therefore not be modified as the first coefficient that uses Table B.14 is the second coefficient in the block.

#### 7.2.2.3 Escape coding

Many possible combinations of run and level have no variable length code to represent them. In order to encode these statistically rare combinations an Escape coding method is used.

Table B.16 defines the escape coding method. The Escape VLC is followed by a 6-bit fixed length code giving "*run*". This is followed by a 12-bit fixed length code giving the values of "*signed_level*".

NOTE – Attention is drawn to the fact that the escape coding method used in this Specification is different to that used in ISO/IEC 11172-2.

#### 7.2.2.4 Summary

To summarise 7.2.2. The variable length decoding process shall be equivalent to the following. At the start of this process *n* shall take the value zero for non-intra blocks and one for intra blocks.

```
eob_not_read = 1;
while ( eob_not_read )
{
    <decode VLC, decode Escape coded coefficient if required>
    if ( <decoded VLC indicates End of block> ) {
        eob_not_read = 0;
        while ( n < 64) {
            QFS[n] = 0;
            n = n + 1;
        }
    } else {
        for ( m = 0; m < run; m++ ) {
            QFS[n] = 0;
            n = n + 1;
        }
        QFS[n] = signed_level
        n = n + 1;
    }
}
```

NOTE – *eob_not_read* and *m* are temporary variables that are not used elsewhere in this Specification.

### 7.3 Inverse scan

Let the data at the output of the variable length decoder be denoted by *QFS*[*n*]. *n* is in the range 0 to 63.

This subclause specifies the way in which the one-dimensional data, *QFS*[*n*], is converted into a two-dimensional array of coefficients denoted by *QF*[*v*][*u*]. *u* and *v* both lie in the range 0 to 7.

Two scan patterns are defined. The scan that shall be used shall be determined by alternate_scan which is encoded in the picture coding extension.

Figure 7-2 defines *scan*[alternate_scan][*v*][*u*] for the case that alternate_scan is zero. Figure 7-3 defines *scan*[alternate_scan][*v*][*u*] for the case that alternate_scan is one.

The inverse scan shall be any process equivalent to the following:

```
for (v = 0; v < 8; v++)
    for (u = 0; u < 8; u++)
        QF[v][u] = QFS[scan[alternate_scan][v][u]]
```

NOTE – The scan patterns defined here are often referred to as "zigzag scanning order".

| | | u | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 1 | 5 | 6 | 14 | 15 | 27 | 28 |
| 1 | 2 | 4 | 7 | 13 | 16 | 26 | 29 | 42 |
| 2 | 3 | 8 | 12 | 17 | 25 | 30 | 41 | 43 |
| 3 | 9 | 11 | 18 | 24 | 31 | 40 | 44 | 53 |
| 4 | 10 | 19 | 23 | 32 | 39 | 45 | 52 | 54 |
| 5 | 20 | 22 | 33 | 38 | 46 | 51 | 55 | 60 |
| 6 | 21 | 34 | 37 | 47 | 50 | 56 | 59 | 61 |
| v 7 | 35 | 36 | 48 | 49 | 57 | 58 | 62 | 63 |

**Figure 7-2 – Definition of *scan*[0][*v*][*u*]**

| | | u | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 4 | 6 | 20 | 22 | 36 | 38 | 52 |
| 1 | 1 | 5 | 7 | 21 | 23 | 37 | 39 | 53 |
| 2 | 2 | 8 | 19 | 24 | 34 | 40 | 50 | 54 |
| 3 | 3 | 9 | 18 | 25 | 35 | 41 | 51 | 55 |
| 4 | 10 | 17 | 26 | 30 | 42 | 46 | 56 | 60 |
| 5 | 11 | 16 | 27 | 31 | 43 | 47 | 57 | 61 |
| 6 | 12 | 15 | 28 | 32 | 44 | 48 | 58 | 62 |
| v 7 | 13 | 14 | 29 | 33 | 45 | 49 | 59 | 63 |

**Figure 7-3 – Definition of *scan*[1][*v*][*u*]**

### 7.3.1 Inverse scan for matrix download

When the quantisation matrices are downloaded they are encoded in the bitstream in a scan order that is converted into the two-dimensional matrix used in the inverse quantiser in an identical manner to that used for coefficients.

For matrix download the scan defined by Figure 7-2 (i.e. *scan*[0][*v*][*u*]) shall always be used.

Let $W[w][v][u]$ denote the weighting matrix in the inverse quantiser (see 7.4.2.1), and $W'[w][n]$ denote the matrix as it is encoded in the bitstream. The matrix download shall then be equivalent to the following:

```
for (v = 0; v < 8; v++)
    for (u = 0; u < 8; u++)
        W[w][v][u] = W'[w][scan[0][v][u]]
```

## 7.4 Inverse quantisation

The two-dimensional array of coefficients, $QF[v][u]$, is inverse quantised to produce the reconstructed DCT coefficients. This process is essentially a multiplication by the quantiser step size. The quantiser step size is modified by two mechanisms; a weighting matrix is used to modify the step size within a block and a scale factor is used in order that the step size can be modified at the cost of only a few bits (as compared to encoding an entire new weighting matrix).

Figure 7-4 illustrates the overall inverse quantisation process. After the appropriate inverse quantisation arithmetic the resulting coefficients, $F''[v][u]$, are saturated to yield $F'[v][u]$ and then a mismatch control operation is performed to give the final reconstructed DCT coefficients, $F[v][u]$.

NOTE – Attention is drawn to the fact that the method of achieving mismatch control in this Specification is different to that employed by ISO/IEC 11172-2.

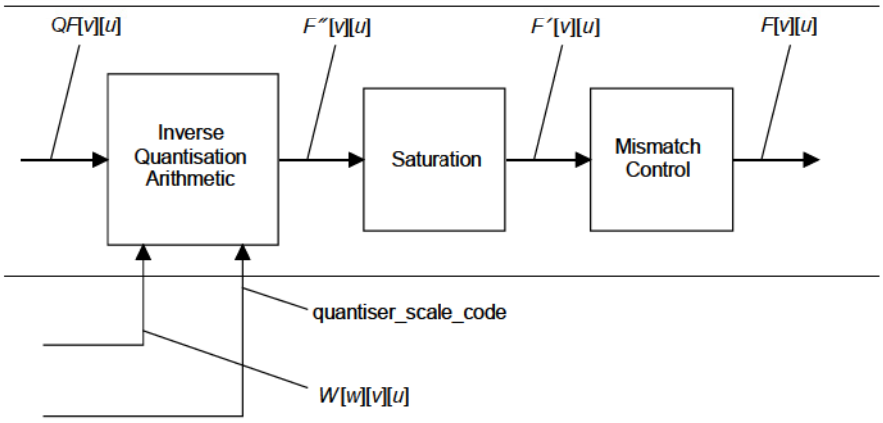



**Figure 7-4 – Inverse quantisation process**

### 7.4.1 Intra DC coefficient

The DC coefficients of intra coded blocks shall be inverse quantised in a different manner to all other coefficients.

In intra blocks $F''[0][0]$ shall be obtained by multiplying $QF[0][0]$ by a constant multiplier, *intra_dc_mult*, (constant in the sense that it is not modified by either the weighting matrix or the scale factor). The multiplier is related to the parameter intra_dc_precision that is encoded in the picture coding extension. Table 7-4 specifies the relation between intra_dc_precision and *intra_dc_mult*.

Thus; F"[0][0] = intra_dc_mult × QF[0][0]

**Table 7-4 – Relation between intra_dc_precision and *intra_dc_mult***

| intra_dc_precision | Bits of precision | *intra_dc_mult* |
|---|---|---|
| 0 | 8 | 8 |
| 1 | 9 | 4 |
| 2 | 10 | 2 |
| 3 | 11 | 1 |

### 7.4.2 Other coefficients

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause.

#### 7.4.2.1 Weighting matrices

When 4:2:0 data is used two weighting matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. When 4:2:2 or 4:4:4 data is used, four matrices are used allowing different matrices to be used for luminance and chrominance data. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix as explained in 6.2.3.2.

Let the weighting matrices be denoted by $W[w][v][u]$ where $w$ takes the values 0 to 3 indicating which of the matrices is being used. Table 7-5 summarises the rules governing the selection of $w$.

**Table 7-5 – Selection of *w***

| | 4:2:0 | | 4:2:2 and 4:4:4 | |
|---|---|---|---|---|
| | Luminance (cc = 0) | Chrominance (cc ≠ 0) | Luminance (cc = 0) | Chrominance (cc ≠ 0) |
| **intra blocks (macroblock_intra = 1)** | 0 | 0 | 0 | 2 |
| **non-intra blocks (macroblock_intra = 0)** | 1 | 1 | 1 | 3 |

#### 7.4.2.2 Quantiser scale factor

The quantisation scale factor is encoded as a 5-bit fixed length code, quantiser_scale_code. This indicates the appropriate *quantiser_scale* to apply in the inverse quantisation arithmetic.

q_scale_type (encoded in the picture coding extension) indicates which of two mappings between quantiser_scale_code and *quantiser_scale* shall apply. Table 7-6 shows the two mappings between quantiser_scale_code and *quantiser_scale*.

#### 7.4.2.3 Reconstruction formulae

The following equation specifies the arithmetic to reconstruct $F''[v][u]$ from $QF[v][u]$ (for all coefficients except intra DC coefficients).

$$F''[v][u] = ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantizer\_scale / 32$$

where:

$$k = \begin{cases} 0 & \text{intra blocks} \\ Sign(QF[v][u]) & \text{non - intra blocks} \end{cases}$$

NOTE – The above equation uses the "/" operator as defined in 4.1.

### 7.4.3 Saturation

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range [–2048:+2047]. Thus:

$$F'[v][u] = \begin{cases} 2047 & F''[v][u] > 2047 \\ F''[v][u] & -2048 \le F''[v][u] \le 2047 \\ -2048 & F''[v][u] < -2048 \end{cases}$$

### 7.4.4 Mismatch control

Mismatch control shall be performed by any process equivalent to the following. Firstly all of the reconstructed, saturated coefficients, $F'[v][u]$ in the block shall be summed. This value is then tested to determine whether it is odd or even. If the sum is even then a correction shall be made to just one coefficient; $F[7][7]$. Thus:

$$sum = \sum_{v=0}^{v<8} \sum_{u=0}^{u<8} F'[v][u]$$

$$F[v][u] = F'[v][u] \text{ for all u, v except } u = v = 7$$

$$F[7][7] = \begin{cases} F'[7][7] & \text{if sum is odd} \\ \begin{cases} F'[7][7] - 1 & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1 & \text{if } F'[7][7] \text{ is even} \end{cases} & \text{if sum is even} \end{cases}$$

NOTE 1 – It may be useful to note that the above correction for *F*[7][7] may simply be implemented by toggling the least significant bit of the twos complement representation of the coefficient. Also since only the "oddness" or "evenness" of the *sum* is of interest an exclusive OR (of just the least significant bit) may be used to calculate "*sum*".

NOTE 2 – Warning – Small non-zero inputs to the IDCT may result in zero output for compliant IDCTs. If this occurs in an encoder, mismatch may occur in some pictures in a decoder that uses a different compliant IDCT. An encoder should avoid this problem and may do so by checking the output of its own IDCT. It should ensure that it never inserts any non-zero coefficients into the bitstream when the block in question reconstructs to zero through its own IDCT function. If this action is not taken by the encoder, situations can arise where large and very visible mismatches between the state of the encoder and decoder occur.

**Table 7-6 – Relation between *quantiser_scale* and quantiser_scale_code**

| quantiser_scale_code | quantiser_scale[q_scale_type] | |
|---|---|---|
| | q_scale_type = 0 | q_scale_type = 1 |
| 0 | (Forbidden) | |
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 3 | 6 | 3 |
| 4 | 8 | 4 |
| 5 | 10 | 5 |
| 6 | 12 | 6 |
| 7 | 14 | 7 |
| 8 | 16 | 8 |
| 9 | 18 | 10 |
| 10 | 20 | 12 |
| 11 | 22 | 14 |
| 12 | 24 | 16 |
| 13 | 26 | 18 |
| 14 | 28 | 20 |
| 15 | 30 | 22 |
| 16 | 32 | 24 |
| 17 | 34 | 28 |
| 18 | 36 | 32 |
| 19 | 38 | 36 |
| 20 | 40 | 40 |
| 21 | 42 | 44 |
| 22 | 44 | 48 |
| 23 | 46 | 52 |
| 24 | 48 | 56 |
| 25 | 50 | 64 |
| 26 | 52 | 72 |
| 27 | 54 | 80 |
| 28 | 56 | 88 |
| 29 | 58 | 96 |
| 30 | 60 | 104 |
| 31 | 62 | 112 |

### 7.4.5 Summary

In summary the inverse quantisation process is any process numerically equivalent to:

```
for (v = 0; v < 8;v++) {
    for (u = 0; u < 8;u++) {
        if ( (u==0) && (v==0) && (macroblock_intra) ) {
            F''[v][u] = intra_dc_mult * QF[v][u];
        } else {
            if ( macroblock_intra ) {
                F''[v][u] = ( QF[v][u] * W[w][v][u] * quantiser_scale * 2 ) / 32;
            } else {
                F''[v][u] = ( ( ( QF[v][u] *  2 ) + Sign(QF[v][u]) ) * W[w][v][u]
                                                                    * quantiser_scale ) / 32;
            }
        }
    }
}

sum = 0;
    for (v = 0; v < 8;v++) {
        for (u = 0; u < 8;u++) {
            if ( F''[v][u] > 2047 ) {
                F'[v][u] = 2047;
            } else {
                if ( F''[v][u] < –2048 ) {
                    F'[v][u] = –2048;
                } else {
                    F'[v][u] = F''[v][u];
                }
            }
        sum = sum + F'[v][u];
        F[v][u] = F'[v][u];
        }
}

if ((sum & 1) == 0) {
    if ((F[7][7] & 1) ! = 0) {
        F[7][7] = F'[7][7] – 1;
    } else {
        F[7][7] = F'[7][7] + 1;
    }
}
```

## 7.5 Inverse DCT

Once the DCT coefficients, F[v][u] are reconstructed, an IDCT transform that conforms to the specifications of Annex A shall be applied to obtain the inverse transformed values f[y][x].

### 7.5.1 Non-coded blocks and skipped macroblocks

In a macroblock that is not skipped, if pattern_code[i] is one for a given block in the macroblock, then coefficient data is included in the bitstream for that block. This is decoded using as specified in the preceding clauses.

However, if pattern_code[i] is zero, or if the macroblock is skipped, then that block contains no coefficient data. The sample domain coefficients $f[y][x]$ for such a block shall all take the value zero.

## 7.6 Motion compensation

The motion compensation process forms predictions from previously decoded pictures which are combined with the coefficient data (from the output of the IDCT) in order to recover the final decoded samples. Figure 7-5 shows a simplified diagram of this process.

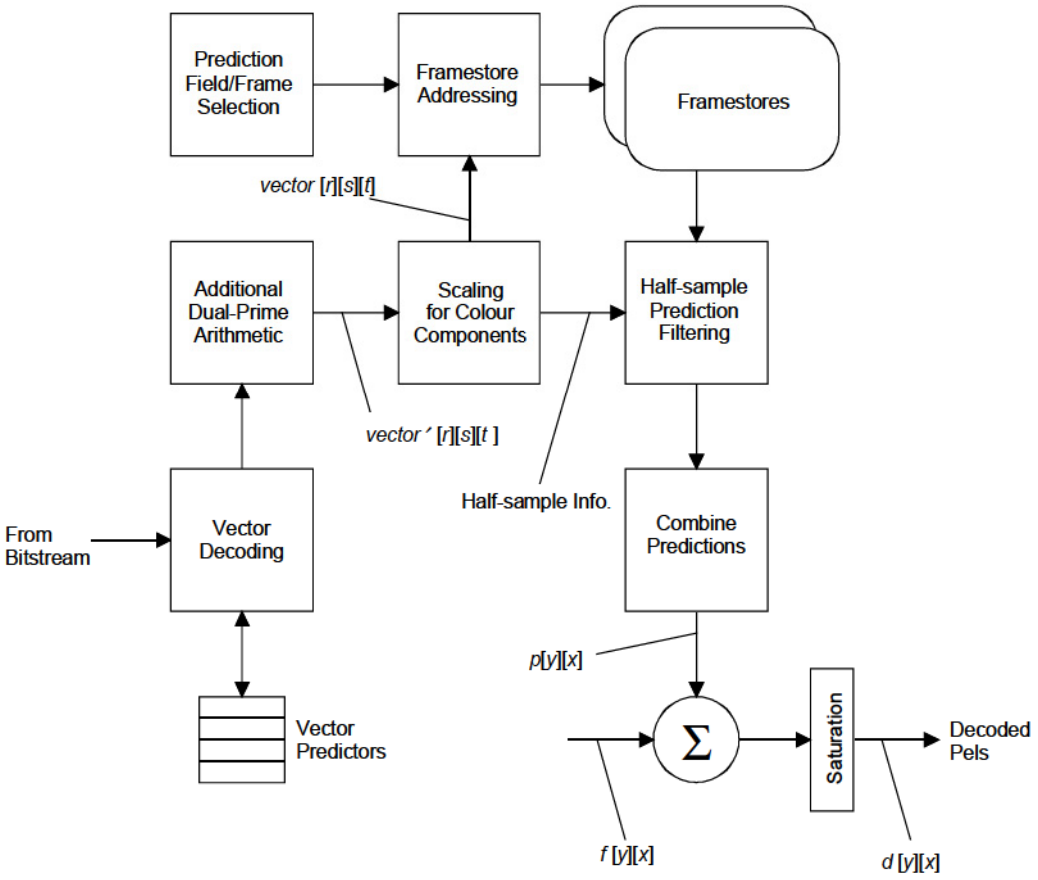



**Figure 7-5 – Simplified motion compensation process**

In general up to four separate predictions are formed for each block which are combined together to form the final prediction block p[y][x].

In the case of intra coded macroblocks no prediction is formed so that p[y][x] will be zero. The saturation shown in Figure 7-5 is still required in order to remove negative values from f[y][x]. Intra coded macroblocks may carry motion vectors known as "concealment motion vectors". Despite this no prediction is formed in the normal course of events. This motion vector information is intended for use in the case that bitstream errors preclude the decoding of coefficient information. The way in which a decoder shall use this information is not specified. The only requirement for these motion vectors is that they shall have the correct syntax for motion vectors. A description of the way in which these motion vectors may be used can be found in 7.6.3.9.

In the case where a block is not coded, either because the entire macroblock is skipped or the specific block is not coded there is no coefficient data. In this case f[y][x] is zero and the decoded samples are simply the prediction, p[y][x].

### 7.6.1 Prediction modes

There are two major classifications of the prediction mode:

- field prediction; and
- frame prediction.

In field prediction, predictions are made independently for each field by using data from one or more previously decoded fields. Frame prediction forms a prediction for the frame from one or more previously decoded frames. It must be understood that the fields and frames from which predictions are made may themselves have been decoded as either field pictures or frame pictures.

Within a field picture all predictions are field predictions. However, in a frame picture either field predictions or frame predictions may be used (selected on a macroblock-by macroblock basis).

In addition to the major classification of field or frame prediction two special prediction modes are used:

- *16 × 8 motion compensation* – In which two motion vectors are used for each macroblock. The first motion vector is used for the upper 16 × 8 region, the second for the lower 16 × 8 region. In the case of a bidirectionally predicted macroblock a total of four motion vectors will be used since there will be two for the forward prediction and two for the backward prediction. In this Specification 16 × 8 motion compensation shall only be used with field pictures.

- *Dual-prime* – In which only one motion vector is encoded (in its full format) in the bitstream together with a small differential motion vector. In the case of field pictures two motion vectors are then derived from this information. These are used to form predictions from two reference fields (one top, one bottom) which are averaged to form the final prediction. In the case of frame pictures this process is repeated for the two fields so that a total of four field predictions are made. This mode shall only be used in P-pictures where there are no B-pictures between the predicted and reference fields or frames.

### 7.6.2 Prediction field and frame selection

The selection of which fields and frames shall be used to form predictions shall be made as detailed in this clause.

#### 7.6.2.1 Field prediction

In P-pictures, the two reference fields from which predictions shall made are the most recently decoded reference top field and the most recently decoded reference bottom field. The simplest case illustrated in Figure 7-6 shall be used when predicting the first picture of a coded frame or when using field prediction within a frame-picture. In these cases the two reference fields are part of the same reconstructed frame.

NOTE 1 – The reference fields may themselves have been reconstructed from two field-pictures or a single frame-picture.

NOTE 2 –In the case of predicting a field picture, the field being predicted may be either the top field or the bottom field.

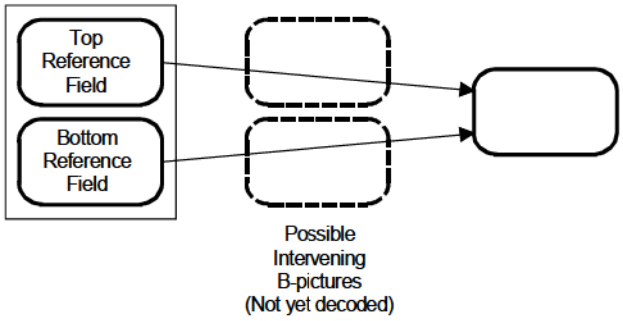



**Figure 7-6 – Prediction of the first field or field prediction in a frame-picture**

The case when predicting the second field picture of a coded frame is more complicated because the two most recently decoded reference fields shall be used, and in this case, the most recent reference field was obtained from decoding the first field picture of the coded frame. Figure 7-7 illustrates the situation when this second picture is the bottom field. Figure 7-8 illustrates the situation when this second picture is the top field.

NOTE 3 – The earlier reference field may itself have been reconstructed by decoding a field picture or a frame picture.

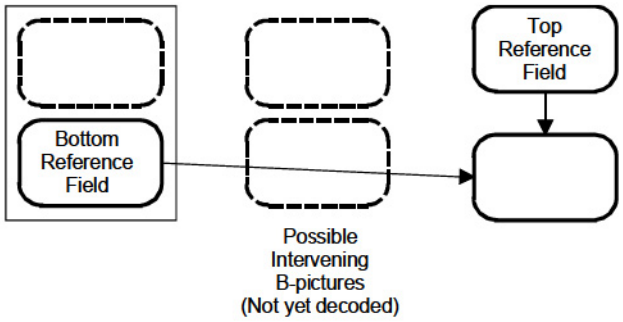



**Figure 7-7 – Prediction of the second field-picture when it is the bottom field**

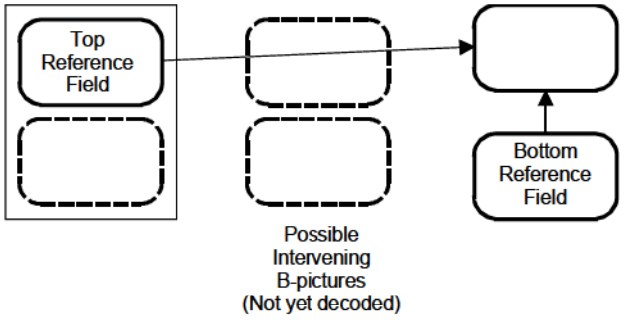



**Figure 7-8 – Prediction of the second field-picture when it is the top field**

Field prediction in B-pictures shall be made from the two fields of the two most recently reconstructed reference frames. Figure 7-9 illustrates this situation.

NOTE 4 – The reference frames may themselves have been reconstructed from two coded field-pictures or a single coded frame-picture.

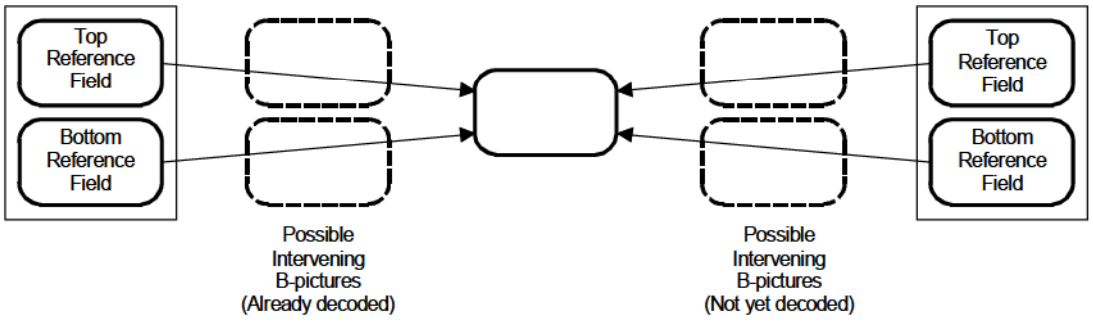



**Figure 7-9 – Field-prediction of B-field pictures or B-frame pictures**

#### 7.6.2.2 Frame prediction

In P-pictures prediction shall be made from the most recently reconstructed reference frame. This is illustrated in Figure 7-10.

NOTE 1 – The reference frame may itself have been coded as two field pictures or a single frame picture.

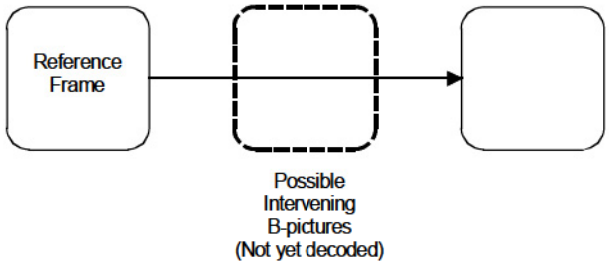



**Figure 7-10 – Frame-prediction for I-pictures and P-pictures**

Similarly frame prediction in B-pictures shall be made from the two most recently reconstructed reference frames as illustrated in Figure 7-11.

NOTE 2 – The reference frames themselves may each have been coded as either two field pictures or a single frame picture.

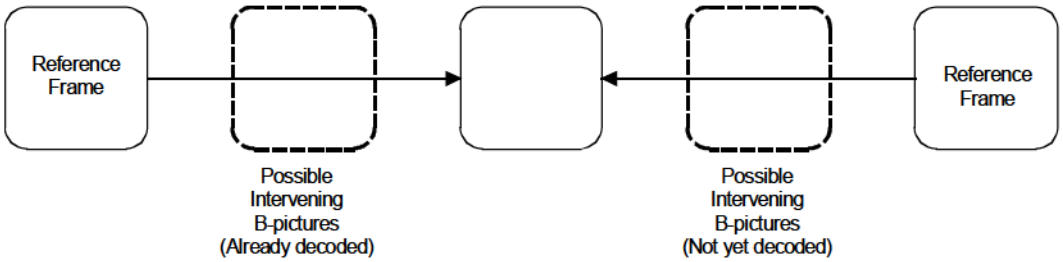



**Figure 7-11 – Frame-prediction for B-pictures**

#### 7.6.3 Motion vectors

Motion vectors are coded differentially with respect to previously decoded motion vectors in order to reduce the number of bits required to represent them. In order to decode the motion vectors the decoder shall maintain four motion vector predictors (each with a horizontal and vertical component) denoted $PMV[r][s][t]$. For each prediction, a motion vector, $vector'[r][s][t]$ is first derived. This is then scaled depending on the sampling structure (4:2:0, 4:2:2 or 4:4:4) to give a motion vector, $vector[r][s][t]$, for each colour component. The meanings associated with the dimensions in this array are defined in Table 7-7.

**Table 7-7 – Meaning of indices in $PMV[r][s][t]$, $vector[r][s][t]$ and $vector'[r][s][t]$**

| | 0 | 1 |
|---|---|---|
| $r$ | First motion vector in Macroblock | Second motion vector in Macroblock |
| $s$ | Forward motion Vector | Backwards motion Vector |
| $t$ | Horizontal Component | Vertical Component |
| NOTE – $r$ also takes the values 2 and 3 for derived motion vectors used with dual-prime prediction. Since these motion vectors are derived they do not themselves have motion vector predictors. | | |

#### 7.6.3.1 Decoding the motion vectors

Each motion vector component, *vector'*[*r*][*s*][*t*], shall be calculated by any process that is equivalent to the following one. Note that the motion vector predictors shall also be updated by this process.

```
r_size = f_code[s][t] – 1
f = 1 << r_size
high = ( 16 * f ) – 1;
low = ( (–16) * f );
range = ( 32 * f );

if ( (f == 1) || (motion_code[r][s][t] == 0) )
    delta = motion_code[r][s][t] ;
else {
    delta = ( ( Abs(motion_code[r][s][t]) – 1 ) * f ) + motion_residual[r][s][t] + 1;
    if (motion_code[r][s][t] < 0)
        delta = – delta;
}

prediction = PMV[r][s][t];
if ( (mv_format == "field") && (t==1) && (picture_structure == "Frame picture") )
    prediction = PMV[r][s][t] DIV 2;

vector'[r][s][t] = prediction + delta;
if (vector'[r][s][t] < low )
    vector'[r][s][t] = vector'[r][s][t] + range;
if (vector'[r][s][t] > high)
    vector'[r][s][t] = vector'[r][s][t] – range;

if ( (mv_format == "field") && (t == 1) && (picture_structure == "Frame picture") )
    PMV[r][s][t] = vector'[r][s][t] * 2;
else
    PMV[r][s][t] = vector'[r][s][t];
```

The parameters in the bitstream shall be such that the reconstructed differential motion vector, *delta*, shall lie in the range [*low*:*high*]. In addition the reconstructed motion vector, *vector'*[*r*][*s*][*t*], and the updated value of the motion vector predictor *PMV*[*r*][*s*][*t*], shall also lie in the range [*low* : *high*].

*r_size*, *f*, *delta*, *high*, *low* and *range* are temporary variables that are not used in the remainder of this Specification.

motion_code[r][s][t] and motion_residual[r][s][t] are fields recovered from the bitstream. mv_format is recovered from the bitstream using Tables 6-17 and 6-18.

*r*, *s* and *t* specify the particular motion vector component being processed as identified in Table 7-7.

*vector'*[*r*][*s*][*t*] is the final reconstructed motion vector for the luminance component of the macroblock.

The vector'[r][s][t] value considered in this subclause is the one obtained from the pseudo code above. In case of dual-prime, this restriction that vector'[r][s][t] shall be in the range [low:high] does not apply to the scaled motion vectors vector'[2:3][0][0:1] defined in 7.6.3.6. Other restrictions on motion vectors, including scaled dual-prime motion vectors are specified in 7.6.3.8 and 8.3.

#### 7.6.3.2 Motion vector restrictions

In frame pictures, the vertical component of field motion vectors shall be restricted so that they only cover half the range that is supported by the f_code that relates to those motion vectors. This restriction ensures that the motion vector predictors will always have values that are appropriate for decoding subsequent frame motion vectors. Table 7-8 summarises the size of motion vectors that may be coded as a function of f_code.

#### 7.6.3.3 Updating motion vector predictors

Once all of the motion vectors present in the macroblock have been decoded using the process defined in the previous clause it is sometimes necessary to update other motion vector predictors. This is because in some prediction modes fewer than the maximum possible number of motion vectors are used. The remainder of the predictors that might be used in the picture must retain "sensible" values in case they are subsequently used.

**Table 7-8 – Allowable motion vector range as a function of *f_code*[*s*][*t*]**

| *f_code*[*s*][*t*] | Vertical components (*t* == 1) of field vectors in frame pictures | All other cases |
|---|---|---|
| 0 | (Forbidden) | |
| 1 | [–4: +3.5] | [–8: +7.5] |
| 2 | [–8: +7.5] | [–16: +15.5] |
| 3 | [–16: +15.5] | [–32: +31.5] |
| 4 | [–32: +31.5] | [–64: +63.5] |
| 5 | [–64: +63.5] | [–128: +127.5] |
| 6 | [–128: +127.5] | [–256: +255.5] |
| 7 | [–256: +255.5] | [–512: +511.5] |
| 8 | [–512: +511.5] | [–1024: +1023.5] |
| 9 | [–1024: +1023.5] | [–2048: +2047.5] |
| 10-14 | (Reserved) | |
| 15 | (Used when a particular *f_code*[*s*][*t*] will not be used) | |

The motion vector predictors shall be updated as specified in Table 7-9 and 7-10. The rules for updating motion vector predictors in the case of skipped macroblocks are specified in 7.6.6.

NOTE – It is possible for an implementation to optimise the updating (and resetting) of motion vector predictors depending on the picture type. For example in a P-picture the predictors for backwards motion vectors are unused and need not be maintained.

**Table 7-9 – Updating of motion vector predictors in frame pictures**

| **frame_motion_type** | **macroblock_motion_** | | **macroblock_intra** | Predictors to Update |
|---|---|---|---|---|
| | **forward** | **backward** | | |
| Frame-based[a)] | – | – | 1 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0][b)] |
| Frame-based | 1 | 1 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0]<br>*PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Frame-based | 1 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |
| Frame-based | 0 | 1 | 0 | *PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Frame-based[a)] | 0 | 0 | 0 | *PMV*[*r*][*s*][*t*] = 0[c)] |
| Field-based | 1 | 1 | 0 | (None) |
| Field-based | 1 | 0 | 0 | (None) |
| Field-based | 0 | 1 | 0 | (None) |
| Dual prime | 1 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |

a) **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based.

b) If **concealment_motion_vectors** is zero then *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*).

c) (Only occurs in P-picture) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.6.3.4.

NOTE – *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that:

*PMV*[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

**Table 7-10 – Updating of motion vector predictors in field pictures**

| frame_motion_type | macroblock_motion_ | | macroblock_intra | Predictors to Update |
|---|---|---|---|---|
| | forward | backward | | |
| Field-based[a)] | – | – | 1 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0][b)] |
| Field-based | 1 | 1 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0]<br>*PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Field-based | 1 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |
| Field-based | 0 | 1 | 0 | *PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Field-based[a)] | 0 | 0 | 0 | *PMV*[*r*][*s*][*t*] = 0[c)] |
| 16 × 8 MC | 1 | 1 | 0 | (None) |
| 16 × 8 MC | 1 | 0 | 0 | (None) |
| 16 × 8 MC | 0 | 1 | 0 | (None) |
| Dual prime | 1 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |

a) **frame_motion_type** is not present in the bitstream but is assumed to be Field-based.

b) If **concealment_motion_vectors** is zero then *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*).

c) (Only occurs in P-picture) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.6.3.4.

NOTE – *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that:

*PMV*[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

#### 7.6.3.4 Resetting motion vector predictors

All motion vector predictors shall be reset to zero in the following cases:

- At the start of each slice.
- Whenever an intra macroblock is decoded which has no concealment motion vectors.
- In a P-picture when a non-intra macroblock is decoded in which macroblock_motion_forward is zero.
- In a P-picture when a macroblock is skipped.

#### 7.6.3.5 Prediction in P-pictures

In P-pictures, in the case that *macroblock_motion_forward* is zero and *macroblock_intra* is also zero no motion vectors are encoded for the macroblock yet a prediction must be formed. If this occurs in a P-field picture the following apply;

- the prediction type shall be "Field-based";
- the (field) motion vector shall be zero (0;0);
- the motion vector predictors shall be reset to zero;
- predictions shall be made from the field of the same parity as the field being predicted.

If this occurs in a P-frame picture the following apply:

- the prediction type shall be "Frame-based";
- the (frame) motion vector shall be zero (0;0);
- the motion vector predictors shall be reset to zero.

In the case that a P-field picture is used as the second field of a frame in which the first field is an I-field picture a series of semantic restrictions apply. These ensure that prediction is only made from the I field picture. These restrictions are:

- There shall be no macroblocks that are coded with *macroblock_motion_forward* zero and *macroblock_intra* zero.
- Dual prime prediction shall not be used.
- Field prediction in which **motion_vertical_field_select** indicates the same parity as the field being predicted shall not be used.
- There shall be no skipped macroblocks.

#### 7.6.3.6 Dual prime additional arithmetic

In dual prime prediction one field motion vector (*vector*[0][0][1:0]) will have been decoded by the process already described. This represents the motion vector used to form predictions from the reference field (or reference fields in a frame picture) of the same parity as the prediction being formed. Here the word "parity" is used to differentiate the two fields. The top field has parity zero, the bottom field has parity one.

In order to form a motion vector for the opposite parity (*vector*[*r*][0][1:0]) the existing motion vector is scaled to reflect the different temporal distance between the fields. A correction is made to the vertical component (to reflect the vertical shift between the lines of top field and bottom field) and then a small differential motion vector is added. This process is illustrated in Figure 7-12 which shows the situation for a frame picture.

*dmvector*[0] is the horizontal component of the differential motion vector and *dmvector*[1] the vertical component. The two components of the differential motion vector shall be decoded directly using Table B.11 and shall take only one of the values –1, 0, + 1.

*m*[*parity_ref*][*parity_pred*] is the field distance between the predicted field and the reference field as defined in Table 7-11. "*parity_ref*" is the parity of the reference field for which the new motion vector is being computed. "*parity_pred*" is the parity of the field that shall be predicted.

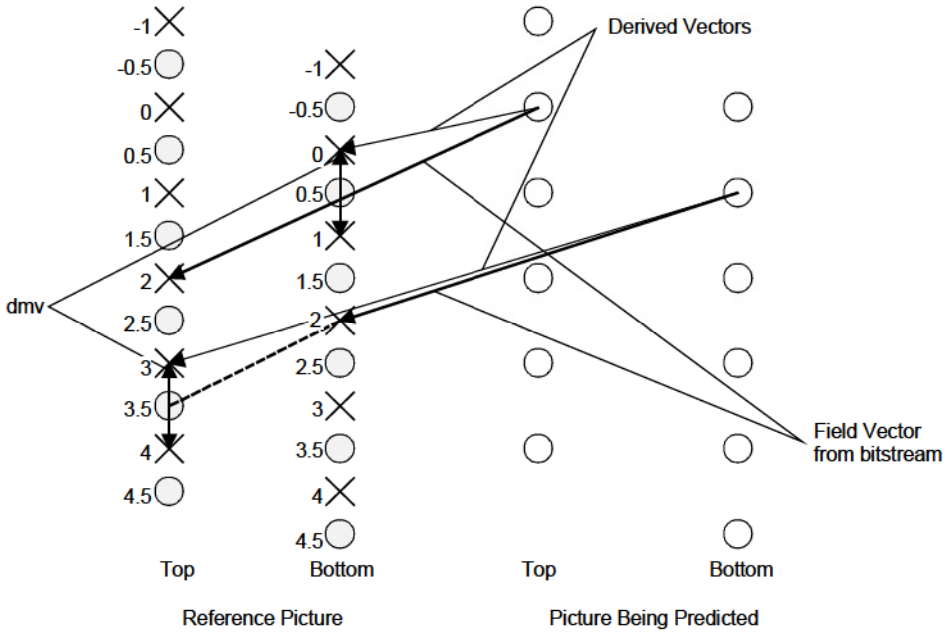



**Figure 7-12 – Scaling of motion vectors for dual prime prediction**

**Table 7-11 – Definition of *m*[*parity_ref*][*parity_pred*]**

| picture_structure | top_field_first | m[*parity_ref*][*parity_pred*] | |
|---|---|---|---|
| | | m[1][0] | m[0][1] |
| 11 (Frame) | 1 | 1 | 3 |
| 11 (Frame) | 0 | 3 | 1 |
| 01 (Top Field) | – | 1 | – |
| 10 (Bottom Field) | – | – | 1 |

*e*[*parity_ref*][*parity_pred*] is the adjustment necessary to reflect the vertical shift between the lines of top field and bottom field as defined in Table 7-12.

**Table 7-12 – Definition of *e*[*parity_ref*][*parity_pred*]**

| *parity_ref* | *parity_pred* | *e[parity_ref][parity_pred]* |
|---|---|---|
| 0 | 1 | +1 |
| 1 | 0 | –1 |

The motion vector (or motion vectors) used for predictions of opposite parity shall be computed as follows:

*vector*'[*r*][0][0] = ((*vector*'[0][0][0] * *m*[*parity_ref*][*parity_pred*])//2) + *dmvector*[0];

*vector*'[*r*][0][1] = ((*vector*'[0][0][1] * *m*[*parity_ref*][*parity_pred*])//2)
+ *e*[*parity_ref*][*parity_pred*] + *dmvector*[1];

In the case of field pictures only one such motion vector is required and here r = 2. Thus, the (encoded) motion vector used for the same parity prediction is *vector*'[0][0][1:0] and the motion vector used for the opposite parity prediction is *vector*'[2][0][1:0].

In the case of frame pictures two such motion vectors are required. Both fields use the encoded motion vector (*vector*'[0][0][1:0]) for predictions of the same parity. The top field shall use *vector*'[2][0][1:0] for opposite parity prediction and the bottom field shall use *vector*'[3][0][1:0] for opposite parity prediction.

#### 7.6.3.7 Motion vectors for chrominance components

The motion vectors calculated in the previous clauses refer to the luminance component where:

*vector*[*r*][*s*][*t*] = *vector*'[*r*][*s*][*t*] (for all r, s and t)

For each of the two chrominance components the motion vectors shall be scaled as follows:

4:2:0 Both the horizontal and vertical components of the motion vector are scaled by dividing by two:

*vector*[*r*][*s*][0] = *vector*'[*r*][*s*][0] / 2;

*vector*[*r*][*s*][1] = *vector*'[*r*][*s*][1] / 2;

4:2:2 The horizontal component of the motion vector is scaled by dividing by two, the vertical component is not altered:

*vector*[*r*][*s*][0] = *vector*'[*r*][*s*][0] / 2;

*vector*[*r*][*s*][1] = *vector*'[r][s][1];

4:4:4 The motion vector is unmodified:

*vector*[*r*][*s*][0] = *vector*'[*r*][*s*][0];

*vector*[*r*][*s*][1] = *vector*'[*r*][*s*][1];

#### 7.6.3.8 Semantic restrictions concerning predictions

It is a requirement on the bitstream that it shall only demand of a decoder that predictions shall be made from slices actually encoded in a reference frame or reference field. This rule applies even for skipped macroblocks and macroblocks in P-pictures in which a zero motion vector is assumed (as explained in 7.6.3.5).

NOTE – As explained in 6.1.2 it is, in general, not necessary for the slices to cover the entire picture. However, in many defined levels of defined profiles the "restricted slice structure" is used in which case the slices do cover the entire picture. In this case the semantic rule may be more simply stated: "it is a restriction on the bitstream that reconstructed motion vectors shall not refer to samples outside the boundary of the coded picture."

#### 7.6.3.9 Concealment motion vectors

Concealment motion vectors are motion vectors that may be carried by intra macroblocks for the purpose of concealing errors if data errors preclude decoding the coefficient data. A concealment motion vector shall be present for all intra macroblocks if (and only if) concealment_motion_vectors (in the picture_coding_extension() ) has the value one.

In the normal course of events no prediction shall be formed for such macroblocks (as would be expected since macroblock_intra = 1). This Specification does not specify how error recovery shall be performed. However it is a recommendation that concealment motion vectors are suitable for use by a decoder that performs concealment by forming predictions as if field_motion_type and frame_motion_type (from which the prediction type is derived) have the following values:

- In a field picture: field_motion_type = "Field-based";
- In a frame picture: frame_motion_type = "Frame-based".

NOTE – If concealment is used in an I-picture then the decoder should perform prediction in a similar way to a P-picture.

Concealment motion vectors are intended for use in the case that a data error results in information being lost. There is therefore little point in encoding the concealment motion vector in the macroblock for which it is intended to be used since if the data error results in the need for error recovery it is very likely that the concealment motion vector itself would be lost or corrupted. As a result the following semantic rules are appropriate:

- For all macroblocks except those in the bottom row of macroblocks concealment motion vectors should be appropriate for use in the macroblock that lies vertically below the macroblock in which the motion vector occurs.
- When the motion vector is used with respect to the macroblock identified in the previous rule a decoder must assume that the motion vector may refer to samples outside of the slices encoded in the reference frame or reference field.
- For all macroblocks in the bottom row of macroblocks the reconstructed concealment motion vectors will not be used. Therefore the motion vector (0;0) may be used to reduce unnecessary overhead.

### 7.6.4 Forming predictions

Predictions are formed by reading prediction samples from the reference fields or frames. A given sample is predicted by reading the corresponding sample in the reference field or frame offset by the motion vector.

A positive value of the horizontal component of a motion vector indicates that the prediction is made from samples (in the reference field/frame) that lie to the right of the samples being predicted.

A positive value of the vertical component of a motion vector indicates that the prediction is made from samples (in the reference field/frame) that lie below the samples being predicted.

All motion vectors are specified to an accuracy of one half sample. Thus, if a component of the motion vector is odd, the samples will be read from mid-way between the actual samples in the reference field/frame. These half-samples are calculated by simple linear interpolation from the actual samples.

In the case of field-based predictions it is necessary to determine which of the two available fields to use to form the prediction. In the case of dual-prime this is specified in that a motion vector is derived for both of the fields and a prediction is formed from each. In the case of field-based prediction and 16 × 8 MC an additional bit, motion_vertical_field_select, is encoded to indicate which field to use.

If motion_vertical_field_select is zero, then the prediction is taken from the top reference field.

If motion_vertical_field_select is one, then the prediction is taken from the bottom reference field.

For each prediction block the integer sample motion vectors int_vec[t] and the half sample flags half_flag[t] shall be formed as follows;

```
for (t = 0; t< 2; t++) {
    int_vec[t] = vector[r][s][t] DIV 2;
    if ((vector[r][s][t] - (2 * int_vec[t]) != 0)
        half_flag[t] = 1;
    else
        half_flag[t] = 0;
}
```

Then for each sample in the prediction block the samples are read and the half sample prediction applied as follows;

```
if ( (! half_flag[0] )&& (! half_flag[1]) )
    pel_pred[y][x] = pel_ref[y + int_vec[1]][x + int_vec[0]];

if ( (! half_flag[0] )&& half_flag[1] )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]+1][x + int_vec[0]] ) // 2;
```

```
if ( half_flag[0]&& (! half_flag[1]) )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]][x + int_vec[0]+1] ) // 2;
if ( half_flag[0]&& half_flag[1] )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]][x + int_vec[0]+1] +
                           pel_ref[y + int_vec[1]+1][x + int_vec[0]] +
                               pel_ref[y + int_vec[1]+1][x + int_vec[0]+1] ) // 4;
```

where *pel_pred*[*y*][*x*] is the prediction sample being formed and *pel_ref*[*y*][*x*] are samples in the reference field or frame.

### 7.6.5 Motion vector selection

Table 7-13 shows the prediction modes used in field pictures and Table 7-14 shows the predictions used in frame pictures. In each table the motion vectors that are present in the bitstream are listed in the order in which they appear in the bitstream.

**Table 7-13 – Predictions and motion vectors in field pictures**

| field_ motion_ type | macroblock_motion_- forward | macroblock_motion_- backward | macro- block_- intra | Motion vector | Prediction formed for |
|---|---|---|---|---|---|
| Field-based[a] | – | – | 1 | *vector*[0][0][1:0][b] | None (motion vector is for concealment) |
| Field-based | 1 | 1 | 0 | *vector*[0][0][1:0]<br>*vector*[0][1][1:0] | Whole field, forward<br>Whole field, backward |
| Field-based | 1 | 0 | 0 | *vector*[0][0][1:0] | Whole field, forward |
| Field-based | 0 | 1 | 0 | *vector*[0][1][1:0] | Whole field, backward |
| Field-based[a] | 0 | 0 | 0 | *vector*[0][0][1:0][c] [d] | Whole field, forward |
| 16 × 8 MC | 1 | 1 | 0 | *vector*[0][0][1:0]<br>*vector*[1][0][1:0]<br>*vector*[0][1][1:0]<br>*vector*[1][1][1:0] | Upper 16 × 8 field, forward<br>Lower 16 × 8 field, forward<br>Upper 16 × 8 field, backward<br>Lower 16 × 8 field, backward |
| 16 × 8 MC | 1 | 0 | 0 | *vector*[0][0][1:0]<br>*vector*[1][0][1:0] | Upper 16 × 8 field, forward<br>Lower 16 × 8 field, forward |
| 16 × 8 MC | 0 | 1 | 0 | *vector*[0][1][1:0]<br>*vector*[1][1][1:0] | Upper 16 × 8 field, backward<br>Lower 16 × 8 field, backward |
| Dual prime | 1 | 0 | 0 | *vector*[0][0][1:0]<br>*vector*[2][0][1:0][c] [e] | Whole field, from same parity, forward<br>Whole field, from opposite parity, forward |

a) **field_motion_type** is not present in the bitstream but is assumed to be Field-based.

b) The motion vector is only present if **concealment_motion_vectors** is one.

c) These motion vectors are not present in the bitstream.

d) The motion vector is taken to be (0; 0) as explained in 7.6.3.5.

e) These motion vectors are derived from *vector*[0][0][1:0] as described in 7.6.3.6.

NOTE – Motion vectors are listed in the order they appear in the bitstream.

**Table 7-14 – Predictions and motion vectors in frame pictures**

| frame_motion_type | macroblock_motion_ forward | macroblock_motion_ backward | macroblock_intra | Motion vector | Prediction formed for |
|---|---|---|---|---|---|
| Frame-based[a] | – | – | 1 | *vector*'[0][0][1:0][b] | None (motion vector is for concealment) |
| Frame-based | 1 | 1 | 0 | *vector*'[0][0][1:0] | Frame, forward |
| | | | | *vector*'[0][1][1:0] | Frame, backward |
| Frame-based | 1 | 0 | 0 | *vector*'[0][0][1:0] | Frame, forward |
| Frame-based | 0 | 1 | 0 | *vector*'[0][1][1:0] | Frame, backward |
| Frame-based[a] | 0 | 0 | 0 | *vector*'[0][0][1:0][c] [d] | Frame, forward |
| Field-based | 1 | 1 | 0 | *vector*'[0][0][1:0] | Top field, forward |
| | | | | *vector*'[1][0][1:0] | Bottom field, forward |
| | | | | *vector*'[0][1][1:0] | Top field, backward |
| | | | | *vector*'[1][1][1:0] | Bottom field, backward |
| Field-based | 1 | 0 | 0 | *vector*'[0][0][1:0] | Top field, forward |
| | | | | *vector*'[1][0][1:0] | Bottom field, forward |
| Field-based | 0 | 1 | 0 | *vector*'[0][1][1:0] | Top field, backward |
| | | | | *vector*'[1][1][1:0] | Bottom field, backward |
| Dual prime | 1 | 0 | 0 | *vector*'[0][0][1:0] | Top field, from same parity, forward |
| | | | | *vector*'[0][0][1:0] | Bottom field, from same parity, forward |
| | | | | *vector*'[2][0][1:0][c] [e] | Top field, from opposite parity, forward |
| | | | | *vector*'[3][0][1:0][c] [e] | Bottom field, from opposite parity, forward |

a) **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based.

b) The motion vector is only present if **concealment_motion_vectors** is one.

c) These motion vectors are not present in the bitstream.

d) The motion vector is taken to be (0; 0) as explained in 7.6.3.5.

e) These motion vectors are derived from *vector*'[0][0][1:0] as described in 7.6.3.6.

NOTE – Motion vectors are listed in the order they appear in the bitstream.

### 7.6.6 Skipped macroblocks

A skipped macroblock is a macroblock for which no data is encoded, that is part of a coded slice. Except at the start of a slice, if the number (macroblock_address - previous_macroblock_address - 1) is larger than zero, then this number indicates the number of macroblocks that have been skipped. The decoder shall form a prediction for skipped macroblocks which shall then be used as the final decoded sample values.

The handling of skipped macroblocks is different between P-pictures and B-pictures. In addition, the process differs between field pictures and frame pictures.

There shall be no skipped macroblocks in I-pictures except when either:

- picture_spatial_scalable_extension() follows the picture_header() of the current picture; or
- sequence_scalable_extension() is present in the bitstream and scalable_mode = "SNR scalability".

#### 7.6.6.1 P field picture

- the prediction shall be made as if field_motion_type is "Field-based";
- the prediction shall be made from the field of the same parity as the field being predicted;
- motion vector predictors shall be reset to zero;
- the motion vector shall be zero.

#### 7.6.6.2 P frame picture

- the prediction shall be made as if frame_motion_type is "Frame-based";
- motion vector predictors shall be reset to zero;
- the motion vector shall be zero.

#### 7.6.6.3 B field picture

- the prediction shall be made as if field_motion_type is "Field-based";
- the prediction shall be made from the field of the same parity as the field being predicted;
- the direction of the prediction forward/backward/bi-directional shall be the same as the previous macroblock;
- motion vector predictors are unaffected;
- the motion vectors are taken from the appropriate motion vector predictors. Scaling of the motion vectors for colour components shall be performed as described in 7.6.3.7.

#### 7.6.6.4 B frame picture

- the prediction shall be made as if frame_motion_type is "Frame-based";
- the direction of the prediction forward/backward/bi-directional shall be the same as the previous macroblock;
- motion vector predictors are unaffected;
- the motion vectors are taken directly from the appropriate motion vector predictors. Scaling of the motion vectors for colour components shall be performed as described in 7.6.3.7.

### 7.6.7 Combining predictions

The final stage is to combine the various predictions together in order to form the final prediction blocks.

It is also necessary to organise the data into blocks that are either field organised or frame organised in order to be added directly to the decoded coefficients.

The transform data is either field organised or frame organised as specified by *dct_type*.

#### 7.6.7.1 Simple frame predictions

In the case of simple frame predictions the only further processing that may be required is to average forward and backward predictions in B-pictures. If *pel_pred_forward*[*y*][*x*] is the forwards prediction sample and *pel_pred_backward*[*y*][*x*] is the corresponding backward prediction, then the final prediction sample shall be formed as:

$$pel\_pred[y][x] = (pel\_pred\_forward[y][x] + pel\_pred\_backward[y][x])//2$$

The predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 8 lines, 8 samples by 16 lines and 16 samples by 16 lines respectively.

#### 7.6.7.2 Simple field predictions

In the case of simple field predictions (i.e. neither 16 × 8 or dual prime) the only further processing that may be required is to average forward and backward predictions in B-pictures. This shall be performed as specified for "Frame predictions" in the previous subclause.

In the case of simple field prediction in a frame picture the predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats for each field shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

In the case of simple field prediction in a field picture the predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats for each field shall be of size 8 samples by 8 lines, 8 samples by 16 lines and 16 samples by 16 lines respectively.

#### 7.6.7.3 16 × 8 Motion compensation

In this prediction mode separate predictions are formed for the upper 16 × 8 region of the macroblock and the lower 16 × 8 region of the macroblock.

The predictions for chrominance components, for each 16 × 8 region, of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

#### 7.6.7.4 Dual prime

In dual prime mode two predictions are formed for each field in an analogous manner to the backward and forward predictions in B-pictures. If *pel_pred_same_parity*[*y*][*x*] is the prediction sample from the same parity field and *pel_pred_opposite_parity*[*y*][*x*] is the corresponding sample from the opposite parity field then the final prediction sample shall be formed as:

$$pel\_pred[y][x] = (pel\_pred\_same\_parity[y][x] + pel\_pred\_opposite\_parity[y][x])//2;$$

In the case of dual prime prediction in a frame picture, the predictions for chrominance components of each field of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

In the case of dual prime prediction in a field picture, the predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 8 lines, 8 samples by 16 lines and 16 samples by 16 lines respectively.

### 7.6.8 Adding prediction and coefficient data

The prediction blocks have been formed and reorganised into blocks of prediction samples $p[y][x]$ which match the field/frame structure used by the transform data blocks.

The transform data $f[y][x]$ shall be added to the prediction data and saturated to form the final decoded samples $d[y][x]$ as follows:

```
for (y = 0; y < 8; y++) {
    for (x = 0; x < 8; x++) {
        d[y][x] = f[y][x] + p[y][x];
        if (d[y][x] <  0) d[y][x] = 0;
        if (d[y][x] > 255) d[y][x] = 255;
    }
}
```

## 7.7 Spatial scalability

This subclause specifies the additional decoding process required for the spatial scalable extensions.

Both the lower layer and the enhancement layer shall use the "restricted slice structure" (no gaps between slices).

Figure 7-13 is a diagram of the video decoding process with spatial scalability. The diagram is simplified for clarity.

### 7.7.1 Higher syntactic structures

In general, the base layer of a spatial scalable hierarchy can conform to any coding standard including Recommendation H.261, ISO/IEC 11172-2 and this Specification. Note however, that within this Specification the decodability of a spatial scalable hierarchy is only considered in the case that the base layer conforms to this Specification or ISO/IEC 11172-2.

Due to the "loose coupling" of layers only one syntactic restriction is needed in the enhancement layer if both lower and enhancement layer are interlaced. In that case picture_structure has to take the same value as in the reference frame used for prediction from the lower layer. See 7.7.3.1 for how to identify this reference frame.

### 7.7.2 Prediction in the enhancement layer

A motion compensated temporal prediction is made from reference frames in the enhancement layer as described in 7.6. In addition, a spatial prediction is formed from the lower layer decoded frame ($d_{lower}[y][x]$), as described in 7.7.3. These predictions are selected individually or combined to form the actual prediction.

In general, up to four separate predictions are formed for each macroblock which are combined together to form the final prediction macroblock p[y][x].

In the case that a macroblock is not coded, either because the entire macroblock is skipped or the specific macroblock is not coded, there is no coefficient data. In this case f[y][x] is zero and the decoded samples are simply the prediction, p[y][x].

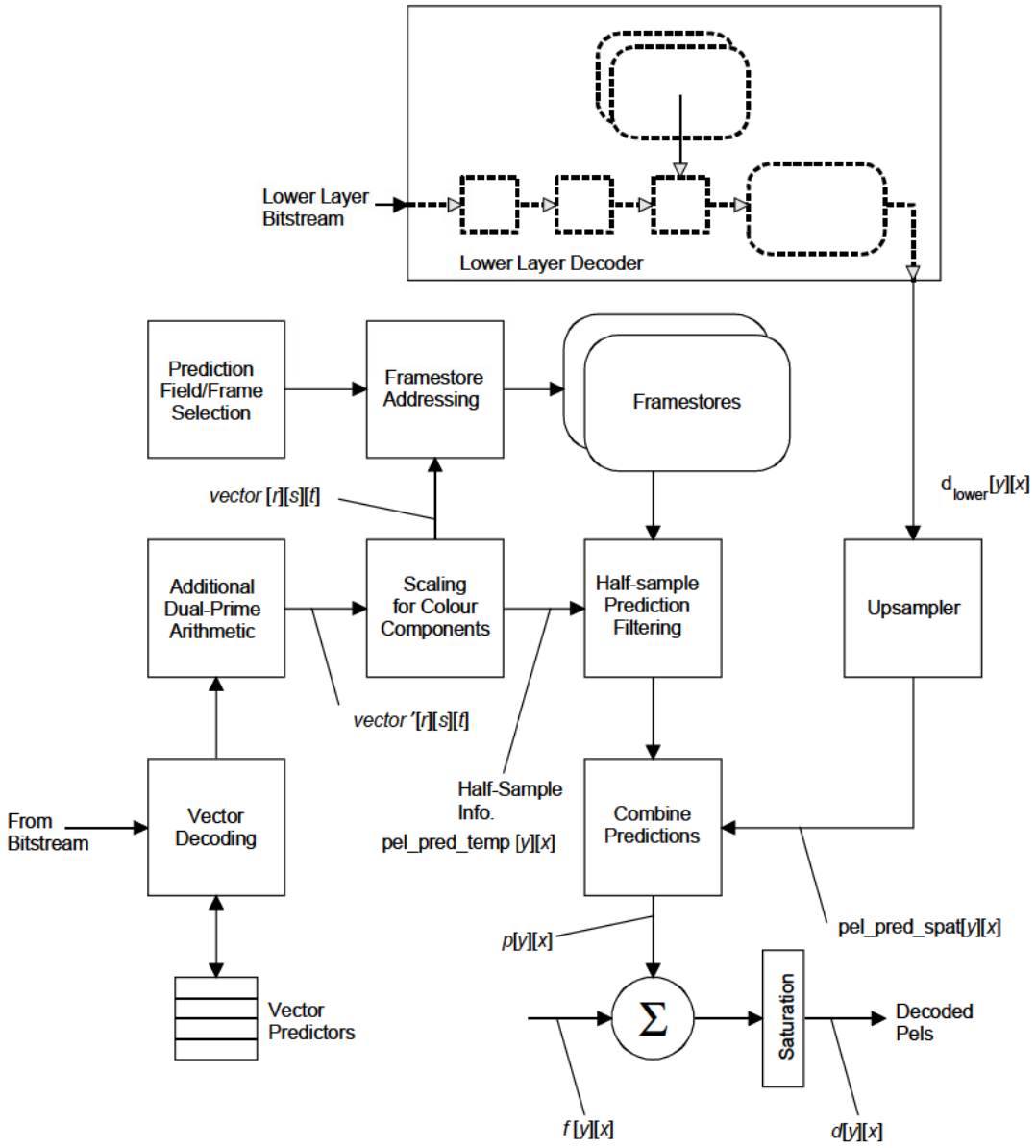



**Figure 7-13 – Simplified motion compensation process for spatial scalability**

### 7.7.3 Formation of spatial prediction

Forming the spatial prediction requires identification of the correct reference frame and definition of the spatial resampling process, which is done in the following subclauses.

The resampling process is defined for a whole frame, however, for decoding of a macroblock, only the 16 × 16 region in the upsampled frame, which corresponds to the position of this macroblock, is needed.

#### 7.7.3.1 Selection of reference frame

The spatial prediction is made from the reconstructed frame of the lower layer referenced by the lower_layer_temporal_reference. However, if lower and enhancement layer bitstreams are embedded in ITU-T Rec. H.220.0 | ISO/IEC 13818-1 (Systems) multiplex, this information is overridden by the timing information given by the decoding time stamps (DTS) in the PES headers.

NOTE – If group_of_pictures_header() occurs often in the lower layer bitstream, then the temporal reference in the lower layer may be ambiguous (because temporal_reference is reset after a group_of_pictures_header()).

The reconstructed picture from which the spatial prediction is made shall be one of the following:

- The coincident or most recently decoded lower layer picture.
- The coincident or most recently decoded lower layer I-picture or P-picture.
- The second most recently decoded lower layer I-picture or P-picture provided that the lower layer does not have low_delay set to '1'. Note furthermore that spatial scalability will only work efficiently when predictions are formed from frames in the lower layer which are also coincident (or very close) in display time with the predicted frame in the enhancement layer.

#### 7.7.3.2 Resampling process

The spatial prediction is made by resampling the lower layer reconstructed frame to the same sample grid as the enhancement layer. This grid is defined in terms of frame coordinates, even if a lower-layer interlaced frame was actually coded with a pair of field pictures.

This resampling process is illustrated in Figure 7-14.

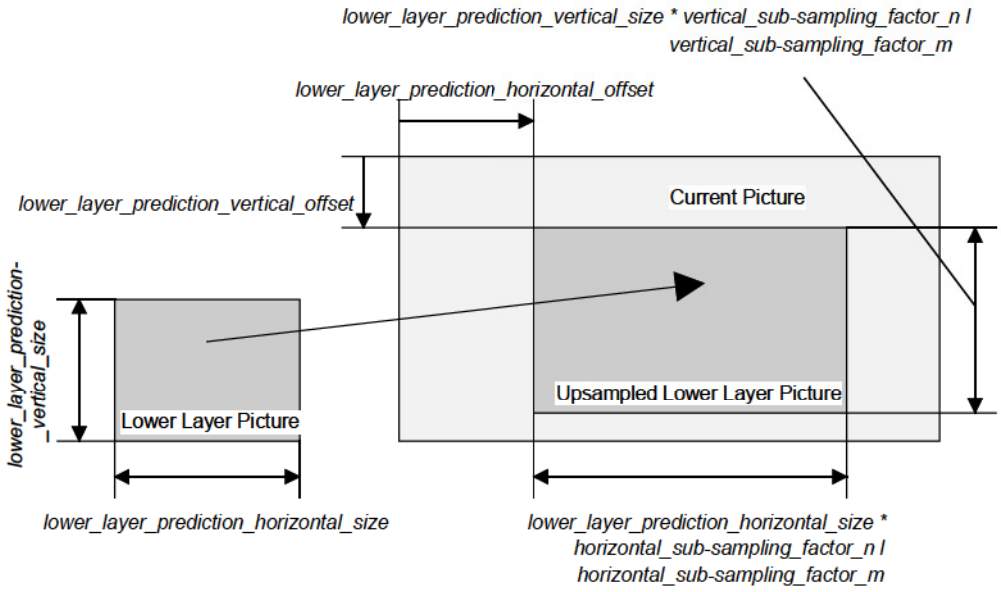



**Figure 7-14 – Formation of the "spatial" prediction by interpolation of the lower layer picture**

Spatial predictions shall only be made for macroblocks in the enhancement layer that lie wholly within the upsampled lower layer reconstructed frame.

The upsampling process depends on whether the lower layer reconstructed frame is interlaced or progressive, as indicated by lower_layer_progressive_frame and whether the enhancement layer frame is interlaced or progressive, as indicated by progressive_frame.

When lower_layer_progressive_frame is '1', the lower layer reconstructed frame (renamed to prog_pic) is resampled vertically as described in 7.7.3.4. The resulting frame is considered to be progressive if progressive_frame is '1' and interlaced if progressive_frame is '0'. The resulting frame is resampled horizontally as described in 7.7.3.6. lower_layer_deinterlaced_field_select shall have the value '1'.

When lower_layer_progressive_frame is '0' and progressive_frame is '0', each lower layer reconstructed field is deinterlaced as described in 7.7.3.4, to produce a progressive field (prog_pic). This field is resampled vertically as described in 7.7.3.5. The resulting field is resampled horizontally as described in 7.7.3.6. Finally the resulting field is subsampled to produce an interlaced field. lower_layer_deinterlaced_field_select shall have the value '1'.

When lower_layer_progressive_frame is '0' and progressive_frame is '1', each lower layer reconstructed field is deinterlaced as described in 7.7.3.4, to produce a progressive field (prog_pic). Only one of these fields is required. When lower_layer_deinterlaced_field_select is '0' the top field is used, otherwise the bottom field is used. The one that is used is resampled vertically as described in 7.7.3.5. The resulting frame is resampled horizontally as described in 7.7.3.6.

For interlaced frames, if the current (and implicitly the lower-layer) frames are encoded as field pictures, the deinterlacing process described in 7.7.3.5 is done within the field.

lower_layer_vertical_offset and lower_layer_horizontal_offset, defining the position of the lower layer frame within the current frame, shall be taken into account in the resampling definitions in 7.7.3.5 and 7.7.3.6 respectively. The lower layer offsets are limited to even values when the chrominance in the enhancement layer is subsampled in that dimension in order to align the chrominance samples between the two layers.

The upsampling process is summarised Table 7-15.

**Table 7-15 – Upsampling process**

| lower_layer_ deinterlaced_ field_select | lower_layer_ progressive_frame | progressive_ frame | Apply deinterlace process | Entity used for prediction |
|---|---|---|---|---|
| 0 | 0 | 1 | Yes | Top field |
| 1 | 0 | 1 | Yes | Bottom field |
| 1 | 1 | 1 | No | Frame |
| 1 | 1 | 0 | No | Frame |
| 1 | 0 | 0 | Yes | Both fields |

### 7.7.3.3 Colour component processing

Due to the different sampling grids of luminance and chrominance components, some variables used in 7.7.3.4 to 7.7.3.6 take different values for luminance and chrominance resampling. Furthermore it is permissible for the chrominance formats in the lower layer and the enhancement layer to be different from one another.

Table 7-16 defines the values for the variables used in 7.7.3.4 to 7.7.3.6

**Table 7-16 – Local variables used in 7.7.3.3 to 7.7.3.6**

| Variable | Value for luminance processing | Value for chrominance processing |
|---|---|---|
| ll_h_size | lower_layer_prediction_horizontal_size | lower_layer_prediction_horizontal_size / chroma_ratio_horizontal[lower] |
| ll_v_size | lower_layer_prediction_vertical_size | lower_layer_prediction_vertical_size / chroma_ratio_vertical[lower] |
| ll_h_offset | lower_layer_horizontal_offset | lower_layer_horizontal_offset / chroma_ratio_horizontal[enhance] |
| ll_v_offset | lower_layer_vertical_offset | lower_layer_vertical_offset / chroma_ratio_vertical[enhance] |
| h_subs_m | horizontal_subsampling_factor_m | horizontal_subsampling_factor_m |
| h_subs_n | horizontal_subsampling_factor_n | horizontal_subsampling_factor_n * format_ratio_horizontal |
| v_subs_m | vertical_subsampling_factor_m | vertical_subsampling_factor_m |
| v_subs_n | vertical_subsampling_factor_n | vertical_subsampling_factor_n * format_ratio_vertical |

Tables 7-17 and 7-18 give additional definitions.

**Table 7-17 – chrominance subsampling ratios for layer = {lower, enhance}**

| Chrominance format lower layer | chroma_ratio_ horizontal[layer] | chroma_ratio_ vertical[layer] |
|---|---|---|
| 4:2:0 | 2 | 2 |
| 4:2:2 | 2 | 1 |
| 4:4:4 | 1 | 1 |

**Table 7-18 – Chrominance format ratios**

| Chrominance format lower layer | Chrominance format enhancement layer | format_ratio_ horizontal | format_ratio_ vertical |
|---|---|---|---|
| 4:2:0 | 4:2:0 | 1 | 1 |
| 4:2:0 | 4:2:2 | 1 | 2 |
| 4:2:0 | 4:4:4 | 2 | 2 |
| 4:2:2 | 4:2:2 | 1 | 1 |
| 4:2:2 | 4:4:4 | 2 | 1 |
| 4:4:4 | 4:4:4 | 1 | 1 |

#### 7.7.3.4 Deinterlacing

If deinterlacing needs not to be done (according to Table 7-16), the lower layer reconstructed frame ($d_{lower}[y][x]$) is renamed to input_pic.

First, each lower layer field is padded with zeros to form a progressive grid at a frame rate equal to the field rate of the lower layer, and with the same number of lines and samples per line as the lower layer frame. Table 7-19 specifies the filters to be applied next. The luminance component is filtered using the relevant two field aperture filter if picture_structure == "Frame-Picture" or else using the one field aperture filter. The chrominance component is filtered using the one field aperture filter.

**Table 7-19 – Deinterlacing Filter**

| Temporal | Vertical | Two field aperture | | One field aperture |
|---|---|---|---|---|
| | | Filter for first field | Filter for second field | Filter (both fields) |
| –1 | –2 | 0 | –1 | 0 |
| –1 | 0 | 0 | 2 | 0 |
| –1 | 2 | 0 | –1 | 0 |
| 0 | –1 | 8 | 8 | 8 |
| 0 | 0 | 16 | 16 | 16 |
| 0 | 1 | 8 | 8 | 8 |
| 1 | –2 | –1 | 0 | 0 |
| 1 | 0 | 2 | 0 | 0 |
| 1 | +2 | –1 | 0 | 0 |

The temporal and vertical columns of Table 7-19 indicate the relative spatial and temporal coordinates of the samples to which the filter taps defined in the other two columns apply. An intermediate sum is formed by adding the multiplied coefficients together.

The output of the filter (sum) is then scaled according to the following formula:

$$prog\_pic[y][x] = sum \,//\, 16$$

and saturated to lie in the range [0:255].

The filter aperture can extend outside the coded picture size. In this case the samples of the lines outside the active picture shall take the value of the closest neighbouring existing sample (below or above) of the same field as defined below.

For all samples [y][x]:

```
if (y < 0 && (y&1 == 1))
    y = 1
if (y < 0 && (y&1 == 0))
    y = 0
if (y >= ll_v_size &&
            ( (y-ll_v_size)&1 == 1))
    y = ll_v_size – 1
if (y >= ll_v_size &&
            ((y-ll_v_size)&1 == 0))
    y = ll_v_size – 2
```

#### 7.7.3.5 Vertical resampling

The frame subject to vertical resampling, prog_pic, is resampled to the enhancement layer vertical sampling grid using linear interpolation between the sample sites according to the following formula, where vert_pic is the resulting field:

$$vert\_pic[y_h + ll\_v\_offset][x] = (16 - phase) * prog\_pic[y1][x] + phase * prog\_pic[y2][x]$$

where

$y_h + ll\_v\_offset$ = output sample co-ordinate in vert_pic

$y1$ = $(y_h * v\_subs\_m) / v\_subs\_n$

$y2$ = $y1 + 1$ if $y1 < ll\_v\_size - 1$; $y1$ otherwise

$phase$ = $(16 * ((y_h * v\_subs\_m) \,\%\, v\_subs\_n)) \,//\, v\_subs\_n$

Samples which lie outside the lower layer reconstructed frame which are required for upsampling are obtained by border extension of the lower layer reconstructed frame.

NOTE – The calculation of phase assumes that the sample position in the enhancement layer at $y_h = 0$ is spatially coincident with the first sample position of the lower layer. It is recognised that this is an approximation for the chrominance component if the chroma_format == 4:2:0.

#### 7.7.3.6 Horizontal resampling

The frame subject to horizontal resampling, vert_pic, is resampled to the enhancement layer horizontal sampling grid using linear interpolation between the sample sites according to the following formula, where hor_pic is the resulting field:

$$hor\_pic[y][x_h + ll\_h\_offset] = ((16 - phase) * vert\_pic[y][x1] + phase * vert\_pic[y][x2]) \,//\, 256$$

where

$x_h$+ ll_h_offset = output sample coordinate in hor_pic

x1 = ($x_h$ * h_subs_m) / h_subs_n

x2 = x1 + 1 if x1 < ll_h_size - 1

x1 otherwise

phase = (16 * (( $x_h$ * h_subs_m) % h_subs_n)) // h_subs_n

Samples which lie outside the lower layer reconstructed frame which are required for upsampling are obtained by border extension of the lower layer reconstructed frame.

#### 7.7.3.7 Reinterlacing

If reinterlacing needs not to be done, the result of the resampling process, hor_pic, is renamed to spat_pred_pic.

If hor_pic was derived from the top field of a lower layer interlaced frame, the even lines of hor_pic are copied to the even lines of spat_pred_pic.

If hor_pic was derived from the bottom field of a lower layer interlaced frame the odd lines of hor_pic are copied to the odd lines of spat_pred_pic.

If hor_pic was derived from a lower layer progressive frame, hor_pic is copied to spat_pred_pic.

### 7.7.4 Selection and combination of spatial and temporal predictions

The spatial and temporal predictions can be selected or combined to form the actual prediction. The macroblock_type (see Tables B.5, B.6 and B.7) and the additional spatial_temporal_weight_code (see Table 7-21) indicate, by use of the spatial_temporal_weight_class, whether the prediction is temporal-only, spatial-only or a weighted combination of temporal and spatial predictions. Classes are defined in the following way:

- Class 0 indicates temporal-only prediction;
- Class 1 indicates that neither field has spatial-only prediction;
- Class 2 indicates that the top field is spatial-only prediction;
- Class 3 indicates that the bottom field is spatial-only prediction;
- Class 4 indicates spatial-only prediction.

In intra pictures, if spatial_temporal_weight_class is 0, normal intra coding is performed, otherwise the prediction is spatial-only. In predicted and interpolated pictures, if the spatial_temporal_weight_class is 0, prediction is temporal-only, if the spatial_temporal_weight_class is 4, prediction is spatial-only, otherwise one or a pair of prediction weights is used to combine the spatial and temporal predictions.

The possible spatial_temporal_weights are given in a weight table which is selected in the picture spatial scalable extension. Up to four different weight tables are available for use depending on whether the current and lower layers are interlaced or progressive, as indicated in Table 7-20 (allowed, yet not recommended values given in brackets).

**Table 7-20 – Intended (allowed) spatial_temporal_weight_code_table_index values**

| Lower layer format | Enhancement layer format | **spatial_temporal_weight_ code_table_index** |
|---|---|---|
| Progressive or interlaced | Progressive | 00 |
| Progressive coincident with enhancement layer top fields | Interlaced | 10 (00; 01; 11) |
| Progressive coincident with enhancement layer from bottom fields | Interlaced | 01 (00; 10; 11) |
| Interlaced (picture_structure == Frame-Picture) | Interlaced | 00 or 11 (01; 10) |
| Interlaced (picture_structure != Frame-Picture) | Interlaced | 00 |

In macroblock_modes(), a two bit code, spatial_temporal_weight_code, is used to describe the prediction for each field (or frame), as shown in the Table 7-21. In this table spatial_temporal_integer_weight identifies those spatial_temporal_weight_codes that can also be used with dual prime prediction (see Tables 7-22, 7-23).

**Table 7-21 – spatial_temporal_weights and spatial_temporal_weight_classes for the spatial_temporal_weight_code_table_index and spatial_temporal_weight_codes**

| spatial_temporal_ weight_code_table_ index | spatial_ temporal_ weight_code | spatial_ temporal_ weight (s) | spatial_ temporal_ weight class | spatial_ temporal_ integer_weight |
|---|---|---|---|---|
| 00[a)] | – | (0,5) | 1 | 0 |
| 01 | 00 | (0; 1) | 3 | 1 |
| | 01 | (0; 0,5) | 1 | 0 |
| | 10 | (0,5; 1 ) | 3 | 0 |
| | 11 | (0,5; 0,5) | 1 | 0 |
| 10 | 00 | (1; 0) | 2 | 1 |
| | 01 | (0,5; 0) | 1 | 0 |
| | 10 | (1; 0,5 ) | 2 | 0 |
| | 11 | (0,5; 0,5) | 1 | 0 |
| 11 | 00 | (1; 0) | 2 | 1 |
| | 01 | (1; 0,5) | 2 | 0 |
| | 10 | (0,5; 1 ) | 3 | 0 |
| | 11 | (0,5; 0,5) | 1 | 0 |

a) For spatial_temporal_weight_code_table_index == 00 no spatial_temporal_weight_code is transmitted.

NOTE – Spatial-only prediction (weight_class == 4) is signalled by different values of macroblock_type (see Tables B.5 to B.7).

When the spatial_temporal_weight combination is given in the form (a; b), "a" gives the proportion of the prediction for the top field which is derived from the spatial prediction and "b" gives the proportion of the prediction for the bottom field which is derived from the spatial prediction for that field.

When the spatial_temporal_weight is given in the form (a), "a" gives the proportion of the prediction for the picture which is derived from the spatial prediction for that picture.

The precise method for predictor calculation is as follows:

pel_pred_temp[y][x] is used to denote the temporal prediction (formed within the enhancement layer) as defined for pel_pred[y][x] in 7.6. pel_pred_spat[y][x] is used to denote the prediction formed from the lower layer by extracting the appropriate samples, co-located with the current macroblock position, from spat_pred_pic.

If the spatial_temporal_weight is zero, then no prediction is made from the lower layer. Therefore:

$$\text{pel\_pred}[y][x] = \text{pel\_pred\_temp}[y][x];$$

If the spatial_temporal_weight is one, then no prediction is made from the enhancement layer. Therefore:

$$\text{pel\_pred}[y][x] = \text{pel\_pred\_spat}[y][x];$$

If the weight is one half then the prediction is the average of the temporal and spatial predictions. Therefore:

$$\text{pel\_pred}[y][x] = (\text{pel\_pred\_temp}[y][x] + \text{pel\_pred\_spat}[y][x])//2;$$

When progressive_frame == 0 chrominance is treated as interlaced, that is, the first weight is used for the top field chrominance lines and the second weight is used for the bottom field chrominance lines.

Addition of prediction and coefficient data is then done as in 7.6.8.

### 7.7.5 Updating motion vector predictors and motion vector selection

In frame pictures where field prediction is used the possibility exists that one of the fields is predicted using spatial-only prediction. In this case no motion vector is present in the bitstream for the field which has spatial-only prediction. For the case where both fields of a frame have spatial-only prediction, the macroblock_type is such that no motion vectors are present in the bitstream for that macroblock.

The spatial_temporal_weight_class also indicates the number of motion vectors which are present in the coded bitstream and how the motion vector predictors are updated as defined in Table 7-22 and Table 7-23.

**Table 7-22 – Updating of motion vector predictors in Field Pictures**

| frame_motion_type | macroblock_motion_forward | macroblock_motion_backward | macroblock_intra | spatial_temporal_weight_class | Predictors to update |
|---|---|---|---|---|---|
| Field-based[a)] | – | – | 1 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0][b)] |
| Field-based | 1 | 1 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0]<br>*PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Field-based | 1 | 0 | 0 | 0,1 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |
| Field-based | 0 | 1 | 0 | 0,1 | *PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Field-based[a)] | 0 | 0 | 0 | 0,1,4 | *PMV*[*r*][*s*][*t*] = 0[c)] |
| 16 × 8 MC | 1 | 1 | 0 | 0 | (None) |
| 16 × 8 MC | 1 | 0 | 0 | 0,1 | (None) |
| 16 × 8 MC | 0 | 1 | 0 | 0,1 | (None) |
| Dual prime | 1 | 0 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |

a) **field_motion_type** is not present in the bitstream but is assumed to be Field-based.

b) If **concealment_motion_vectors** is zero then *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*).

c) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.6.3.4.

NOTE – *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that:

*PMV*[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

#### 7.7.5.1 Resetting motion vector predictors

In addition to the cases identified in 7.6.3.4, the motion vector predictors shall be reset in the following cases:

- In a P-picture when a macroblock is purely spatially predicted (spatial_temporal_weight_class == 4)
- In a B-picture when a macroblock is purely spatially predicted (spatial_temporal_weight_class == 4)

NOTE – In case of spatial_temporal_weight_class == 2 in a frame picture when field-based prediction is used, the transmitted vector is applied for the *bottom* field (see Table 7-25). However this vector[0][s][1:0] is predicted from PMV[0][s][1:0] . PMV[1][s][1:0] is then updated as shown in Table 7-23.

### 7.7.6 Skipped macroblocks

In all cases, a skipped macroblock is the result of a prediction only, and all the DCT coefficients are considered to be zero.

If sequence_scalable_extension is present and scalable_mode = "spatial scalability", the following rules apply in addition to those given in 7.6.6.

**Table 7-23 – Updating of motion vector predictors in Frame Pictures**

| frame_motion_type | macroblock_motion_forward | macroblock_motion_backward | macroblock_intra | spatial_temporal_weight_class | Predictors to update |
|---|---|---|---|---|---|
| Frame-based[a)] | – | – | 1 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0][c)] |
| Frame-based | 1 | 1 | 0 | 0 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0]<br>*PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Frame-based | 1 | 0 | 0 | 0,1,2,3 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |
| Frame-based | 0 | 1 | 0 | 0,1,2,3 | *PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Frame-based[a)] | 0 | 0 | 0 | 0,1,2,3,4 | *PMV*[r][s][t] = 0[d)] |
| Field-based | 1 | 1 | 0 | 0 | (None) |
| Field-based | 1 | 0 | 0 | 0,1 | (None) |
| Field-based | 1 | 0 | 0 | 2 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |
| Field-based | 1 | 0 | 0 | 3 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |
| Field-based | 0 | 1 | 0 | 0,1 | (None) |
| Field-based | 0 | 1 | 0 | 2 | *PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Field-based | 0 | 1 | 0 | 3 | *PMV*[1][1][1:0] = *PMV*[0][1][1:0] |
| Dual prime[b)] | 1 | 0 | 0 | 0,2,3 | *PMV*[1][0][1:0] = *PMV*[0][0][1:0] |

a) **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based.

b) Dual prime can not be used when spatial_temporal_integer_weight = '0'.

c) If **concealment_motion_vectors** is zero then *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*).

d) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.6.3.4.

NOTE – *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that:

*PMV*[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

In I-pictures, skipped macroblocks are allowed. These are defined as spatial-only predicted.

In P-pictures and B-pictures, the skipped macroblock is temporal-only predicted.

In B-pictures a skipped macroblock shall not follow a spatial-only predicted macroblock.

### 7.7.7 VBV buffer underflow in the lower layer

In the case of spatial scalability, VBV buffer underflow in the lower layer may cause problems. This is because of possible uncertainty in precisely which frames will be repeated by a particular decoder.

## 7.8 SNR scalability

See Figure 7-15.

This clause describes the additional decoding process required for the SNR scalable extensions.

SNR scalability defines a mechanism to refine the DCT coefficients encoded in another (lower) layer of a scalable hierarchy. As illustrated in Figure 7-15 data from two bitstreams is combined after the inverse quantisation processes by adding the DCT coefficients. Until the data is combined, the decoding processes of the two layers are independent of one another.

**Table 7-24 – Predictions and motion vectors in field pictures**

| field_motion_type | macroblock_motion_forward | macroblock_motion_backward | macroblock_intra | spatial_temporal_weight_class | Motion vector | Prediction formed for |
|---|---|---|---|---|---|---|
| Field-based[a)] | – | – | 1 | 0 | *vector*'[0][0][1:0][b)] | None (motion vector is for concealment) |
| Field-based | 1 | 1 | 0 | 0 | *vector*'[0][0][1:0] | Whole field, forward |
| | | | | | *vector*'[0][1][1:0] | Whole field, backward |
| Field-based | 1 | 0 | 0 | 0,1 | *vector*'[0][0][1:0] | Whole field, forward |
| Field-based | 0 | 1 | 0 | 0,1 | *vector*'[0][1][1:0] | Whole field, backward |
| Field-based[a)] | 0 | 0 | 0 | 0,1,4 | *vector*'[0][0][1:0][c) d)] | Whole field, forward |
| 16 × 8 MC | 1 | 1 | 0 | 0 | *vector*'[0][0][1:0] | Upper 16 × 8 field, forward |
| | | | | | *vector*'[1][0][1:0] | Lower 16 × 8 field, forward |
| | | | | | *vector*'[0][1][1:0] | Upper 16 × 8 field, backward |
| | | | | | *vector*'[1][1][1:0] | Lower 16 × 8 field, backward |
| 16 × 8 MC | 1 | 0 | 0 | 0,1 | *vector*'[0][0][1:0] | Upper 16 × 8 field, forward |
| | | | | | *vector*'[1][0][1:0] | Lower 16 × 8 field, forward |
| 16 × 8 MC | 0 | 1 | 0 | 0,1 | *vector*'[0][1][1:0] | Upper 16 × 8 field, backward |
| | | | | | *vector*'[1][1][1:0] | Lower 16 × 8 field, backward |
| Dual prime | 1 | 0 | 0 | 0 | *vector*'[0][0][1:0] | Whole field, same parity, forward |
| | | | | | *vector*'[2][0][1:0][c) e)] | Whole field, opposite parity, forward |

a) **field_motion_type** is not present in the bitstream but is assumed to be Field-based.

b) The motion vector is only present if **concealment_motion_vectors** is one.

c) These motion vectors are not present in the bitstream.

d) The motion vector is taken to be (0; 0) as explained in 7.6.3.5.

e) These motion vectors are derived from *vector*'[0][0][1:0] as described in 7.6.3.6.

NOTE – Motion vectors are listed in the order they appear in the bitstream.

Subclause 7.8.1 defines how to identify these bitstreams in a scalable hierarchy, however they can be classified as follows.

The lower layer, derived from the first bitstream, can itself be either non-scalable, or require the spatial or temporal scalability decoding process (and hence the decoding of additional bitstreams) to be applied.

The enhancement layer, derived from the second bitstream, contains mainly coded DCT coefficients and a small overhead. The decoding process for this layer and the combination of the two layers are described in this subclause.

NOTE – All information regarding prediction is contained in the lower layer bitstream only. Therefore it is not possible to reconstruct an enhancement layer without decoding the lower layer bitstream data in parallel.

Furthermore prediction and reconstruction of the pictures as described in 7.6, 7.7 and 7.9 for the combined lower and enhancement layer is identical to the respective steps for decoding of the lower layer bitstream only.

Semantics and decoding process described in this subclause include a mechanism for "chroma simulcast". This may be used (for instance) to enhance 4:2:0 in the lower layer to 4:2:2 after processing the enhancement layer data. While the luminance data is processed as described before, in this case the chrominance information retrieved from the lower layer bitstream (with exception of intra-DC values, see 7.8.3.4) shall be discarded and replaced by the new information with higher chrominance resolution decoded from the enhancement layer.

**Table 7-25 – Predictions and motion vectors in frame pictures**

| frame_motion_type | macroblock_motion_forward | macroblock_motion_backward | macroblock_intra | spatial_temporal_weight_class | Motion vector | Prediction formed for |
|---|---|---|---|---|---|---|
| Frame-based[a)] | – | – | 1 | 0 | *vector*'[0][0][1:0][c)] | None (motion vector is for concealment) |
| Frame-based | 1 | 1 | 0 | 0 | *vector*'[0][0][1:0] | Frame, forward |
| | | | | | *vector*'[0][1][1:0] | Frame, backward |
| Frame-based | 1 | 0 | 0 | 0,1,2,3 | *vector*'[0][0][1:0] | Frame, forward |
| Frame-based | 0 | 1 | 0 | 0,1,2,3 | *vector*'[0][1][1:0] | Frame, backward |
| Frame-based[a)] | 0 | 0 | 0 | 0,1,2,3,4 | *vector*'[0][0][1:0][d) e)] | Frame, forward |
| Field-based | 1 | 1 | 0 | 0 | *vector*'[0][0][1:0] | Top field, forward |
| | | | | | *vector*'[1][0][1:0] | Bottom field, forward |
| | | | | | *vector*'[0][1][1:0] | Top field, backward |
| | | | | | *vector*'[1][1][1:0] | Bottom field, backward |
| Field-based | 1 | 0 | 0 | 0,1 | *vector*'[0][0][1:0] | Top field, forward |
| | | | | | *vector*'[1][0][1:0] | Bottom field, forward |
| Field-based | 1 | 0 | 0 | 2 | | Top field, spatial |
| | | | | | *vector*'[0][0][1:0] | Bottom field, forward |
| Field-based | 1 | 0 | 0 | 3 | *vector*'[0][0][1:0] | Top field, forward |
| | | | | | | Bottom field, spatial |
| Field-based | 0 | 1 | 0 | 0,1 | *vector*'[0][1][1:0] | Top field, backward |
| | | | | | *vector*'[1][1][1:0] | Bottom field, backward |
| Field-based | 0 | 1 | 0 | 2 | | Top field, spatial |
| | | | | | *vector*'[0][1][1:0] | Bottom field, backward |
| Field-based | 0 | 1 | 0 | 3 | *vector*'[0][1][1:0] | Top field, backward |
| | | | | | | Bottom field, spatial |
| Dual prime[b)] | 1 | 0 | 0 | 0,2,3 | *vector*'[0][0][1:0] | Top field, same parity, forward |
| | | | | | *vector*'[0][0][1:0][d)] | Bottom field, same parity, forward |
| | | | | | *vector*'[2][0][1:0][d) f)] | Top field, opposite parity, forward |
| | | | | | *vector*'[3][0][1:0][d) f)] | Bottom field, opposite parity, forward |

a) **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based.

b) Dual prime cannot be used when spatial_temporal_integer_weight = '0'.

c) The motion vector is only present if **concealment_motion_vectors** is one.

d) These motion vectors are not present in the bitstream.

e) The motion vector is taken to be (0; 0) as explained in 7.6.3.5.

f) These motion vectors are derived from *vector*'[0][0][1:0] as described in 7.6.3.6.

NOTE – Motion vectors are listed in the order they appear in the bitstream.

It is inherent in SNR scalability that the two layers are very tightly coupled to one another. It is a requirement that corresponding pictures in each layer shall be decoded at the same time as one another.

In the case that the lower layer bitstream conforms to ISO/IEC 11172-2 (and not this Specification), then two different IDCT mismatch control schemes are being used in decoding. Care must be taken in the encoder to take account of this.

### 7.8.1 Higher syntactic structures

The two bitstream layers in this subclause are identified by their layer_id, decoded from the sequence_scalable_extension.

The two bitstreams shall have consecutive layer ids, with enhancement layer bitstream having layer_id = $id_{enhance}$ and the lower layer bitstream having layer_id = $id_{enhance}$–1.

The syntax and semantics of the enhancement layer are as defined in 6.2 and 6.3, respectively.

In the case that the lower layer bitstream conforms to ISO/IEC 11172-2 (and not this Specification), then both this lower and the enhancement layer shall use the "restricted slice structure" defined in this Specification.

Semantic restrictions apply to several values in the headers and extensions of the enhancement layer as follows.

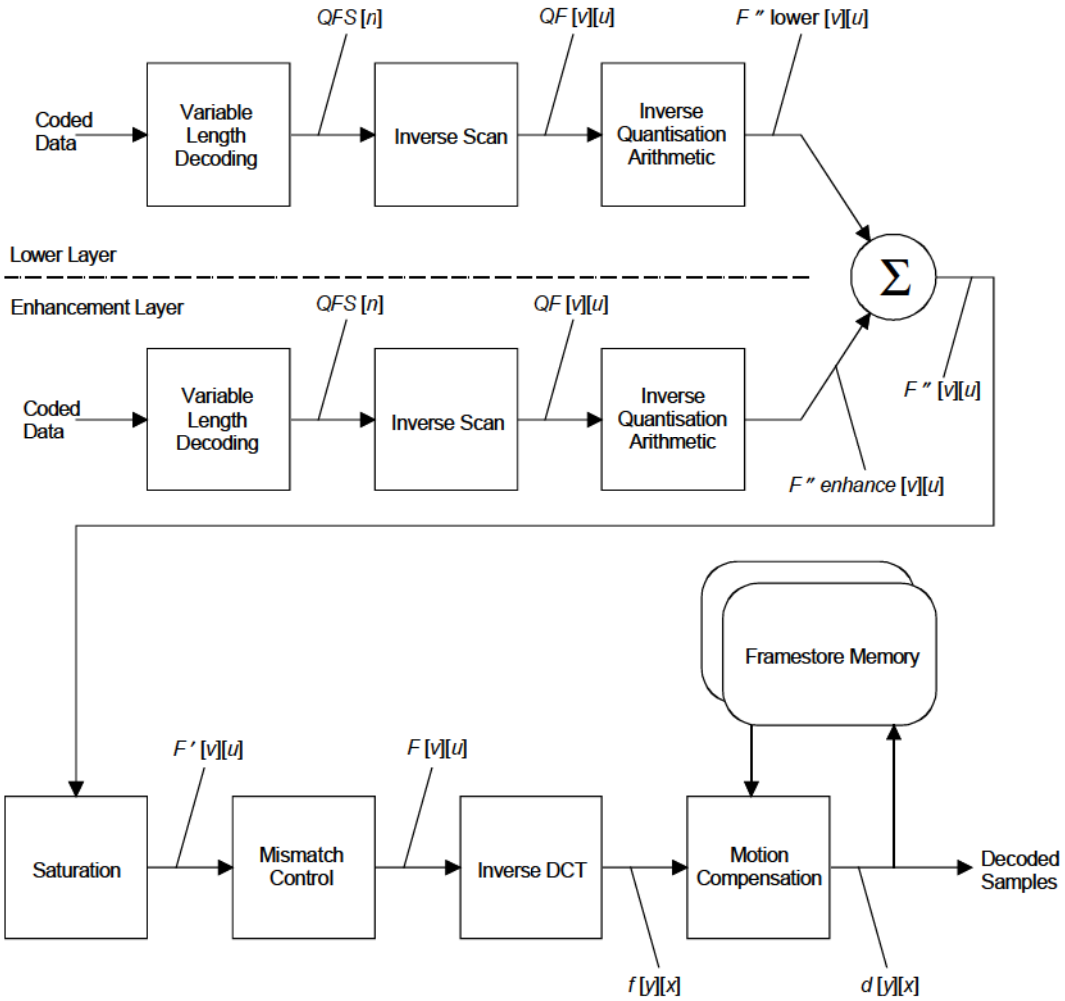



**Figure 7-15 – Illustration of decoding process for SNR scalability**

**Sequence header**

This header shall be identical to the one in the lower layer bitstream except for the values of bit_rate, vbv_buffer_size, load_intra_quantiser_matrix, intra_quantiser_matrix, load_non_intra_quantiser_matrix and non_intra_quantiser_matrix. These can be selected independently except for load_intra_quantiser_matrix which shall be zero.

**Sequence extension**

This extension shall be identical to the one in the lower layer bitstream except for the values of profile_and_level_indication, chroma_format, bit_rate_extension and vbv_buffer_size_extension. Those can be selected independently.

A different value of chroma_format in each layer will cause the chroma_simulcast flag to be set as specified by Table 7-26.

**Table 7-26 – chroma_simulcast flag**

| chroma_format (lower layer) | chroma_format (enhancement layer) | chroma_simulcast |
|---|---|---|
| 4:2:0 | 4:2:0 | 0 |
| 4:2:0 | 4:2:2 | 1 |
| 4:2:0 | 4:4:4 | 1 |
| 4:2:2 | 4:2:2 | 0 |
| 4:2:2 | 4:4:4 | 1 |
| 4:4:4 | 4:4:4 | 0 |

The chroma_format of the enhancement layer shall be higher or equal to the chroma_format of the lower layer bitstream.

In the case that the lower layer bitstream conforms to ISO/IEC 11172-2 (and not This Specification), sequence_extension() is not present in the lower layer bitstream, and the following values shall be assumed for the decoding process.

progressive_sequence = 1
chroma_format = "4:2:0"
horizontal_size_extension = 0
vertical_size_extension = 0
bit_rate_extension = 0
vbv_buffer_size_extension = 0
low_delay = 0
frame_rate_extension_n = 0
frame_rate_extension_d = 0

The sequence_extension() in the enhancement layer shall have the values shown above.

**Sequence display extension**

This extension shall not be present as there is no separate display process for the enhancement layer.

**Sequence scalable extension**

This extension shall be present with scalable_mode = "SNR scalability".

**GOP header**

This header shall be identical to the one in the lower layer bitstream.

NOTE 1 – The GOP header must be present in each layer in order that the temporal_reference in each layer are reset on the same frame.

**Picture header**

This header shall be identical to the one in the lower layer bitstream except for the value of vbv_delay. This can be selected independently.

**Picture coding extension**

This extension shall be identical to the one in the lower layer bitstream except for the value of q_scale_type and alternate_scan. These can be selected independently.

chroma_420_type shall be set to '0' if chroma_simulcast is set. Else it shall have the same value as in the lower layer bitstream.

In the case that the lower layer bitstream conforms to ISO/IEC 11172-2 (and not this Specification), then picture_coding_extension() is not present in the lower layer bitstream and the following values shall be assumed for the decoding process:

| | | |
|---|---|---|
| f_code[0][0] | = | forward_f_code in the lower layer bitstream or 15 |
| f_code[0][1] | = | forward_f_code in the lower layer bitstream or 15 |
| f_code[1][0] | = | backward_f_code in the lower layer bitstream or 15 |
| f_code[1][1] | = | backward_f_code in the lower layer bitstream or 15 |
| intra_dc_precision | = | 0 |
| picture_structure | = | "Frame Picture" |
| top_field_first | = | 0 |
| frame_pred_frame_dct | = | 1 |
| concealment_motion_vectors | = | 0 |
| intra_vlc_format | = | 0 |
| repeat_first_field | = | 0 |
| chroma_420_type | = | 1 |
| progressive_frame | = | 1 |
| composite_display_flag | = | 0 |

The picture_coding_extension() in the enhancement layer shall have the values shown above.

For the lower layer q_scale_type and alternate_scan shall be assumed to have the value zero.

NOTE 2 – q_scale_type and alternate_scan can be set independently in the enhancement layer.

**Quant matrix extension**

This extension is optional. Semantics are described in 6.3.11.

load_intra_quantiser_matrix and load_chroma_intra_quantiser_matrix shall both be zero.

NOTE 3 – Only the non-intra matrices will be used in the subsequent decoding process.

**Picture display extension**

This extension shall not be present.

NOTE 4 – There is no separate display process for the enhancement layer. If pan-scan functionality is desired, it can be accomplished already by using the information conveyed by the pan-scan extension of the lower layer bitstream.

**Slice header**

Slices shall be coincident with those in the lower layer. The value of quantiser_scale_code can be set independently from the lower layer bitstream.

### 7.8.2 Macroblock

Subsequently the "current macroblock" denotes the currently processed macroblock. The current macroblock of the lower layer denotes the macroblock identified by having the same macroblock_address as the current macroblock.

The decoding of the macroblock header information is done according to semantics in 6.3.17.

NOTE – Table B.8 which is used if scalable_mode = = "SNR scalability" will never set the macroblock_intra, macroblock_motion_forward or macroblock_motion_backward flags, since a macroblock in the enhancement layer contains only refinement data for the current macroblock of the lower layer.

However the corresponding syntax elements and flags of the current macroblock in the lower layer bitstream are relevant for the combined decoding process of lower and enhancement layer following the inverse DCT as described in 7.8.3.5.

#### 7.8.2.1 dct_type

The syntax element dct_type may be present in none, one or both of the lower and enhancement layer macroblock_modes(), as indicated by the semantics in 6.3.17.

If dct_type is present in the macroblock_modes() in both layers it shall have identical values.

#### 7.8.2.2 Skipped Macroblocks

Macroblocks can be skipped in the enhancement layer bitstream, meaning that no coefficient enhancement is done ($F''_{enhance}[v][u] = 0$, for all v, u). Regarding this, the decoding process detailed in 7.8.3 shall be applied.

When macroblocks are skipped in both, the lower and the enhancement layer bitstreams, the decoding process is exactly as specified in 7.6.6.

Macroblocks can also be skipped in the lower layer bitstream, while still being coded in the enhancement layer bitstream. In that case the decoding process detailed in the following has to be applied, but $F''_{lower}[v][u] = 0$, for all $v$, $u$.

### 7.8.3 Block

The first part of the decoding process of the enhancement layer block is independent from the lower layer.

The second part of the decoding process of the enhancement layer block has to be done jointly with the decoding process of the coincident lower layer block.

Two sets of inverse quantised coefficients $F''_{lower}$ and $F''_{enhance}$ are added to form F" (see Figure 7-15).

$F''_{lower}$ is derived from the lower layer bitstream exactly as defined in 7.1 to 7.4.2.3.

$F''_{enhance}$ is derived as is defined in the clauses below.

The resulting F" is further processed, starting with saturation, as defined in 7.4.3 to 7.6 (7.7, 7.9).

#### 7.8.3.1 Variable length decoding

In an enhancement layer block the VLC decoding shall be performed according to 7.2, as for a non-intra block (as indicated by macroblock_intra = 0).

#### 7.8.3.2 Inverse scan

Inverse scan shall be done exactly as defined in 7.3.

#### 7.8.3.3 Inverse quantisation

In an enhancement layer block the inverse quantisation shall be performed according to 7.4.2 as for a non-intra block.

In the case that the lower layer bitstream conforms to ISO/IEC 11172-2 (and not this Specification), then the "inverse quantisation arithmetic" used to derive $F''_{lower}[v][u]$ (see Figure 7-14) shall include the IDCT mismatch control (oddification) and saturation specified in ISO/IEC 11172-2.

#### 7.8.3.4 Addition of coefficients from the two layers

Corresponding coefficients from the blocks of each layer shall be added together to form $F''$ (see Figure 7-15).

$$F''[v][u] = F''_{lower}[v][u] + F''_{enhance}[v][u], \text{ for all } u, v$$

If chroma_simulcast = 1 is set only the luminance blocks are treated as described above.

For chrominance blocks the DC coefficient of the base layer is used as a prediction of the DC coefficient in the coincident block in the enhancement layer, whereas the AC coefficients of the base layer are discarded and AC coefficients of the enhancement layer form F" in Figure 7-14 according to the following formulae:

$$F''[0][0] = F''_{lower}[0][0] + F''_{enhance}[0][0]$$

$$F''[v][u] = F''_{enhance}[v][u], \text{ for all } u, v \text{ except } u = v = 0$$

NOTE – Chroma simulcast blocks are inverse quantised like non-intra blocks and use the chrominance non-intra matrix.

Table 7-27 gives the index of the chrominance block whose DC coefficient ($F''_{lower}[0][0]$) is to be used to predict the DC coefficient in the coincident chrominance block of the enhancement layer ($F''_{enhance}[0][0]$).

**Table 7-27 – Block index used to predict DC coefficient**

| chroma_format | Block index | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| base: 4:2:0<br>upper: 4:2:2 | 4 | 5 | 4 | 5 | | | | |
| base: 4:2:0<br>upper: 4:4:4 | 4 | 5 | 4 | 5 | 4 | 5 | 4 | 5 |
| base: 4:2:2<br>upper: 4:4:4 | 4 | 5 | 6 | 7 | 4 | 5 | 6 | 7 |

#### 7.8.3.5 Remaining macroblock decoding steps

After addition of coefficients from the two layers, the remainder of the macroblock decoding steps is exactly as described in 7.4.3 to 7.6 (7.7, 7.9, if applicable), since there is now only one data stream F"[v][u] to be processed.

In this process, the spatio/temporal prediction p[y][x] is derived according to the macroblock type syntax elements and flags for the current macroblock known from the lower layer bitstream.

## 7.9 Temporal scalability

Temporal scalability involves two layers, a lower layer and an enhancement layer. Both the lower and the enhancement layers process the same spatial resolution. The enhancement layer enhances the temporal resolution of the lower layer and if temporally re-multiplexed with the lower layer provides full temporal rate. This is the frame rate indicated in the enhancement layer. The decoding process for enhancement layer pictures is similar to the normal decoding process described in 7.1 to 7.6. The only difference is in the "Prediction field and frame selection" described in 7.6.2.

The reference frames for prediction are selected by reference_select_code as described in Tables 7-28 and 7-29. In P-pictures, the forward reference picture can be one of the following three: most recent enhancement picture, most recent lower layer frame, or next lower layer frame in display order. Note that in the latter case, the reference frame in lower layer used for prediction is backward in time.

**Table 7-28 – Prediction references selection in P-pictures**

| reference_select_code | Forward prediction reference |
|---|---|
| 00 | Most recent decoded enhancement picture(s) |
| 01 | Most recent lower layer frame in display order |
| 10 | Next lower layer frame in display order |
| 11 | Forbidden |

**Table 7-29 – Prediction references selection in B-pictures**

| reference_select_code | Forward prediction reference | Backward prediction reference |
|---|---|---|
| 00 | Forbidden | Forbidden |
| 01 | Most recent decoded enhancement picture(s) | Most recent lower layer picture in display order |
| 10 | Most recent decoded enhancement picture(s) | Next lower layer picture in display order |
| 11 | Most recent lower layer picture in display order | Next lower layer picture in display order |

In B-pictures, the forward reference can be one of the following two: most recent the enhancement pictures or most recent (or temporally coincident) lower layer frame whereas the backward reference can be one of the following two: most recent lower layer picture including temporally coincident picture in display order or next lower layer frame in display order. Note that in this case, the backward reference frame in lower layer used for prediction is forward in time.

Backward prediction cannot be made from a picture in the enhancement layer. This avoids the need for frame reordering in the enhancement layer. Motion compensation process forms predictions using lower layer decoded pictures and/or previous temporal prediction from the enhancement layer.

The enhancement layer can contain I-pictures, P-pictures or B-pictures, but B-pictures in enhancement layer behave more like P-pictures in the sense that a decoded B-picture can be used to predict the following P-pictures or B-pictures in the enhancement layer.

When the most recent frame in the lower layer is used as the reference, this includes the frame that is temporally coincident with the frame or the first field (in case of field pictures) in the enhancement layer. The prediction references used for P-picture and B-pictures are shown in Table 7-28 and Table 7-29 respectively.

The lower and enhancement layers shall use the restricted slice structure.

Figure 7-16 shows a simplified diagram of the motion compensation process for the enhancement layer using temporal scalability.

I-pictures do not use prediction references; to indicate this, the reference_select_code for I-pictures shall be '11'.

Depending on picture_coding_type, when forward_temporal_reference or backward_temporal_reference do not imply references to be used for prediction, they shall take the value 0.

### 7.9.1 Higher syntactic structures

The two bitstream layers in this subclause are identified by their layer_id, decoded from the sequence_scalable_extension.

The two bitstreams shall have consecutive layer ids, with enhancement layer having layer_id = $id_{enhance}$ and the lower layer having layer_id = $id_{enhance}$–1.

The syntax and semantics of enhancement layers are as defined in 6.2 and 6.3 respectively.

Semantic restrictions apply to several values in the headers and extensions of the enhancement layer as follows.

The lower layer shall conform to this Specification (and not to ISO/IEC 11172-2).

**Sequence header**

The values in this header can be different from the lower layer except for horizontal_size_value, vertical_size_value and aspect_ratio_information.

**Sequence extension**

This extension shall be identical to the one in the lower layer except for values of profile_and_level_indication, bit_rate_extension, vbv_buffer_size_extension, low_delay, frame_rate_extension_n and frame_rate_extension_d. These can be selected independently. Note that progressive_sequence indicates the scanning format of the enhancement layer frames only rather than of the output frames after multiplexing. The latter is indicated by mux_to_progressive_sequence (see sequence scalable extension).

**Sequence display extension**

This extension shall not be present as there is no separate display process for the enhancement layer.

**Sequence scalable extension**

This extension shall be present with scalable_mode = "Temporal scalability".

When progressive_sequence = 0 and mux_to_progressive_sequence = 0, top_field_first and picture_mux_factor can be selected.

When progressive_sequence = 0 and mux_to_progressive_sequence = 1, top_field_first shall contain a complement of the value of top_field_first of the lower layer but picture_mux_factor shall be 1.

When progressive_sequence = 1 and mux_to_progressive_sequence = 1, top_field_first shall be zero but picture_mux_factor can be selected.

The combination of progressive_sequence = 1 and mux_to_progressive_sequence = 0 shall not occur.

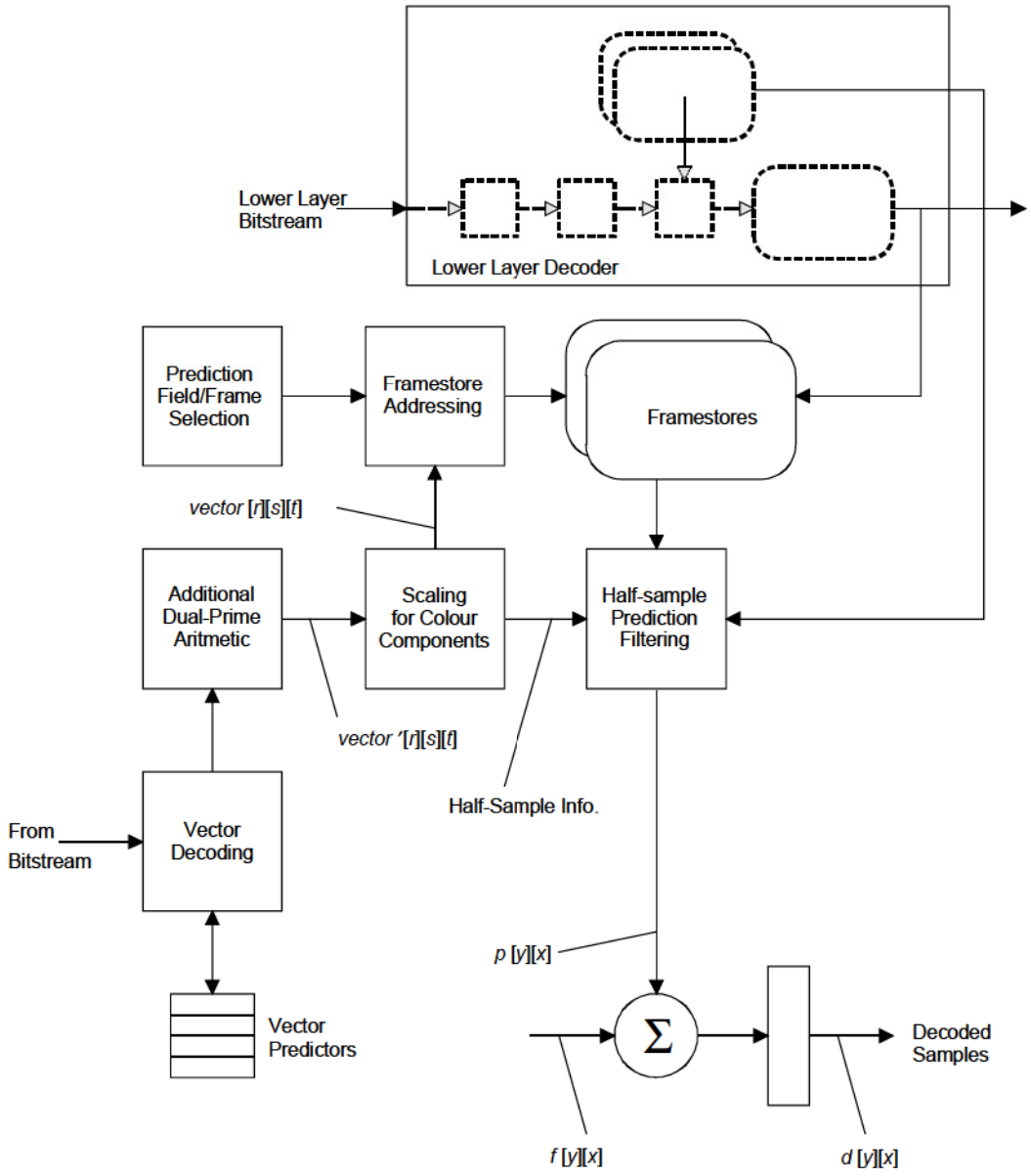



**Figure 7-16 – Simplified motion compensation process for the enhancement layer using temporal scalability**

**GOP header**

There is no restriction on GOP header (if present) to be the same as that for lower layer.

**Picture header**

There is no restriction on picture headers to be the same as in the lower layer.

**Picture coding extension**

The values in this extension can be different from the lower layer except for top_field_first, concealment_motion_vectors, and chroma_420_type and progressive_frame. The top_field_first shall be based on progressive_sequence and mux_to_progressive_sequence (see sequence_scalable_extension above) and concealment_motion_vectors shall be 0. Chroma_420_type shall be identical to the lower layer. Progressive_frame shall always have the same value as progressive_sequence.

**Picture temporal scalable extension**

This extension shall be present for each picture.

**Quant matrix extension**

This extension may be present in the enhancement layer.

### 7.9.2 Restrictions on temporal prediction

Although temporal predictions can be made from decoded pictures referenced by forward_temporal_reference or both forward_temporal_reference and backward_temporal_references, temporal scalability is efficient if predictions are formed using decoded picture/pictures from lower layer and enhancement layer that are very close in time to the enhancement picture being predicted. It is a requirement on the bitstreams that P-pictures and B-pictures shall form predictions from most recent or next pictures as illustrated by Tables 7-28 and 7-29.

In case group_of_pictures_header occurs very often in lower_layer, ambiguity can occur due to possibility of non-uniqueness of temporal references (which are reset at each group_of_pictures_header). This ambiguity shall be resolved with help of systems layer timing information.

## 7.10 Data partitioning

Data partitioning is a technique that splits a video bitstream into two layers, called partitions. A priority breakpoint indicates which syntax elements are placed in partition 0, which is the base partition (also called high priority partition). The remainder of the bitstream is placed in partition 1 (which is also called low priority partition). Sequence, GOP, and picture headers are redundantly copied in partition 1 to facilitate error recovery. The sequence_end_code is also redundantly copied into partition 1. All fields in the redundant headers must be identical to the original ones. The only extensions allowed (and required) in partition 1 are sequence_extension(), picture_coding_extension() and sequence_scalable_extension().

NOTE – The slice() syntax given in 6.2.4 is followed in both partitions up to (and including) the syntax element extra_bit_slice.

The interpretation of priority_breakpoint is given in Table 7-30.

**Table 7-30 – Priority breakpoint values and associated semantics**

| priority_breakpoint | Syntax elements included in partition zero |
|---|---|
| 0 | This value is reserved for partition 1. All slices in partition 1 shall have a priority_breakpoint equal to 0. |
| 1 | All data at the sequence, GOP, picture and slice() down to **extra_bit_slice** in slice(). |
| 2 | All data included above, plus macroblock syntax elements up to and including **macroblock_address_increment**. |
| 3 | All data included above, plus macroblock syntax elements up to but not including coded_block_pattern(). |
| 4 … 63 | Reserved. |
| 64 | All syntax elements up to and including coded_block_pattern() or DC coefficient (**dct_dc_differential**), and the first (run, level) DCT coefficient pair (or EOB). (Note) |
| 65 | All syntax elements above, plus up to 2 (run, level) DCT coefficient pairs. |
| … | |
| 63 + *j* | All syntax elements above, plus up to *j* (run, level) DCT coefficient pairs. |
| … | |
| 127 | All syntax elements above, plus up to 64 (run, level) DCT coefficient pairs. |
| NOTE – A priority_breakpoint immediately following the DC coefficient is disallowed since it might cause start code emulation. | |

Semantics of VBV remains unchanged, i.e. the VBV refers to the sum of two partitions, not any single one.

The bitstream parameters bit_rate (bit_rate_value and bit_rate_extension), vbv_buffer_size (vbv_buffer_size_value and vbv_buffer_size_extension) and vbv_delay shall take the same value in the two partitions. These parameters refer to the characteristics of the entire bitstream formed from the two partitions.

The decoding process is modified in the following manner:

– Set *current_partition* to 0, and start decoding from bitstream that contains the sequence_scalable_extension (partition 0).

– If *current_partition* = 0, check to see if the current point in the bitstream is a priority breakpoint.

  If yes, set *current_partition* to 1. Next item will be decoded from partition 1.

  Otherwise, continue decoding from partition 0. Remove sequence, GOP, and picture headers from both partitions.

– If *current_partition = 1*, check the priority breakpoint to see if the next item to be decoded is expected in partition 0.

  If yes, set *current_partition* to 0. Next item will be decoded from partition 0.

  Otherwise, continue decoding from partition 1.

An example is shown in Figure 7-17 where the priority breakpoint is set at 64 [one (run, level) pair].

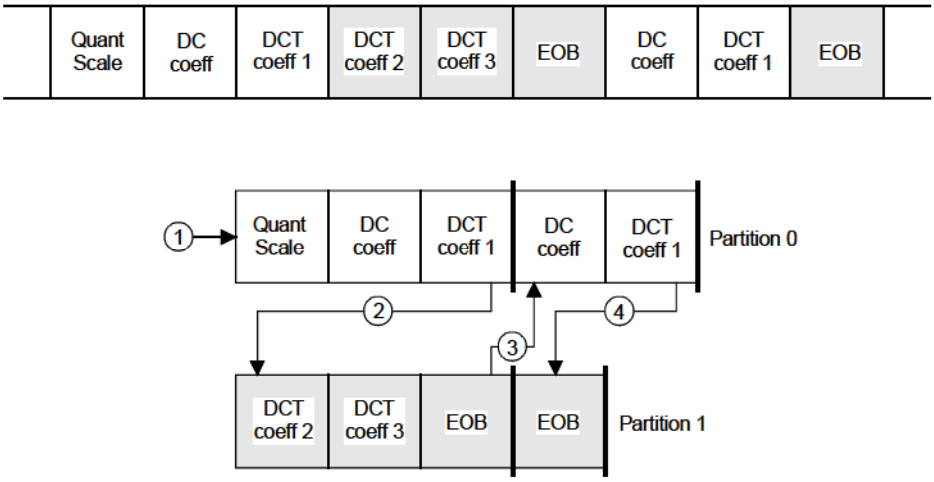



**Figure 7-17 – A segment from a bitstream with two partitions, with priority_breakpoint set to 64 (one (run, level) pair). The two partitions are shown, with arrows indicating how the decoder needs to switch between partitions**

## 7.11 Hybrid scalability

Hybrid scalability is the combination of two different types of scalability. The types of scalability that can be combined are SNR scalability, spatial scalability and temporal scalability. When two types of scalability are combined, there are three bitstreams that have to be decoded. The layers to which these bitstreams belong are named in Table 7-31.

**Table 7-31 – Names of layers**

| layer_id | Name |
|---|---|
| 0 | Base layer |
| 1 | Enhancement layer 1 |
| 2 | Enhancement layer 2 |
| … | … |

For the scalability between the enhancement layers 1 and 2, the enhancement layer 1 is its lower layer, and the enhancement layer 2 is its enhancement layer. No layer can be omitted from the hierarchical ladder. E.g. if there is SNR scalability between enhancement layer 1 and enhancement layer 2, the prediction types in enhancement layer 1 are also valid for the combined decoding process for enhancement layers 1 and 2.

The coupling of layers is more loose with spatial and temporal scalability than with SNR scalability. Therefore, in these kinds of scalability, first the base layer has to be decoded and upconverted before it can be used in the enhancement layer. In SNR scalability, both layers are decoded simultaneously. The decoding order can be summarised as follows:

*Case 1*

base layer

*<spatial or temporal scalability>*

enhancement layer 1

*<SNR scalability>*

enhancement layer 2

First decode the base layer, and then decode both enhancement layers simultaneously.

*Case 2*

base layer

*<SNR scalability>*

enhancement layer 1

*<spatial or temporal scalability>*

enhancement layer 2

First decode the base layer and the enhancement layer 1 simultaneously, and then decode the enhancement layer 2.

*Case 3*

base layer

*<spatial or temporal scalability>*

enhancement layer 1

*<spatial or temporal scalability>*

enhancement layer 2

First decode the base layer, then decode the enhancement layer 1, and finally decode enhancement layer 2.

## 7.12 Output of the decoding process

This subclause describes the output of the theoretical model of the decoding process that decodes bitstreams conforming to this Specification.

The decoding process input is one or more coded video bitstreams (one for each of the layers). The video layers are generally multiplexed by the means of a system stream that also contains timing information.

The output of the decoding process is a series of fields or frames that are normally the input of a display process. The order in which fields or frames are output by the decoding process is called the display order, and may be different from the coded order (when B-pictures are used). The display process is responsible for the action of displaying the decoded fields or frames on a display device. If the display device cannot display at the frame rate indicated in the bitstream, the display process may perform frame rate conversion. This Specification does not describe a theoretical model of display process nor the operation of the display process.

Since some of the syntax elements, such as progressive_frame, may be needed by the display process, in this theoretical model of the decoding process, all the syntactic elements that are decoded by the decoding process are output by the decoding process and may be accessed by the display process.

When the progressive sequence is decoded (progressive_sequence is equal to 1), the luminance and chrominance samples of the reconstructed frames are output by decoding process in the form of progressive frames and the output rate is the frame rate. Figure 7-18 illustrates this in the case of chroma_format equals to 4:2:0.

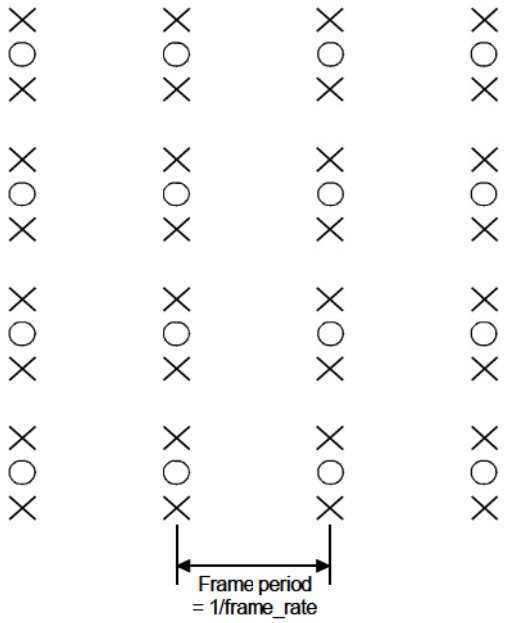



**Figure 7-18 – progressive_sequence = = 1**

The same reconstructed frame is output one time if repeat_first_field is equal to 0, and two or three consecutive times if repeat_first_field is equal to 1, depending on the value of top_field_first. Figure 7-19 illustrates this in the case of chroma_format equals to 4:2:0 and repeat_first_field equals 1.

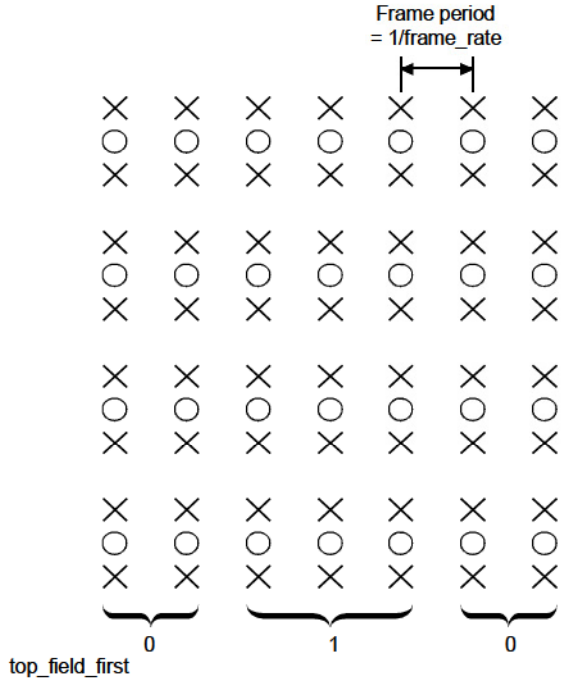



**Figure 7-19 – progressive_sequence = = '1'; repeat_first_field = = '1'**

When decoding an interlaced sequence (progressive_sequence is equal to 0), the luminance samples of the reconstructed frames are output by the decoding process in the form of interlaced fields at a rate that is twice the frame rate. Figure 7-20 illustrates this.

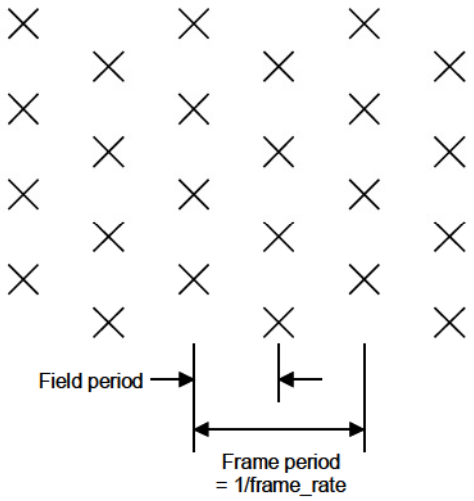



**Figure 7-20 – progressive_sequence = = 0**

It is a requirement on the bitstream that the fields at the output of the decoding process shall always be alternately top and bottom (note that the very first field of a sequence may be either top or bottom).

If the reconstructed frame is interlaced (progressive_frame is equal to 0), the luminance samples and chrominance samples are output by the decoding process in the form of two consecutive fields. The first field output by the decoding process is the top field or the bottom field of the reconstructed frame, depending on the value of top_field_first.

Although all the samples of progressive frames represent the same instant in time, all the samples are not output at the same time by the decoding process when the sequence is interlaced.

If the reconstructed frame is progressive (progressive_frame is equal to 1), the luminance samples are output by the decoding process in the form of two or three consecutive fields, depending on the value of repeat_first_field.

NOTE – The information that these fields originate from the same progressive frame in the bitstream is conveyed to the display process.

All of the chrominance samples of the reconstructed progressive frame are output by the decoding process at the same time as the first field of luminance samples. This is illustrated in Figures 7-21 and 7-22.

# 8 Profiles and levels

NOTE – In this Specification the word "profile" is used as defined below. It should not be confused with other definitions of "profile" and in particular it does not have the meaning that is defined by JTC1/SGFS.

Profiles and levels provide a means of defining subsets of the syntax and semantics of this Specification and thereby the decoder capabilities required to decode a particular bitstream. A profile is a defined subset of the entire bitstream syntax that is defined by this Specification. A level is a defined set of constraints imposed on parameters in the bitstream. Conformance tests will be carried out against defined profiles at defined levels.

The purpose of defining conformance points in the form of profiles and levels is to facilitate bitstream interchange among different applications. Implementers of this Specification are encouraged to produce decoders and bitstreams which correspond to those defined conformance regions. The discretely defined profiles and levels are the means of bitstream interchange between applications of this Specification.

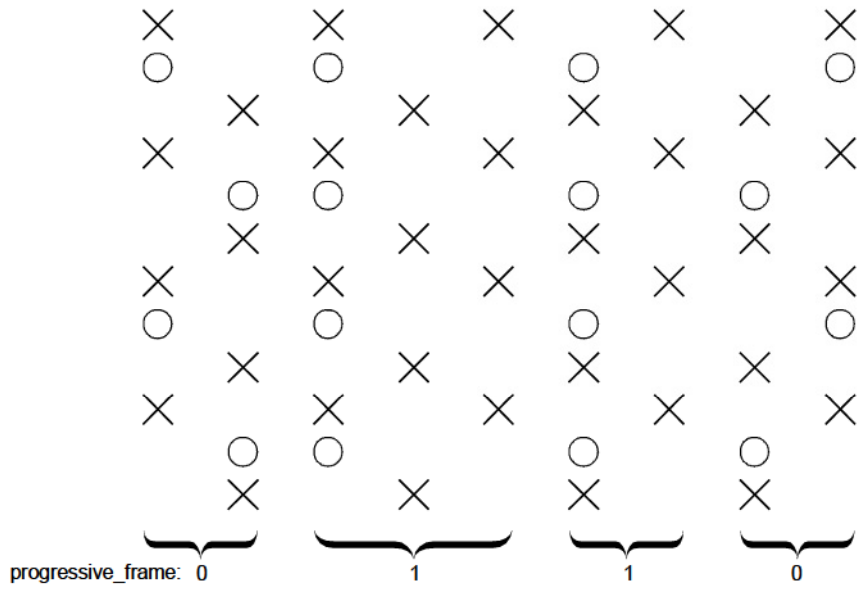



**Figure 7-21 – progressive_sequence == 0 with 4:2:0 chrominance**

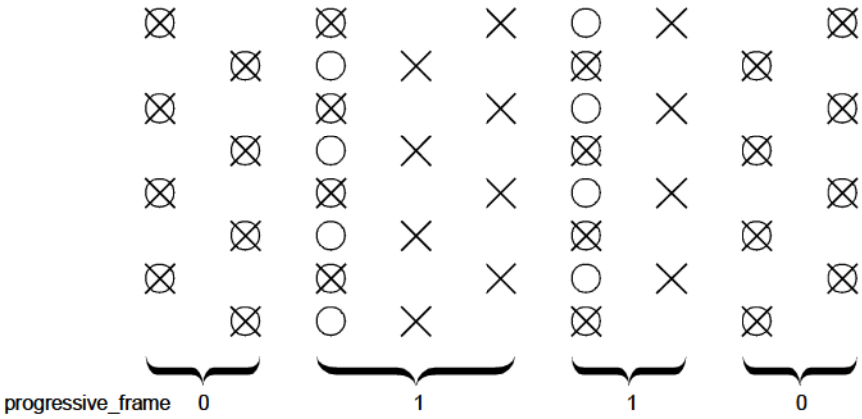



**Figure 7-22 – progressive_sequence == 0 with 4:2:2 or 4:4:4 chrominance**

In this clause the constrained parts of the defined profiles and levels are described. All syntactic elements and parameter values which are not explicitly constrained may take any of the possible values that are allowed by this Specification. In general, a decoder shall be deemed to be conformant to a given profile at a given level if it is able to properly decode all allowed values of all syntactic elements as specified by that profile at that level. One exception to this rule exists in the case of a Simple profile Main level decoder, which must also be able to decode Main profile, Low level bitstreams. A bitstream shall be deemed to be conformant if it does not exceed the allowed range of allowed values and does not include disallowed syntactic elements.

Attention is drawn to 5.4 which defines the convention for specifying a range of numbers. This is used throughout to specify the range of values and parameters.

The profile_and_level_indication in the sequence_extension indicates the profile and level to which the bitstream complies. The most significant bit of profile_and_level_indication is called 'escape bit'. When the escape bit is set to zero, the profile and level are derived from profile_and_level_indication according to Tables 8-1, 8-2 and 8-3.

**Table 8-1 – Meaning of bits in profile_and_level_indication**

| Bits | Field Size (bits) | Meaning |
|---|---|---|
| [7:7] | 1 | Escape bit |
| [6:4] | 3 | Profile identification |
| [3:0] | 4 | Level identification |

Table 8-2 specifies the profile identification codes and Table 8-3 the level identification codes. When the escape bit equals zero a profile with a numerically larger identification value will be a subset of a profile with a numerically smaller identification value. Similarly, whenever the escape bit equals zero, a level with a numerically larger identification value will be a subset of a level with a numerically smaller identification value.

**Table 8-2 – Profile identification**

| Profile identification | Profile |
|---|---|
| 110 to 111 | (Reserved) |
| 101 | Simple |
| 100 | Main |
| 011 | SNR Scalable |
| 010 | Spatially Scalable |
| 001 | High |
| 000 | (Reserved) |

**Table 8-3 – Level identification**

| Level identification | Level |
|---|---|
| 1011 to 1111 | (Reserved) |
| 1010 | Low |
| 1001 | (Reserved) |
| 1000 | Main |
| 0111 | (Reserved) |
| 0110 | High 1440 |
| 0101 | (Reserved) |
| 0100 | High |
| 0000 to 0011 | (Reserved) |

Table 8-4 describes profiles and levels when the escape bit equals 1. For these profiles and levels there is no implied hierarchy from the assignment of profile_and_level_indication and profiles and levels are not necessarily subsets of others.

Attention is drawn to Annex E, which describes in detail those parts of ISO/IEC 13818-2 that are used for a given profile and level.

NOTE 1 – On 4:2:2 Profile: The ITU-T Rec. H.262 | ISO/IEC 13818-2 compression algorithm exploits temporal redundancy, spatial redundancy, and human psycho-visual properties and is not a lossless algorithm. For sequences with substantial spatial and temporal redundancies, or without many sharp lines/edges, the quality of the sequences obtained after decompression will be higher than that obtained for sequences with lower redundancy, or with a large number of sharp lines/edges.

The 4:2:2 profile can provide higher video quality, better chroma resolution and allows a higher bit rate (at Main level, up to 50 Mbit/s) than MP@ML. It also provides the capability to encode all active lines of video.

Although it is not part of the hierarchy of profiles and levels, the 4:2:2 profile @ Main level decoder is required to decode all the bit streams decodable by MP@ML decoders.