CCITT SG XV
Working Party XV/1
Experts group on ATM Video Coding

Document, AVC-212
March 1992

INTERNATIONAL ORGANISATION FOR STANDARDISATION

ORGANISATION INTERNATIONALE DE NORMALISATION

ISO-IEC/JTC1/SC29/WG11

CODED REPRESENTATION OF PICTURE AND AUDIO INFORMATION

ISO-IEC/JTC1/SC29/WG11
MPEG 92/086

Title: Preliminary Working Draft (PWD)

Source: PWD Editing Committee

Editorial Comments

Some times it was not so clear what was decided in Singapore. In these situations the Ad-Hoc group wrote down the best solution it could think of, which does not nessarily represent the thought of the MPEG or the CCITT Experts group.

Some questions and concerns are listed below:

- How are chrominance samples handle in the Field DCT coding mode?
- How are chrominance sample motion compensated in the Field prediction mode?
- Is the Chrominance sample position issue resolved? This also relates to the question above.
- In section 5.7.1 full search is defined as part of the motion estimation, was this the outcome of the Singapore meeting?
- Field/frame coding on intra blocks was not indictated in MPEG92/80, a text is given is section 6.2.

Arian Koster
PTT Research
Tel: +31 70 332 5664
Fax: +31 70 332 6477
Email: a.koster@research.ptt.nl

1 INTRODUCTION

This document gives a comprehensive description of the MPEG-2 Test Model (TM). This model is used in the course of the research for comparison purposes.

In order to obtain results for comparison this document describes some techniques that are not a matter of standardisation. Some of these techniques are of debatable value but are included to provide a common basis for comparisons. In order to have comparable simulation results the methods described in this document are therefore mandatory.

The readers are asked to give comments and corrections to remove ambiguous parts.

In several places this TM will be different from what is possible with CD 11172-2, sections or paragraphs describing this are marked in the margin.

2 GENERAL CODEC OUTLINE

The generic structure is depicted in figures 2.1 and 2.2. The generic structure of the reference model is based on the following main issues:

- input / output format CCIR 601
- Pre- and post processing, as described in section 3.
- Random access of coded pictures, which requires the definition of Group of pictures, as described in Section 4 and 6.
- Motion Vector search in forward and/or backward direction, as described in Section 5.
- Prediction modes, forward, backward and bi-directional motion compensated, field or frame motion compensation, as described in section 6.
- DCT, on frames or fields as described in section 7
- Entropy coding, as described in section 8.
- 4 Mbps and 9 Mbps target rate, including multiplexing and regulation, as described in section 9 an 10 respectively.
- Scalable bit streams as described in Appendix D.
- Extensions for purely field based coding are given in Appendix E.
- Experiments for cell loss are given in Appendix F.

2.1 Arithmetic Precision

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

(a) Where arithmetic precision is not specified, such as in the calculation of DCT transform coefficients, the precision should be sufficient so that significant errors do not occur in the final integer values

(b) The operation / specifies integer division with truncation towards zero. For example, 7/4 is truncated to 1, and -7/4 is truncated to -1.

(c) The operation // specifies 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.

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

Figure 2.1: Encoder Block Diagram

Figure 2.2: Decoder block diagram

3 SOURCE FORMATS

This section gives a description of the Source Formats and their conversion from and to CCIR 601. For the purposes of the simulation work, only the particular formats explained in this section will be used.

3.1 Source Input Formats

The SIF's consists of component coded video Y, Cb and Cr. The simulated algorithm uses two source input formats for moving pictures. The differences are the number of lines, the frame rate and the pixel aspect ratio. One is for 525 lines per frame and 60Hz, the other one is for 625 lines per frame and 50Hz.

The parameters for the so called active 4:2:0-525-format and active 4:2:0-625-format frames are:

	4:2:0-525	4:2:0-625
Number of active lines		
Luminance (Y)	480	576
Chrominance (Cb,Cr)	240	288
Number of active pixels per line		
Luminance (Y)	704	704
Chrominance (Cb,Cr)	352	352
Frame rate (Hz)	30	25
Frame aspect ratio (hor:ver)	4:3	4:3

Table 3.1: Active 4:2:0 Formats

For compatibility with MPEG1 or scalability a second set of formats is defined, the MPEG1 SIF. The parameters for the so called active SIF-525 and active SIF-625 frames are:

	SIF525	SIF625
Number of active lines		
Luminance (Y)	240	288
Chrominance (Cb,Cr)	120	144
Number of active pixels per line		
Luminance (Y)	352	352
Chrominance (Cb,Cr)	176	176
Frame rate (Hz)	30	25
Frame aspect ratio (hor:ver)	4:3	4:3

Table 3.2: Active SIF Format

For compatibility with H.261 a third format is defined, the Common Intermediate Format (CIF). The parameters for the so called active CIF are:

	CIF
Number of active lines	
Luminance (Y)	288
Chrominance (Cb,Cr)	144
Number of active pixels per line	
Luminance (Y)	352
Chrominance (Cb,Cr)	176
Frame rate (Hz)	30
Frame aspect ratio (hor:ver)	4:3

Table 3.3: Active CIF Format

When scalable extensions are used, a hierarchy of formats can exist, with the highest resolution equal to the CCIR 601 Active 4:2:0 format, and with lower resolutions having either 1/2, 1/4, or 1/8, the number of pixels in each row and column.

3.2 Definition of fields and frames

The CCIR 601 and the active 4:2:0 formats are both interlaced. A frame in these formats consists of two fields. The two fields are merged in one frame. The odd lines are within one field the even lines in the other field. There is a sampling time difference between the two fields. Let us define FIELD1 as the field preceding FIELD2.

1. Video data is 50 (50 Hz) or 60 (60 Hz) fields per second. The first field is the odd field, and is numbered field 1. The second field is the even field and is numbered field 2 and so on. So odd numbered fields are odd fields and even numbered fields are even fields.

2. 50 Hz fields have 288 lines each, and 60 Hz fields have 240 lines each. The fields are considered to be interlaced, and the first line of the first (odd) field is above the first line of the second (even) field for both 50 and 60 Hz.

3. The field lines are numbered as if they are combined into a frame, and the numbering starts at one. So, the first line of the frame, which is the first line of the first (odd) field is line 1. The second line of the frame which is the first line of the second (even) field is line 2. And so on, so odd numbered lines are in odd fields, and even numbered lines are in even fields.

4. For display of 50 Hz material, the 288 lines are the active 288 lines of that format. This will display correctly since in that format, the first line of the first field is above the first line of the second field.

5. For display of 60 Hz material, the 240 lines are placed in a specific set of the 243 active lines of that format. The first (odd) field is displayed on lines 21, 23, ..., 499, and the second (even) field is displayed on lines 22, 24, ..., 500. The active lines 19, 501, and 503 of the odd fields and the active lines 18, 20, and 502 of the even fields are displayed as black.

3.3 Conversion of CCIR 601 to the Input formats

3.3.1 Conversion of CCIR 601 to the 4:2:0 format

Pre processing is applied to convert the CCIR 601 format to the 4:2:0 format. This is described in the following.

First the signal is cropped from 720 luminance pels per line to 704 pels per line by removing 8 pels from the left and 8 pels from the right. Similarly the 360 chrominance pels per line are cropped to 352 pels per line by removing 4 pels from the left and 4 pels from the right.

Luminance: two fields are merged in their geometrical order to form a frame.
Remark: Some processing in the running of the coding scheme is however field based (DCT coding, prediction); thus it is still needed to know for each line of pixels which field it originates from.

Chrominance: The following 7 tap vertical filter is used to pre-filter the FIELD1
[-29, 0, 88, 138, 88, 0, -29] /256
Then, vertical sub sampling by 2 is performed.

The following 4-tap vertical filter is used to decimate the FIELD2:
[1, 7, 7, 1]/16
Then, vertical sub sampling by 2 is performed.

The two sub sampled chrominance fields are merged to form a frame. This is shown in figure 3.1.

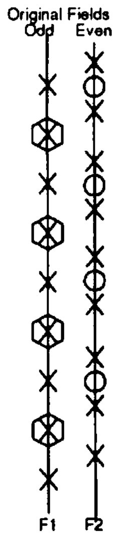

Figure 3.1: 4:2:0 Chrominance sub sampling in the fields

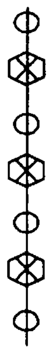

Figure 3.2 : 4:2:0 Chrominance sub sampling in a frame

NOTE: The horizontal positions of the chrominance sample is wrong.

In figure 3.1 and 3.2 the following symbols are used:

X	the vertical position of the original lines
O	the vertical position of lines of the sub sampled odd field
⊗	the vertical position of lines of the sub sampled even field

3.3.2 Conversion of CCIR 601 to SIF

The CCIR 601 formats are converted into their corresponding SIFs by sampling odd fields and using the decimation filter of table 3.4.

```
       720
+----------------+               720                       360
|                | 480    +--------------+ 240      +--------+240
|                | /576   |              | /288     |        |/288
|                |        |              |          |        |
|       Y        | +------>|              +--------> |        |
|                |         |              |Horizontal|        |
|                |         |              |Decimation|        |
|                |         |              | Filter   |        |
|                |         +--------------+          +--------+
+----------------+
    CCIR 601                Odd field only             SIF
     525/625


    360
+---------+          360              180
|         | 480  +---------+ 240      +----+ 240       180
|         | /576 |         | /288     |    | /288     +----+ 120
|   U,V   +----->|         +---------->|    +----->    |    | 144
|         |      |         |Horizontal|    |Vertical  |    |
|         |      |         |Decimation|    |Decimation+----+
|         |      +---------+  Filter  +----+Filter
+---------+
  CCIR 601       Odd field only                       SIF
  525/625
```

Figure 3.3 Conversion from CCIR 601 into SIF

The filter coefficients are depicted in table 3.4.

-29	0	88	138	88	0	-29	//256

Table 3.4 Decimation filter

Note: the odd fields contain the top most full line

3.4 Conversion of the Input Formats to CCIR 601

3.4.1 Conversion of the 4:2:0 Format to CCIR 601

Luminance samples of each 4:2:0 field are copied to the corresponding CCIR 601 field.

Chrominance samples are not horizontally resampled.

Vertical resampling of the chrominance is done differently on field 1 and field 2 because of the different locations of the chrominance samples.

In field 1, the chrominance samples in the CCIR 601 field are obtained by interpolating the chrominance samples in field 1 only of the 4:2:0 format. Referring to line numbers defined in the 4:2:2 frame, samples on lines 1, 5, 9 etc. are copied from the corresponding lines in the 4:2:0 field. Samples on lines 3, 7, 11 etc. are interpolated by the even tap filter [1, 1]//2 from the corresponding adjacent lines in the 4:2:0 field.

In field 2, the chrominance samples in the CCIR 601 field are obtained by interpolating the chrominance samples in field 2 only of the 4:2:0 format. Referring to line numbers defined in the 4:2:2 frame, samples on lines 2, 6, 10 etc. are interpolated from the corresponding adjacent lines in the 4:2:0 field using a [1, 3]//4 filter. Samples on lines 4, 8, 12 etc. are interpolated by a [3, 1]//4 filter from the corresponding adjacent lines in the 4:2:0 field.

3.4.2 Conversion of SIF to CCIR 601

A SIF is converted to its corresponding CCIR 601 format by using the interpolation filter of table 3.5.

```
                                                 720
    360                   720              +----------------+
+--------+ 240    +--------------+ 240    |                |
|        | /288   |              | /288   |                |
|        +------> |              +------->|       Y        | 480
|        | Horiz  |              |Vertical|                | 576
|        | Interp.|              | Interp |                |
+--------+ Filt   +--------------+ Filter |                |
                                          +----------------+
SIF 525/625
                                                 CCIR 601

                                                 360
              180               360              +--------+
 180   120   +----+ 240       +-------+ 240      |        |
+----+ /144  |    | /288      |       | /288     |        | 480
|   +------->|    +---------> |       +--------->|  U,V   | /576
|   |        |    |Horizontal |       | Vertical|        |
+----+Vertical|    |Interp     |       | Interp   |        |
     Interp  +----+ Filter    +-------+ Filter   |        |
     Filter                                      +--------+

SIF 525/625                                        CCIR 601
```

Figure 3.4: Conversion of SIFs to CCIR 601 formats

The filter coefficients are shown in table 3.6.

-12	0	140	256	140	0	-12	//256

Table 3.5 Interpolation filter

Note: the active pel area should be obtained from the significant pel area by padding a black level around the border of the significant pel area.

4 LAYERED STRUCTURE OF VIDEO DATA

4.1 Sequence

A sequence consists of one or more concatenated Groups of Pictures.

4.2 Group of pictures

A Group of Pictures consists of one or more consecutive picture. The order in which pictures are displayed differs from the order in which the coded versions appear in the bit stream. In the bit stream, the first frame in a Group of Pictures is always an intra frame. In display order, the last picture in a Group of Pictures is always an intra or predicted picture, and the first is either an intra picture or the first bidirection picture of the consecutive series of bi-directional pictures which immediately precedes the first intra picture.

4.3 Picture

Pictures can be intra, predicted, or interpolated pictures (known as I-pictures, P-pictures, and B-pictures - see section 6.1). The arrangement of pictures, in display order, in a Group of Pictures of this TM is shown in Figure 4.1. In the figure frames 1-15 are part of a Group of Picture.

```
$$$$$$$$include figures here
```

Figure 4.1 Structure of a Group of Pictures

```
+-1-------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+---------------------------------+
+-30------------------------------+
+ - - - - - - - - - - - - - - - - +
+---------------------------------+
+-36------------------------------+
+---------------------------------+
```

Figure 4.2 Arrangement of Slices in a Picture

For the purposes of simulation, each frame consists of 30 or 36 Macro block Slices (MBS, see section 4.4). 4:2:0-525 has 30 MBSs and 4:2:0-625 has 36. These MBSs cover the significant pel area. The arrangement of these MBSs in a frame is shown in figure 4.2.

4.4 Macro block Slice

A Macroblock Slice consists of a variable number of macroblocks. A Macroblock Slice can start at any MB and finish at any other MB in the same frame. In this Test Model, a Macroblock Slice consists of a single row of 44 Macroblocks, beginning at the left edge of the picture, and ending at the right edge.

```
+----+----+----+----+----+----+---      ---+----+----+----+----+
|  1 |  2 |  3 |  4 |  5 |  6 |            | 41 | 42 | 43 | 44 |
+----+----+----+----+----+----+---      ---+----+----+----+----+
```

Figure 4.3: Test model Macroblock Slice Structure

When scalable extensions are used (Annex D), the Slice layer may contain Macroblocks of resolution lower than 16x16.

4.4.1 Slave Slice (scalable extension)

Slave slices are layers of Slave macroblocks which are spatially co-located with the Macroblocks in the Slice layer.

4.5 Macroblock

A Macroblock consists of 6 blocks. The general structure holds 4 Y, 1 Cb and 1 Cr Blocks and is depicted in figure 4.4.

```
+---+---+
| 1 | 2 |    +---+    +---+
+---+---+    | 5 |    | 6 |
| 3 | 4 |    +---+    +---+
+---+---+

    Y          Cb       Cr
```

Figure 4.4: General Macroblock structure

The internal organisation within the Macroblock is different for Frame and Field DCT coding, and is depicted for the luminance blocks in figure 4.5 and 4.6. The chrominance block is in frame order for both DCT coding macroblock types.

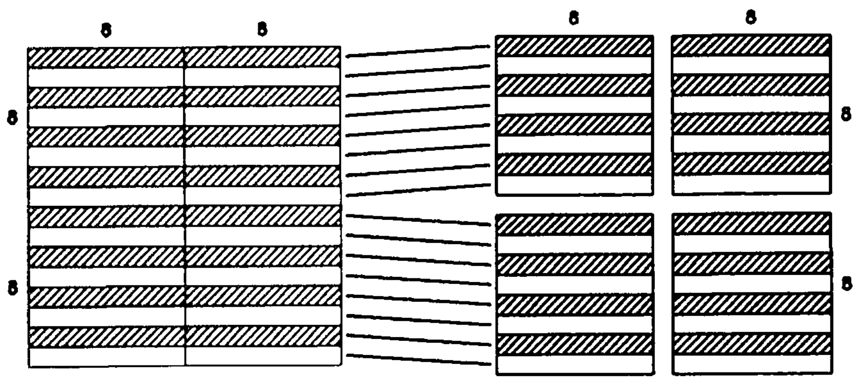

Figure 4.5 : Luminance Macroblock Structure in Frame DCT Coding

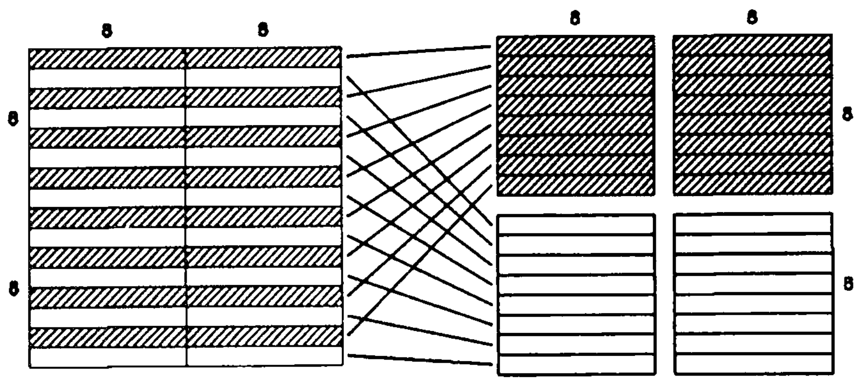

Figure 4.6 : Luminance Macroblock Structure in Field DCT Coding

When scalable extensions are used (Annex D), Macroblocks may contain scaled_blocks of resolution lower than 8x8.

4.5.1 Slave_macroblock (scalable extension)

Slave_macroblocks are layers of slave_blocks which are spatially co-located with the scaled_blocks in the Macroblock layer.

4.6 Block

A Block consists of an array of 8x8 coefficients. Figure 4.7 shows coefficients in the block in zigzag scanned order.

```
+--+--+--+--+--+--+--+--+
| 1| 2| 6| 7|15|16|28|29|    +------> increasing cycles per
+--+--+--+--+--+--+--+--+    |        picture width
| 3| 5| 8|14|17|27|30|43|    |
+--+--+--+--+--+--+--+--+    |
| 4| 9|13|18|26|31|42|44|    |
+--+--+--+--+--+--+--+--+    V
|10|12|19|25|32|41|45|54|
+--+--+--+--+--+--+--+--+    increasing cycles per
|11|20|24|33|40|46|53|55|    picture height
+--+--+--+--+--+--+--+--+
|21|23|34|39|47|52|56|61|
+--+--+--+--+--+--+--+--+
|22|35|38|48|51|57|60|62|
+--+--+--+--+--+--+--+--+
|36|37|49|50|58|59|63|64|
+--+--+--+--+--+--+--+--+
```

Figure 4.7: Block structure

4.6.1 Scaled_block (scalable extension)

When scalable extensions are used (Annex D), a scaled_block is used instead of a block. A scaled_block may consist of an array of NxN coefficients, where N is 1, 2, 4, or 8".

4.6.2 Slave_block (scalable extension)

Slave_blocks are arrays of coefficients, which are used to enhance the spatial or amplitude resolution of the coefficients in the corresponding Scaled_block layer. Figure 4.8 shows the Scaled_block and Slave_block structures that are possible in a scalable bit stream.

```
Block_1  Block_2        Block_4                      Block_8
 +--+    +--+--+    +--+--+--+--+    +--+--+--+--+--+--+--+--+
 | 1|    | 1| 2|    | 1| 2| 6| 7|    | 1| 2| 6| 7|15|16|28|29|
 +--+    +--+--+    +--+--+--+--+    +--+--+--+--+--+--+--+--+
         | 3| 4|    | 3| 5| 8|13|    | 3| 5| 8|14|17|27|30|43|
         +--+--+    +--+--+--+--+    +--+--+--+--+--+--+--+--+
                    | 4| 9|12|14|    | 4| 9|13|18|26|31|42|44|
                    +--+--+--+--+    +--+--+--+--+--+--+--+--+
                    |10|11|15|16|    |10|12|19|25|32|41|45|54|
                    +--+--+--+--+    +--+--+--+--+--+--+--+--+
                                     |11|20|24|33|40|46|53|55|
                                     +--+--+--+--+--+--+--+--+
                                     |21|23|34|39|47|52|56|61|
                                     +--+--+--+--+--+--+--+--+
                                     |22|35|38|48|51|57|60|62|
                                     +--+--+--+--+--+--+--+--+
                                     |36|37|49|50|58|59|63|64|
                                     +--+--+--+--+--+--+--+--+
```

Figure 4.8: Block structures for scalable bit streams

5 MOTION ESTIMATION AND COMPENSATION

To exploit temporal redundancy, motion estimation and compensation are used for prediction.

Prediction is called forward if reference is made to a frame in the past (in display order) and called backward if reference is made to a frame in the future. It is called interpolative if reference is made to both future and past.

For this TM the search range should be appropriate for each sequence, and therefore a vector search range per sequence is listed below:

Table Tennis:	±15 pels/frame	
Flower Garden	±15 pels/frame	
Calendar	±15 pels/frame	
Popple	±15 pels/frame	
Football	±31 pels/frame	
PRL CAR	±31 pels/frame	**[NOTE: isn't 63 pels/frame more appropriate?]**

A positive value of the horizontal or vertical component of the motion vector signifies that the prediction is formed from pixels in the referenced frame, which are spatially to the right or below the pixels being predicted.

5.1 Motion Vector Estimation

For the P and B-frames, two types of motion vectors, Frame Motion Vectors and Field Motion Vectors, will be estimated for each macroblock. In the case of Frame Motion Vectors, one motion vector will be generated in each direction per macroblock, which corresponds to a 16x16 pels luminance area. For the case of Field Motion Vectors, two motion vectors per macroblock will be generated for each direction, one for each of the fields. Each vector corresponds to a 16x8 pels luminance area.

The algorithm uses two steps. First a full search algorithm is applied on original pictures with full pel accuracy. Second a half pel refinement is used, using the local decoded picture.

5.1.1 Full Search

A simplified Frame and Field Motion Estimation routine is listed below. In this routine the following relation is used:

(AE of Frame) = (AE of FIELD1) + (AE of FIELD2)

where AE represents a sum of absolute errors.

With this routine three vectors are calculated, MV_FIELD1, MV_FIELD2 and MV_FRAME.

```
Min_FIELD1 = MAXINT;
Min_FIELD2 = MAXINT;
for (y = -YRange; y < YRange; y++) {
    for (x = -XRange; x < XRange; x++) {
        AE_FIELD1 = AE_Macroblock(prediction_mb(x,y),
                                  lines_of_FIELD1_of_current_mb);
        AE_FIELD2 = AE_Macroblock(prediction_mb(x,y),
                                  lines_of_FIELD2_of_current_mb);
        AE_FRAME = AE_FIELD1 + AE_FIELD2;
        if (AE_FIELD1 < Min_FIELD1) {
            MV_FIELD1 = (x,y);
            Min_FIELD1 = AE_FIELD1;
        }
        if (AE_FIELD2 < Min_FIELD2) {
            MV_FIELD2 = (x,y);
            Min_FIELD2 = AE_FIELD2;
        }
        if (AE_FRAME < Min_FRAME) {
            MV_FRAME = (x,y);
            Min_FRAME = AE_FRAME;
        }
    }
}
```

The search is constrained to take place within the boundaries of the significant pel area. Motion vectors which refer to pixels outside the significant pel area are excluded.

5.1.2 Half pel search

The half pel refinement uses the eight neighbouring half-pel positions in the corresponding local decoded field or frame which are evaluated in the following order:

1	2	3
4	0	5
6	7	8

where 0 represents the previously evaluated integer-pel position. The value of the spatially interpolated pels are calculated as follows:

$$S(x+0.5,y) = (S(x,y)+S(x+1,y))//2,$$
$$S(x,y+0.5) = (S(x,y)+S(x,y+1))//2,$$
$$S(x+0.5,y+0.5) = (S(x,y)+S(x+1,y)+S(x,y+1)+S(x+1,y+1))//4.$$

where x, y are the integer-pel horizontal and vertical coordinates, and S is the pel value. If two or more positions have the same total absolute difference, the first is used for motion estimation.

5.2 Motion Compensation

Motion compensation is performed differently for field coding and for frame coding. General formulas for frame and field coding are listed below.

Forward motion compensation is performed as follows:

$$S(x,y) = S_1(x + FMV_x(x,y), y + FMV_y(x,y))$$

Backward motion compensation is performed as follows:

$$S(x,y) = S_{M+1}(x + BMV_x(x,y), y + BMV_y(x,y))$$

Temporal interpolation is performed by averaging.

$$S(x,y) = (S_1(x + FMV_x(x,y) , y + FMV_y(x,y)) + S_{M+1}(x + BMV_x(x,y), y + BMV_y(x,y)))//2$$

A displacement vector for the chrominance is derived by halving the component values of the corresponding MB vector, using the formula from CD 11172:

```
right_for = (recon_right_for / 2) >> 1;
down_for = (recon_down_for / 2) >> 1;
right_half_for = recon_right_for/2 - 2*right_for;
down_half_for = recon_down_for/2 - 2*down_for;
```

5.2.1 Frame Motion Compensation

In frame prediction macroblocks there is one vector per macroblock. Vectors measure displacements on a frame sampling grid. Therefore an odd-valued vertical displacement causes a prediction from the fields of opposite parity. Vertical half pixel values are interpolated between samples from fields of opposite parity. Chrominance vectors are obtained directly by using the formulae above. The vertical motion compensation is illustrated in figure 5.1.

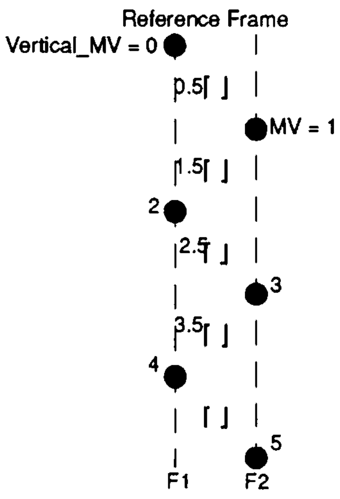

Figure 5.1: Frame Motion Compensation

5.2.2 Field Motion Compensation

In field prediction macroblocks there are two vectors per macroblock. The first vector refers to blocks in field 1 and the second vector refers to blocks in field 2.

```
if ( nint(vertical_motion_vector+0.25) == EVEN)
        reference is made to the same parity field
else
        reference is made to the opposite parity field
```

The integer part of the vector measures displacements on a frame sampling grid. Therefore an odd-valued vertical displacement causes a prediction from a field of opposite parity. Vertical half pixel values are interpolated between samples from fields of the same parity.

Chrominance vectors are indeterminate and the exercise is left to the reader. You could try and do the same as in frame mode.

In the field coding mode, the half pel interpolation is performed with reference to a single field. Vectors with 0.5 pel accuracy result in a linear interpolation of the two or four corresponding pixels, in the reference field. The motion compensation is shown in figures 5.2 and 5.3.

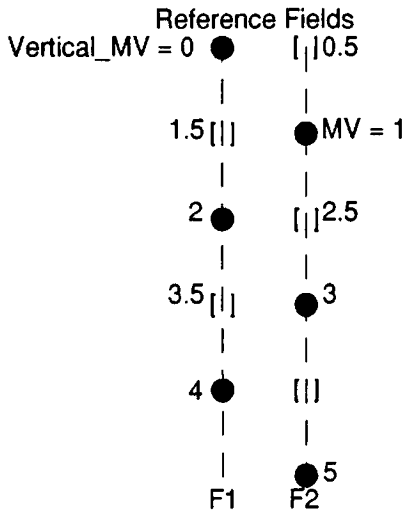

Figure 5.2: Motion Compensation for FIELD1

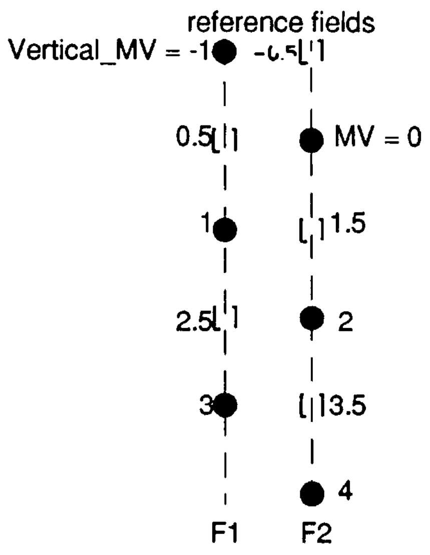

Figure 5.3: Motion Compensation for FIELD2

6 MODES AND MODE SELECTION

In section 6.1, a coding structure with different frame modes is introduced. Within each frame, macroblocks may be coded in several ways, thus aiming at high coding efficiency. The MB modes for intra, predicted and interpolated frames are shown in 6.2 to 6.4.

6.1 Picture types

Pictures are coded in several modes as a trade-off between coding efficiency and random accessibility. There are basically three picture coding modes, or picture types:
- I-pictures: intra coded pictures.
- P-pictures: forward motion-compensated prediction pictures.
- B-pictures: motion compensated interpolation pictures.

Although, in principle, freedom could be allowed for choosing one of these methods for a certain picture, for the Test model a fixed, periodic structure is used depending on the respective picture.

Every N-th frame of a sequence starting with the first frame is coded as intra frame i.e. frames 1, N+1, etc. (see Fig. 5.1). Following every M-th frame in between (within a Group of Pictures) is a predicted frame coded relative to the previous predicted or intra frame. The interpolated frames are coded with reference to the two closest previous and next predicted or intra frames. In this TM, M=3 and N=15 for 29.97 Hz and M=3 and N=12 for 25 Hz.

[NOTE: exact values for M and N have to be defined, for different experiments]

The following parameters are currently to be used for core- experiments:

Frame rate	25 Hz		29.97 Hz	
N		12		15
M		3		3

Coding modes available for predicted and interpolated frames are described in detail in the following paragraphs.

6.2 Macroblock types in an intra picture

In an I-picture the following macroblock types are provided:
- Intra
- Intra with modified quantizer

See also table B.2a

Independent of the macroblock type a compatible prediction and field/frame DCT coding indications are given in the bit stream, see also chapter 9, the macroblock layer section.

The macroblock type selection is done in the following order:
- Compatible prediction
- Field/frame DCT coding
- Modified quantizer

6.3 Macroblock types in a predicted picture

In predicted frames the following macroblocks types can be distinguished:
- Motion compensation coded
- No motion compensation coded
- Motion compensation not coded
- Intra
- Motion compensation coded with modified quantizer
- No Motion compensation coded with modified quantizer
- Intra with modified quantizer

See also table B.2b

Independent of the macroblock type a compatible prediction, field/frame DCT coding and field/frame motion vector prediction indications are given in the bit stream, see also chapter 9, the macroblock layer section.

Macroblock type selection is done in the following order:
- MC/no MC - Field/Frame prediction
- Compatible prediction
- Intra/Inter
- Modified quantizer
- Field/frame DCT coding
- Coded/not Coded

6.4 Macroblock types in an interpolated picture

In interpolated frames the following macroblock types are provided:
- Interpolate, not Coded
- Interpolate, Coded
- Backwards, not Coded
- Backwards, Coded
- Forwards, not Coded
- Forwards, Coded
- Intra
- Interpolate with modified quantizer
- Backwards with modified quantizer
- Forwards with modified quantizer
- Intra with modified quantizer

Independent of the macroblock type a compatible prediction, field/frame DCT coding and field/frame motion vector prediction indications are given in the bit stream, see also chapter 9, the macroblock layer section.

Macroblock type selection is done in the following order:
- Interpolative/Forwards/Backwards - Frame/Field prediction
- Compatible prediction
- Intra/Inter
- Modified quantizer
- Field/frame DCT coding
- Coded/not Coded

6.5 Selection criteria

The following rules apply to interlace and compatible bit streams. A subset of them apply to scalable bit streams as well. For details refer to Annex D.

6.5.1 Motion Compensation/No Motion Compensation - Frame/Field

For P-frames, the decision of selecting the Frame Motion Vector or the Field Motion Vector is by MSE comparison of each error signal: If (SE of Frame) <= (SE of FIELD1 + SE of FIELD2) then the Frame Motion Vector is chosen.

The decision of MC/no MC will be SE based. If (SE of MC) < (SE of No MC) then MC mode.

6.5.2 Forward/Backward/Interpolative - Field/Frame prediction

Each B-frame Macroblock has possible Forward/Backward/Interpolated modes, and each mode has further Frame/Field prediction mode, so there totally 6 possible modes. All SE of the error signals of each mode will be calculated, and the mode with the least SE is chosen. In the case of two modes having the same SE, the mode with Frame prediction only will have higher priority, and also forward prediction will have higher priority than backward with interpolated mode the least priority.

6.5.3 Compatible prediction

When the experiment is not intended for compatible coding, this mode is not selected. When the experiment is intended for compatible coding the following criterion is used.

The SE of the compatible prediction is compared with the SE of the outcome of the previous decision. The best one is selected.

6.5.4 Intra/Inter coding

The implementation of the intra/non-intra decision is based on the comparison of VAR and VAROR as computed in the following algorithm:

```
for (i = 1; i <= 16; i++) {
        for (j = 1; j <= 16; j++) {
                OR  = O(i,j);
                Dif  = OR - S(i,j);
                VAR  = VAR + Dif*Dif;
                VAROR=VAROR + OR*OR;
                MWOR =MWOR + OR;
        }
}
VAR = VAR/256;
VAROR=VAROR/256 - MWOR/256*MWOR/256;
```

Where: O(i,j) denotes the pixels in the original macroblock. S(i,j) denotes the pixels of the reconstructed macroblock, in the frame referred to by the motion vector. Full arithmetic precision is used. The characteristics of the decision are described in Fig. 6.1. (Non-intra decision includes the solid line in Fig. 6.1.)

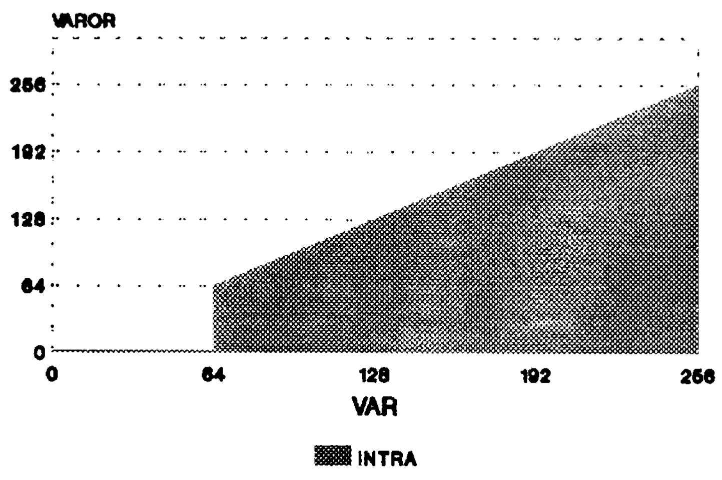

Figure 6.1: Characteristic Intra/Inter

6.5.5 Modified Quantizer

In chapter 10 "Rate control and quantization control" the algorithm for the calculation of the quantizer is given. If the quantizer given by this algorithm is not equal to the previous quantizer the modified quantizer indication is used.

6.5.6 Field/Frame DCT coding

6.5.6 Field/Frame DCT coding decisions

Field based rather than frame based coding is used if the following equation holds:
Var1 => (Var2 + 4096)
where Var1 and Var2 are calculated with the following lines:

```
Var1 = 0;
Var2 = 0;
for (Pix = 0; Pix < 16; Pix++) {
    Sum = 0;
    for (Line = 0; Line < 16; Line += 2) {
        Sum += O(Pix, Line) - O(Pix, Line+1);
    }
    Var1 += Sum * Sum;
    Sum = 0;
    for (Line = 0; Line < 16; Line += 4) {
        Sum += O(Pix,Line) + O(Pix,Line+1) - O(Pix,Line+2) - O(Pix,Line+3);
    }
    Var2 += Sum * Sum;
}
```

where O(Pix, Line) denotes the 16 x 16 macroblock to be transformed.

6.5.7 Coded/Not Coded

The choice of coded or not coded is a result of quantization. When all coefficients are zero then a block is not coded. A Macroblock is not coded if no block in it is coded, else it is coded.

7 TRANSFORMATION AND QUANTIZATION

While mode selection and local motion compensation are based on the macroblock structure, the transformation and quantization is based on 8*8 blocks.

Blocks are transformed with a 2-dimensional DCT as explained in Appendix A. Each block of 8*8 pixels thus results in a block of 8*8 transform coefficients. The DCT coefficients are quantized as described in sections 7.1 and 7.2.

7.1 Quantization of Intra frames and Intra Macroblocks

Intra frame DCT coefficients are quantized with a uniform quantizer without a dead-zone.

7.1.1 DC Coefficients

The quantizer step-size for the DC coefficient of the luminance and chrominance components is always 8. Thus, the quantized DC value, QDC, is calculated as:

QDC = dc//8

where "dc" is the 11-bit unquantized mean value of a block.

7.1.2 AC Coefficients

AC coefficients ac(i,j) are first scaled by individual weighting factors,

$ac\sim(i,j) = (16 * ac(i,j)) // w_I(i,j)$

where $w_I(i,j)$ is the (i,j)th element of the Intra quantizer matrix given in figure 7.1. ac~(i,j) is limited to the range [-2048,2047].

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Figure 7.1 - Intra quantizer matrix

The step-size for quantizing the scaled DCT coefficients, ac~(i,j), is derived from the quantization parameter, mquant. Mquant is calculated for each macroblock by the algorithm defined in Section 10 and is stored in the bit stream in the slice header and, optionally, in any macroblock (see Section 9 for the syntax of the bit-stream and Section 10 for the calculation of mquant in the encoder).

The quantized level QAC(i,j) is given by:

QAC(i,j) = ac~(i,j) // (2*mquant)

and QAC(i,j) is limited to the range [-256..255].

7.2 Quantization of Predicted and Interpolated frames

Non-intra macroblocks in Predicted and Interpolated frames are quantized with a uniform quantizer that has a dead-zone about zero. A non-intra quantizer matrix, given in figure 7.2, is used.

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

Figure 7.2 - Non-intra quantizer matrix

The step-size for quantizing both the scaled DC and AC coefficients is derived from the quantization parameter, mquant. Mquant is calculated for each macroblock by the algorithm defined in Section 10. The following formulae describe the quantization process. Note that INTRA type macroblocks in predicted and interpolated frames are quantized in exactly the same manner as macroblocks in Intra-pictures (section 7.1) and not as described in this section.

$ac\sim(i,j) = (16 * ac(i,j)) // w_N(i,j)$

where:

$w_N(i,j)$ is the non-intra quantizer matrix given in figure 7.2

```
QAC(i,j) = ac~(i,j)      / (2*mquant)      IF mquant == odd
         = (ac~(i,j)+1)  / (2*mquant)      IF mquant == even AND ac~(i,j)>0
         = (ac~(i,j)-1)  / (2*mquant)      IF mquant == even AND ac~(i,j)<0
```

QAC (i,j) is limited to the range [-256..255].

7.3 Inverse Quantization

7.3.1 Intra-coded macroblocks

This section applies to all macroblocks in Intra-Frames and Intra macroblocks in Predicted and Interpolated Frames. Reconstruction levels, rec(i,j), are derived from the following formulae.

```
rec(i,j) = mquant * 2 * QAC(i,j) * $w_I(i,j)$ / 16

if (rec (i,j) is an EVEN number && rec(i,j) > 0)
        rec (i,j) = rec(i,j) - 1

if (rec (i,j) is an then number && rec(i,j) < 0)
        rec (i,j) = rec(i,j) + 1

if (QAC (i,j) == 0)
        rec (i,j) = 0
```

The DC term is special case

```
        rec (1,1) = 8 * QDC
```

Where:

mquant is the quantization parameter stored in the bit stream and calculated according to the algorithm in section 10.

rec(i,j) is limited to the range [-2048..2047].

7.3.2 Non-Intra-coded macroblocks

This section applies to all non-Intra macroblocks in Predicted and Interpolated Frames. Reconstruction levels, rec(i,j), are derived from the following formulae.

```
if (QAC(i,j) > 0)
        rec(i,j) = (2 * QAC(i,j) + 1) * mquant * $w_N(i,j)$ / 16

if (QAC(i,j) < 0)
        rec(i,j) = (2 * QAC(i,j) - 1) * mquant * $w_N(i,j)$ / 16

if (rec (i,j) is an EVEN number && rec(i,) > 0)
        rec (i,j) = rec(i,j) - 1

if (rec (i,j) is an EVEN number && rec(i,j) <0)
        rec (i,j) = rec(i,j) + 1

if (QAC (i,j) == 0)
        rec (i,j) = 0
```

rec(i,j) is limited to the range [-2048..2047].

8 CODING

This section describes the coding methods used to code the attributes and data in each macroblock. The overall syntax of the video coding is described in the following section, section 9.

The spatial position of each macroblock is encoded by a variable length code, the macroblock address (MBA). The use of macroblock addressing is described in section 8.1.

Macroblocks may take on one of a number of different modes. The modes available depend on the picture type. Section 6 describes the procedures used by the encoder to decide on which mode to use. The mode selected is identified in the bit stream by a variable length code known as MTYPE. The use of MTYPE is described in section 8.2.

The coding of motion vectors is addressed in section 8.3.

Some blocks do not contain any DCT coefficient data. To transmit which blocks of a macroblock are coded and which are non-coded, the coded block pattern (CBP) variable length code is used (see section 8.4).

The coefficients in a block are coded with VLC tables as described in section 8.5, 8.6, and 8.7.

| For additional information about scalable bit streams, refer to Annex D.

8.1 Macroblock Addressing

Relative addressing is used to code the position of all macroblocks in all frames. Macroblocks for which no data is stored are run-length encoded using the MBA; these macroblocks are called *skipped* macroblocks.

In Intra frames there are no skipped macroblocks. In predicted frames a macroblock is skipped if its motion vector is zero, all the quantized DCT coefficients are zero, and it is not the first or last macroblock in the slice. In interpolated frames, a macroblock is skipped if it has the same MTYPE as the prior macroblock, its motion vectors are the same as the corresponding motion vectors in the prior macroblock, all its quantized DCT coefficients are zero, and it is not the first or last macroblock in the slice.

A macroblock address (MBA) is a variable length code word indicating the position of a macroblock within a MB-Slice. The order of macroblocks is top-left to bottom-right in raster-scan order and is shown in Figure 4.2. For the first non-skipped macroblock in a macroblock slice, MBA is the macroblock count from the left side of the frame. For the Test Model this corresponds to the absolute address in figure 4.3. For subsequent macroblocks, MBA is the difference between the absolute addresses of the macroblock and the last non-skipped macroblock. The code table for MBA is given in Table B.1.

An extra code word is available in the table for bit stuffing immediately after a macroblock slice header or a coded macroblock (MBA Stuffing). This code word should be discarded by decoders.

The VLC for start code is also shown in Table B.1

8.2 Macroblock Type

Each frame has one of the three modes:

1	Intra	(I-frames)
2	Predicted	(P-frames)
3	Interpolated	(B-frames)

For these three frame types different VLC tables for the Macroblock types are used. See table B.2a for Intra, table B.2b for predictive-coded pictures and table B.2c for bidirectionally predictive-coded pictures.

Methods for mode decisions are described in section 6. In macroblocks that modify the quantizer control parameter the MTYPE code word is followed by a 5-bit number giving the new value of the quantization parameter, mquant, in the range [1..31].

8.2.1 Compatible Prediction Flag

A one-bit flag, **compatible_type,** immediately follows the MBTYPE VLC, when used. If its value is 1 it indicates that compatible prediction is used.

The prediction works as follows. The corresponding quarter of the reconstructed SIF macroblock is up sampled., by a 1/2 1 1/2 bilinear interpolation.

8.2.2 Field/Frame Coding Flag

A one-bit flag, **interlaced_macroblock_type**, immediately follows the MBTYPE VLC. If its value is 1 it indicates that the macroblock coefficient data is in field order as described in chapter 6. If its value is 0 it indicates that the macroblock coefficient data is in frame order.

8.2.3 Field/Frame Motion Compensation Flag

A one-bit flag, **interlaced_motion_type**, immediately follows the **interlaced_macroblock_type** flag. If its value is 1 it indicates that field-based motion prediction is used as described in section 5.2.2. If its value is 0 it indicates that frame-based motion prediction is used as described in section 5.2.1. If field-based prediction is used twice as many motion vectors are stored as are needed in the case of frame-based prediction.

8.3 Motion Vectors

Motion vectors for predicted and interpolated frames are coded differentially within a macroblock slice, obeying the following rules:

- Every forward or backward motion vector is coded relative to the last vector of the same type. Each component of the vector is coded independently, the horizontal component first and then the vertical component.

- The prediction motion vector is set to zero in the macroblocks at the start of a macroblock slice, or if the last macroblock was coded in the intra mode. (**Note**: that in predictive frames a No MC decision corresponds to a reset to zero of the prediction motion vector.)

- In interpolative frames, only vectors that are used for the selected prediction mode (MB type) are coded. Only vectors that have been coded are used as prediction motion vectors.

The VLC used to encode the differential motion vector data depends upon the range of the vectors. The maximum range that can be represented is determined by the **forward_f_code** and **backward_f_code** encoded in the picture header. (**Note**: in this Test Model the **full_pel_flag** is never set - all vectors have half-pel accuracy).

The differential motion vector component is calculated. Its range is compared with the values given in table 8.1 and is reduced to fall in the correct range by the following algorithm:

```
if (diff_vector < -range)
        diff_vector = diff_vector + 2*range;
else if (diff_vector > range-1)
        diff_vector = diff_vector - 2*range;
```

forward_f_code or backward_f_code	Range
1	16
2	32
3	64
4	128
5	256
6	512
7	1024

Table 8.1 Range for motion vectors

This value is scaled and coded in two parts by concatenating a VLC found from table B.4 and a fixed length part according to the following algorithm:

Let f_code be either the **forward_f_code** or **backward_f_code** as appropriate, and diff_vector be the differential motion vector reduced to the correct range.

```
if (diff_vector == 0) {
        residual = 0;
        vlc_code_magnitude = 0;
}
else {
        scale_factor = 1 << (f_code - 1);
        residual = (abs(diff_vector) - 1) % scale_factor;
        vlc_code_magnitude = (abs (diff_vector) - residual) / scale_factor;
        if (scale_factor != 1)
                vlc_code_magnitude += 1;
}
```

vlc_code_magnitude and the sign of diff_vector are encoded according to table B.4. The residual is encoded as a fixed length code using (f_code-1) bits.

For example to encode the following string of vector components (measured in half pel units)

3 10 30 30 -14 -16 27 24

The range is such that an f value of 2 can be used. The initial prediction is zero, so the differential values are:

3 7 20 0 -44 -2 43 -3

The differential values are reduced to the range -32 to +31 by adding or subtracting the modulus 64 corresponding to the forward_f_code of 2:

3 7 20 0 20 -2 -21 -3

These values are then scaled and coded in two parts (the table gives the pair of values to be encoded (vlc, residual)):

(2, 0) (4,0) (10, 1) (0, 0) (10, 1) (-1, 1) (-11, 0) (-2, 0)

The order in a slice is in raster scan order, except for Macroblocks coded in Field prediction mode, where the upper two luminance blocks vector is predicted from the preceding Macroblock and the two lower luminance blocks vector is predicted from the upper one, see also figure 8.1.

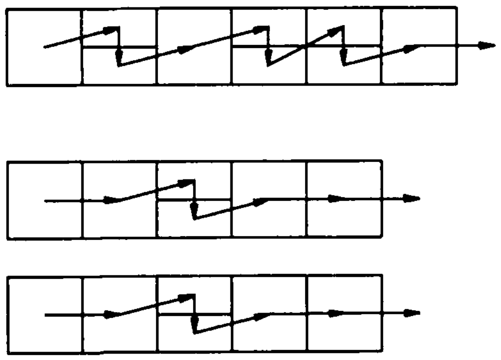

Figure 8.1 : Motion Vector Prediction Order

8.4 Coded Block Pattern

If MTYPE shows that the macroblock is not INTRA coded and all the coefficients of a block are zero after quantization, the block is declared to be not coded. If all six blocks in a macroblock are not coded, the macroblock is declared to be not coded. In all other cases the macroblock is declared to be coded.

If the MTYPE shows that the macroblock is INTRA all blocks are declared to be coded and the CBP code word is not used.

A pattern number defines which blocks within the MB are coded;

Pattern number = $32^*P_1 + 16^*P_2 + 8^*P_3 + 4^*P_4 + 2^*P_5 + P_6$

where P_n is 1 if any coefficient is present for block n, else 0. Block numbering is given in Figure 4.4.

The pattern number is coded using table B.3 Macroblock pattern

8.5 Intraframe Coefficient Coding

8.5.1 DC Prediction

After the DC coefficient of a block has been quantized to 8 bits according to section 7.1.1, it is coded lossless by a DPCM technique. Coding of the luminance blocks within a macroblock follows the normal scan of figure 4.4. Thus the DC value of block 4 becomes the DC predictor for block 1 of the following macroblock. Three independent predictors are used, one each for Y, Cr and Cb.

At the left edge of a macroblock slice, the DC predictor is set to 128 (for the first block (luminance) and the chrominance blocks). At the rest of a macroblock slice, the DC predictor is simply the previously coded DC value of the same type (Y, Cr, or Cb).

At the decoder the original quantized DC values are exactly recovered by following the inverse procedure.

The differential DC values thus generated are categorised according to their "size" as shown in the table below.

DIFFERENTIAL DC (absolute value)	SIZE
0	0
1	1
2 to 3	2
4 to 7	3
8 to 15	4
16 to 31	5
32 to 63	6
64 to 127	7
128 to 255	8

Table 8.2 Differential DC size and VLC

The size value is VLC coded according to table B.5a (luminance) and B.5b (chrominance).

For each category enough additional bits are appended to the SIZE code to uniquely identify which difference in that category actually occurred (table 8.3). The additional bits thus define the signed amplitude of the difference data. The number of additional bits (sign included) is equal to the SIZE value.

DIFFERENTIAL DC	SIZE	ADDITIONAL CODE
-255 to -128	8	00000000 to 01111111
-127 to -64	7	0000000 to 0111111
-63 to -32	6	000000 to 011111
-31 to -16	5	00000 to 01111
-15 to -8	4	0000 to 0111
-7 to -4	3	000 to 011
3 to -2	2	00 to 01
-1	1	0
0	0	
1	1	1
2 to 3	2	10 to 11
4 to 7	3	100 to 111
8 to 15	4	1000 to 1111
16 to 31	5	10000 to 11111
32 to 63	6	100000 to 111111
64 to 127	7	1000000 to 1111111
128 to 255	8	10000000 to 11111111

Table 8.3. Differential DC additional codes

8.5.2 AC Coefficients

AC coefficients are coded as described in section 8.7.

8.6 Non-Intraframe Coefficient Coding

8.6.1 Intra blocks

Intra blocks in non-intra frames are coded as in intra frames. At the start of the macroblock, the DC predictors for luminance and chrominance are reset to 128, unless the previous block was also intra; in this case, the predictors are obtained from the previous block as in intra frames (section 8.5.1).

AC coefficients are coded as described in section 8.7.Transform coefficient data is always present for all 6 blocks in a macroblock when MTYPE indicates INTRA.

8.6.2 Non intra blocks

In other cases MTYPE and CBP signal which blocks have coefficient data transmitted for them. The quantized transform coefficients are sequentially transmitted according to the zig-zag sequence given in Figure 4.5.

The most commonly occurring combinations of successive zeros (RUN) and the following value (LEVEL) are encoded with variable length codes. Other combinations of (RUN, LEVEL) are encoded with a 20-bit or 28-bit word consisting of 6 bits ESCAPE, 6 bits RUN and 8 or 16 bits LEVEL. For the variable length encoding there are two code tables, one being used for the first transmitted LEVEL in INTER and INTER + MC blocks, the second for all other LEVELs except the first one in INTRA blocks, which is encode as described in section 8.6.1.

8.6.3 Scalable blocks

Coding of intra and non intra blocks in a scalable bitstream is described in Annex D.

8.7 Coding of Transform Coefficients

The combinations of zero-run and the following value are encoded with variable length codes as listed in table B.5c to B.5f. End of Block (EOB) is in this set. Because CBP indicates those blocks with no coefficient data, EOB cannot occur as the first coefficient. Hence EOB does not appear in the VLC table for the first coefficient. Note that EOB is stored for all coded blocks.

The last bit 's' denotes the sign of the level, '0' for positive '1' for negative.

The remaining combinations of (RUN, LEVEL) are encoded with a 20-bit or 28-bit word consisting of 6 bits ESCAPE, 6 bits RUN and 8-bits or 16-bits LEVEL. RUN is a 6 bit fixed length code. LEVEL is an 8-bit or 16-bit fixed length code. See table B.5g

9 VIDEO MULTIPLEX CODER

In this section the video multiplex is explained. Unless specified otherwise the most significant bit occurs first. This is Bit 1 and is the left most bit in the code tables in this document.

9.1 Method of Describing Bit Stream Syntax

Each data item in the bit stream 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 bit stream depends on the value of that data element and on data elements previously decoded. The following constructs are used to express the conditions when data elements are present, and are in normal type:

while (condition) {	If the condition is true, then the group of data elements occurs next
data_element	in the data stream. This repeats until the condition is not true.
}	
do {	
data_element	The data element always occurs at least once.
} while (condition)	The data element is repeated until the condition is not true.
if (condition) {	If the condition is true, then the first group of data elements occurs
data_element	next in the data stream.
}	
else {	If the condition is not true, then the second group of data elements
data_element	occurs next in the data stream.
}	
for (i = 0; i < n; i++) {	The group of data elements occurs n times. Conditional constructs
data_element	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 to one for the second occurrence, and so forth.

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]	is the inclusive range of bits between bit m and bit n in the data_element.

While the syntax is expressed in procedural terms, it should not be assumed that this section implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bit stream. Actual decoders must include a 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.

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 bit stream is the first bit in a byte.

Definition of nextbits function

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

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() {
  while ( !bytealigned() )
    zero_bit                                  1        "0"
  while ( nextbits() != '0000 0000 0000 0000 0000
0001' )
    zero_byte                                 8        "00000000"
  }
```

9.2 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bit-stream.

bslbf	Bit string, left bit first, where "left" is the order in which bit strings are written in the standard. Bit strings are 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.
uimsbf	Unsigned integer, most significant bit first.
vlclbf	Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written in Annex B.

The byte order of multi-byte words is most significant byte first.

9.3 Specification of the Coded Video Bit stream Syntax

9.3.1 Start Codes

Start codes are reserved bit patterns that do not otherwise occur in the video stream. All start codes are byte aligned.

name	hexadecimal value
picture_start_code	00000100
slice_start_codes (including slice_vertical_positior	00000101 through 000001AF
slave_slice_start_code	000001B0
reserved	000001B1
user_data_start_code	000001B2
sequence_header_code	000001B3
sequence_error_code	000001B4
extension_start_code	000001B5
reserved	000001B6
sequence_end_code	000001B7
group_start_code	000001B8
system start codes (see note)	000001B9 through 000001FF
NOTE - system start codes are defined in Part 1 CD 11172	

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 the digital storage media interface to indicate where uncorrectable errors have been detected.

9.3.2 Video Sequence Layer

```
video_sequence() {
 next_start_code()
 do {
    sequence_header()
    do {
       group_of_pictures()
    }  while ( nextbits() == group_start_code )
 } while ( nextbits() == sequence_header_code )
 sequence_end_code                                    32          bslbf
}
```

9.3.3 Sequence Header

```
sequence_header() {
 sequence_header_code                                   32       bslbf
  horizontal_size                                       12       uimsbf
  vertical_size                                         12       uimsbf
  pel_aspect_ratio                                      4        uimsbf
  picture_rate                                          4        uimsbf
  bit_rate                                              18       uimsbf
  marker_bit                                            1        "1"
  vbv_buffer_size                                       10       uimsbf
  constrained_parameter_flag                            1
  load_intra_quantizer_matrix                           1
  if ( load_intra_quantizer_matrix )
     intra_quantizer_matrix[64]                         8*64     uimsbf
  load_non_intra_quantizer_matrix                       1
  if ( load_non_intra_quantizer_matrix )
     non_intra_quantizer_matrix[64]                     8*64     uimsbf
   next_start_code()
   if (nextbits() == extension_start_code ) {
     extension_start_code                               32       bslbf
     compatible                                         3        uimsbf
     interlaced                                         1        uimsbf
    scalable                                            1        uimsbf
     reserved                                           3        uimsbf
     if (scalable) {
       do {
        scale_code                                      8        uimsbf
       } while (nextbits != '00000111')
        end_of_scales_code                              8        '00000111'
     }
     while ( nextbits () != '0000 0000 0000 0000 0000 0001' ) {
        sequence_extension_data                         8
     }
     next_start_code()
  }
  if (nextbits() == user_data_start_code ) {
     user_data_start_code                               32       bslbf
     while ( nextbits() != '0000 0000 0000 0000 0000 0001' ) {
        user_data                                       8
     }
     next_start_code()
  }
}
```

compatible -- This is a three-bit integer defined in the following table.

binary value	Standard
000	not compatible
001	MPEG1 compatible
010	H.261 compatible
011	MPEG2 compatible
100	reserved
...	reserved
111	reserved

For scalable bitstreams this integer is set to zero.

interlaced - This is a one-bit integer defined in the following table.

binary value	Coding
0	not interlaced
1	interlaced

For scalable bitstreams this integer is set to zero.

scalable - This is a one-bit integer defined in the following table.

0	not scalable
1	scalable

scale_code - This is an 8-bit integer that defines the DCT size for the scalable layers, i.e. DCT_size = 1<<scale_code. The values of this integer are:

scale_code	
0	scale 1 layer
1	scale 2 layer
2	scale 4 layer
3	scale 8 layer
4	reserved
5	reserved
6	reserved
7	**end_of_scales_code**
8	reserved
256	reserved

Example: The bitstream with scale_codes 1, 2, 3, 3, 7 would be interpreted to mean that there are four layers, a scale-2, followed by a scale-4, followed by two scale-8 layers.

9.3.4 Group of Pictures Layer

```
group_of_pictures() {

    group_start_code                                   32          bslbf
    time_code                                          25
    closed_gop                                         1
    broken_link                                        1
    next_start_code()
    if ( nextbits() == extension_start_code ) {
      extension_start_code                             32          bslbf
      while ( nextbits() != '0000 0000 0000 0000 0000 0001
        group_extension_data                           8
        }
      next_start_code()
      }
    if ( nextbits() == user_data_start_code ) {
      user_data_start_code                             32          bslbf
      while ( nextbits() != '0000 0000 0000 0000 0000 0001
        user_data                                      8
        }
      next_start_code()
      }
    do {
      picture()
      } while ( nextbits() == picture_start_code )
    }
```

9.3.5 Picture Layer

```
picture() {

  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 == :
       full_pel_forward_vector                         1
       forward_f_code                                  3         uimsbf
    }
    if ( picture_coding_type == 3 ) {
       full_pel_backward_vector                        1
       backward_f_code                                 3         uimsbf
       }
    while ( nextbits() == '1' ) {
       extra_bit_picture                               1         "1"
       extra_information_picture                       8
    }
     extra_bit_picture                                 1         "0"
    next_start_code()

    if (nextbits() == extension_start_code ) {
       extension_start_code                            32        bslbf
       while ( nextbits() != '0000 0000 0000 0000 0000 0001
          picture_extension_data                       8
          }
       next_start_code()
       }
    if ( nextbits() == user_data_start_code ) {
       user_data_start_code                            32        bslbf
       while ( nextbits() != '0000 0000 0000 0000 0000 0001
          user_data                                    8
          }
       next_start_code()
       }
    do {
       slice()
    } while ( nextbits() == slice_start_code )

  }
```

9.3.6 Slice Layer

```
slice() {
      slice_start_code                              32          bslbf
      quantizer_scale                               5           uimsbf
       if (scalable) {
        extra_bit_slice                             1           "1"
        dct_size                                    8           uimsbf
      }
      while ( nextbits() == '1' ) {
          extra_bit_slice                           1           "1"
          extra_information_slice                   8
      }
      extra_bit_slice                               1           "0"
     do {
      macroblock()
      } while ( nextbits() != '000 0000 0000 0000 0000 0000' )
      next_start_code()
  }
```

9.3.6.1 Slave Slice Layer

```
slave_slice() {
      slave_slice_start_code                        32          bslbf
      quantizer_delta_magnitude                     5           uimsbf
    quantizer_delta_sign                            1           uimsbf
    dct_size                                        8           uimsbf
      for (s=0; s<slice_size; s++) {
       slave_macroblock(dct_size)
}
```

For scalable bitstreams, the following definitions apply: dct_size - 1, 2, 4, or 8.
quantizer_delta: This integer is added to all "quantizer_scale" values in the slice and macroblock layers, to derive the corresponding quantizer_scale values (mquant) in the slave_slice and slave_macroblock layers. For the Test Model this delta is zero.
quantizer_delta_magnitude: specifies the magnitude of "quantizer_delta".
quantizer_delta_sign: specifies the sign of "quantizer_delta".
slice_size: the total number of macroblocks in the slice layer (44).

9.3.7 Macroblock Layer

```
macroblock() {
    while ( nextbits() == '0000 0001 111' )
        macroblock_stuffing                                 11      vlclbf
    while ( nextbits() == '0000 0001 000' )
        macroblock_escape                                   11      vlclbf
    macroblock_address_increment                            1-11    vlclbf
    macroblock_type                                         1-6     vlclbf
    if (compatible)
        compatible_type                                     1       uimsbf
    if (interlaced) {
        if (macroblock_intra | | macroblock_pattern )
            interlaced_macroblock_type                      1       uimsbf
        if ((macroblock_motion_forward) | |
                        (macroblock_motion_backward))
            interlace_motion_type                           1       uimsbf
        }
    if ( macroblock_quant )
        quantizer_scale                                     5       uimsbf
    if ( macroblock_motion_forward ) {
        motion_horizontal_forward_code                      1-11    vlclbf
        if ( (forward_f != 1) &&
                    (motion_horizontal_forward_code != 0) )
            motion_horizontal_forward_r                     1-6     uimsbf
        motion_vertical_forward_code                        1-11    vlclbf
        if ( (forward_f != 1) &&
                    (motion_vertical_forward_code != 0) )
            motion_vertical_forward_r                       1-6     uimsbf
        }
        if (interlace_motion_type) {
            motion_horizontal_forward_code_2                1-11    vlclbf
            if ((forward_f != 1) &&
                    (motion_horizontal_forward_code_2 != 0))
                motion_horizontal_forward_r_2               1-6     uimsbf
            motion_vertical_forward_code_2                  1-11    vlclbf
            if ((forward_f != 1) &&
                    (motion_vertical_forward_code_2 != 0))
                motion_vertical_forward_r_2                 1-6     uimsbf
            }
        }
    if ( macroblock_motion_backward ) {
        motion_horizontal_backward_code                     1-11    vlclbf
        if ((backward_f != 1) &&
                    (motion_horizontal_backward_code != 0))
            motion_horizontal_backward_r                    1-6     uimsbf
        motion_vertical_backward_code                       1-11    vlclbf
        if ((backward_f != 1) &&
                    (motion_vertical_backward_code != 0))
            motion_vertical_backward_r                      1-6     uimsbf
        if (interlace_motion_type) {
            motion_horizontal_backward_code_2            1-11      vlclbf
              if ((backward_f != 1) &&
                   (motion_horizontal_backward_code_2 != 0))
                motion_horizontal_backward_r_2           1-6       uimsbf
            motion_vertical_backward_code_2              1-11      vlclbf
              if ((backward_f != 1) &&
                   (motion_vertical_backward_code_2 != 0))
                motion_vertical_backward_r_2             1-6       uimsbf
              }
          }
        if ( macroblock_pattern)
          coded_block_pattern                            3-9       vlclbf
        for ( i=0; i<6; i++ )
          if (scalable) {
            scaled_block(i)
          }
          else {
            block( i )
          }
        }
        if ( picture_coding_type == 4 )
          end_of_macroblock                              1         "1"
    }
```

compatible_type - This is a one-bit integer defined in the following table.

binary value	Prediction
0	Not Compatible
1	Compatible

interlace_macroblock_type - This is a one-bit integer indicating whether the macoblock is frame DCT coded or field DCT coded. If this is set to "1", the macroblock is field DCT coded.

interlace_motion_type - This is a one-bit integer indicting whether the macroblock motin prediciton is frame based or field based. If this is set to "1", the macroblock is field based motion compensated, and two motion vectors per prediction direction are sent.

9.3.7.1 Slave Macroblock Layer

```
slave_macroblock(dct_size) {
  for (i=0; i<6; i++) {
    slave_block[i, dct_size]
  }
}
```

9.3.8 Block Layer

```
block( i ) {

    if ( pattern_code[i] ) {
      if ( macroblock_intra ) {
        if ( i<4 ) {
          dct_dc_size_luminance                       2-7      vlclbf
           if(dct_dc_size_luminance != 0)
             dct_dc_differential                      1-8      uimsbf
           }
        else {
          dct_dc_size_chrominance                     2-8      vlclbf
           if(dct_dc_size_chrominance !=0)
             dct_dc_differential                      1-8      uimsbf
           }
        }
      else {
        dct_coeff_first                               2-28     vlclbf
        }
      if ( picture_coding_type != 4 ) {
        while ( nextbits() != '10')
          dct_coeff_next                              3-28     vlclbf
        end_of_block                                  2        "10"
        }
      }
  }
```

9.3.8.1 Scaled Block Layer

For scalable bitstreams the following syntax extensions apply:

```
scaled_block(i) {
  if (pattern_code[i]) {
    if (macroblock_intra) {
      if (i<4) {
        dct_dc_size_luminance                   2-7    vlclbf
         if (dct_dc_size_luminance != 0)
          dct_dc_differential                   1-8    vlclbf
        }
      else {
        dct_dc_size_chrominance                 2-8    vlclbf
         if (dct_dc_size_chrominance != 0)
          dct_dc_differential                   1-8    vlclbf
        }
    }
    if (dct_size > 1) {
        while ((nextbits() != eob_code) && more_coefs)
          next_dct_coef(dct_size)              2-16   vlclbf
         if (more_coefs)
          end_of_block                         2-16   vlclbf
        }
    }
  }
```

```
}

slave_block [i,dct_size] {
  if (pattern_code[i]) {
    while ((nextbits() != eob_code) && more_coefs) {
      next_dct_coef(dct_size)                 2-16   vlclbf
    }
    if (more_coefs) {
      end_of_block                            2-16   vlclbf
    }
  }
}
```

pattern_code(i) - For slave_blocks, this code is the same as that of the colocated scaled_block in the slice layer.

more_coefs - more_coefs is true if we have not already decoded the last coefficient in the block of DCT coefficients except that, for the 8x8 slave_slice, more_coefs is always true (this is to retain compatibility with MPEG-1 style of coding 8x8 blocks, which always includes an end_of_block code).

eob_code - An end_of_block Huffman code specified in the appropriate resolution scale VLC table.

next_dct_coef - DCT coefficient coded by run/amplitude or run/size VLCs. The VLC table used depends on "dct_size", as explained in Annex D.

10 RATE CONTROL AND QUANTIZATION CONTROL

This section describes the procedure for controlling the bit-rate of the Test Model by adapting the macroblock quantization parameter. The algorithm works in three-steps:

1 **Target bit allocation**: this step estimates the number of bits available to code the next frame. It is performed before coding the frame.

2 **Rate control**: by means of a "virtual buffer", this step sets the reference value of the quantization parameter for each macroblock.

3 **Adaptive quantization**: this step modulates the reference value of the quantization parameter according to the spatial activity in the macroblock to derive the value of the quantization parameter, mquant, that is used to quantize the macroblock.

Step 1 - Bit Allocation

Complexity estimation

After a frame of a certain type (I, P, or B) is encoded, the respective "global complexity measure" (X_i, X_p, or X_b) is updated as:

$$X_i = S_i\ Q_i\ , \quad X_p = S_p\, Q_p, \quad X_b = S_b\, Q_b$$

where S_i, S_p, S_b are the number of bits generated by encoding this frame and Q_i, Q_p and Q_b are the average quantization parameter computed by averaging the actual quantization values used during the encoding of the all the macroblocks, including the skipped macroblocks.

Initial values

$X_i = 160 * \textbf{bit\_rate} / 115$
$X_p = 60 * \textbf{bit\_rate} / 115$
$X_b = 42 * \textbf{bit\_rate} / 115$

bit_rate is measured in bits/s.

Picture Target Setting

The target number of bits for the next frame in the Group of pictures (T_i, T_p, or T_b) is computed as:

$$T_i = \max \left\{ \frac{R}{1 + \frac{N_p X_p}{X_i K_p} + \frac{N_b X_b}{X_i K_b}} , \textbf{bit\_rate} / (8^{*}\textbf{picture\_rate}) \right\}$$

$$T_p = \max \left\{ \frac{R}{N_p + \frac{N_b K_p X_b}{K_b X_p}} , \textbf{bit\_rate} / (8^{*}\textbf{picture\_rate}) \right\}$$

$$T_b = \max \left\{ \frac{R}{N_b + \frac{N_p K_b X_p}{K_p X_b}} , \textbf{bit\_rate} / (8^{*}\textbf{picture\_rate}) \right\}$$

Where:

K_p and K_b are "universal" constants dependent on the quantization matrices. For the matrices specified in sections 7.1 and 7.2 $K_p = 1.0$ and $K_b = 1.4$.

R is the remaining number of bits assigned to the GROUP OF PICTURES. R is updated as follows:

After encoding a frame , $R = R - S_{i,p,b}$

Where is $S_{i,p,b}$ is the number of bits generated in the picture just encoded (picture type is I, P or B).

Before encoding the first frame in a GROUP OF PICTURES (an I-frame):

$R = G + R$
$G = \textbf{bit\_rate} * N / \textbf{picture\_rate}$
N is the number of frames in the GROUP OF PICTURES.

At the start of the sequence $R = 0$.

N_p and N_b are the number of P-frames and B-frames remaining in the current GROUP OF PICTURES in the encoding order.

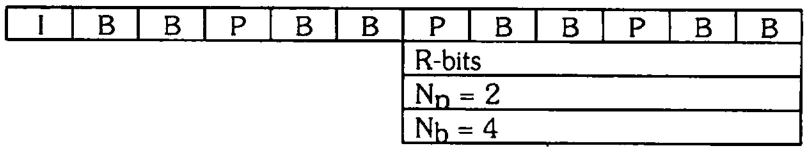

Figure 10.1 - GROUP OF PICTURES structure

Step 2 - Rate Control

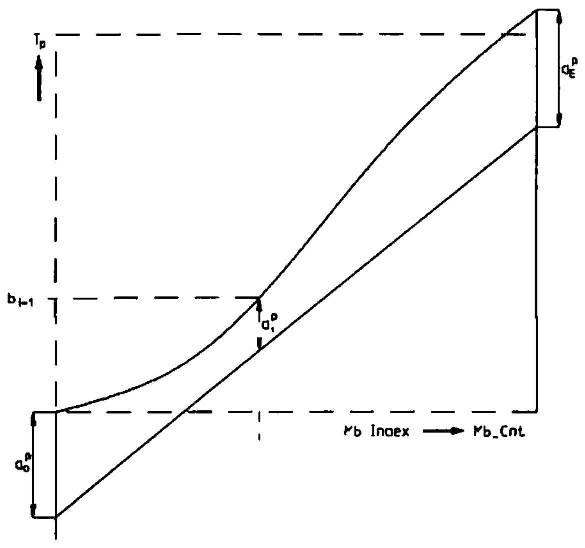

Figure 10.2 : Rate Control for P-frames

Before encoding macroblock j (j >= 1), compute the fullness of the appropriate virtual buffer:

$$d_j^i = d_0^i + B_{j-1} - \frac{T_i\,(j-1)}{MB\_cnt}$$

or

$$d_j^p = d_0^p + B_{j-1} - \frac{T_p\,(j-1)}{MB\_cnt}$$

or

$$d_j^b = d_0^b + B_{j-1} - \frac{T_b\,(j-1)}{MB\_cnt}$$

depending on the picture type.

where

d_0^i, d_0^p, d_0^b are initial fullnesses of virtual buffers - one for each frame type.

B_j is the number of bits generated by encoding all macroblocks in the frame up to and including j.

MB_cnt is the number of macroblocks in the frame.

d_j^i, d_j^p, d_j^b are the fullnesses of virtual buffers at macroblock j- one for each frame type.

The final fullness of the virtual buffer (d_j^i , d_j^p, d_j^b: j =MB_cnt) is used as d_0^i, d_0^p, d_0^b for encoding the next frame of the same type.

Next compute the reference quantization parameter Q_j for macroblock j as follows:

$d_j * 31$

$$Q_j = \frac{\quad}{r}$$

where the "reaction parameter" r is given by

$$r = 2 * \textbf{bit\_rate} / \textbf{picture\_rate}$$

and d_j is the fullness of the appropriate virtual buffer.

The initial value for the virtual buffer fullness is:

$$d_0^i = 10 * r/31$$
$$d_0^p = K_p\, d_0^i$$
$$d_0^b = K_b\, d_0^i$$

Step 3 - Adaptive Quantization

Compute a spatial activity measure for the macroblock j from the four luminance sub-blocks using the intra (ie original) pixels values:

$$act_j = 1 + \min_{sblk=1,4} (var\_sblk)$$

where

$$var\_sblk = \sum_{k=1}^{64} (P_k - P\_mean)^2$$

$$P\_mean = \frac{1}{64} \sum_{k=1}^{64} P_k$$

and P_k are the pixel values in the original 8*8 block.

Normalize act_j:

$$N\_act_j = \frac{2 * act_j + avg\_act}{act_j + 2 * avg\_act}$$

avg_act is the average value of act_j the last frame to be encoded. On the first frame, avg_act = 400.

Obtain $mquant_j$ as:

$$mquant_j = Q_j * N\_act_j$$

where Q_j is the reference quantization parameter obtained in step 2. The final value of $mquant_j$ is clipped to the range [1..31] and is used and coded as described in sections 7, 8 and 9 in either the slice or macroblock layer.

Known Limitations

- Step 1 does not handle scene changes efficiently.

- Step 3 does not work well on highly interlaced material, since the entire method uses frame macroblocks.

- A wrong value of avg_act is used in step 3 after a scene change.

- VBV compliance is not guaranteed.

APPENDIX A: Discrete Cosine Transform (DCT)

The 2-dimensional DCT is defined as:

$$F(u,v) = (1/4)\, C(u)\, C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left\{\frac{(2x+1)u*pi}{16}\right\} \cos\left\{\frac{(2y+1)v*pi}{16}\right\},$$

with u, v, x, y = 0, 1, 2, ... 7

where x, y = spatial coordinates in the pel domain
u, v = coordinates in the transform domain

C(u), C(v) = 1/SQRT (2) for u, v=0
1 otherwise

The inverse DCT (IDCT) is defined as:

$$f(x,y) = (1/4) \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)\, C(v)\, F(u,v) \cos\left\{\frac{(2x+1)u*pi}{16}\right\} \cos\left\{\frac{(2y+1)v*pi}{16}\right\},$$

The input to the forward transform and output from the inverse transform is represented with 9 bits. The coefficients are represented in 12 bits. The dynamic range of the DCT coefficients is (-2048, ..., 2047).

Accuracy Specification

The 8 by 8 inverse discrete transform shall conform to IEEE Draft Standard Specification for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, P1180/D2, July 18, 1990. Note that Section 2.3 P1180/D2 "Considerations of Specifying IDCT Mismatch Errors" requires the specification of periodic intra-picture coding in order to control the accumulation of mismatch errors. The maximum refresh period requirement for this standard shall be 132 pictures, the same as indicated in P1180/D2 for visual telephony according to CCITT Recommendation H.261 (see Bibliography).

APPENDIX B: VARIABLE LENGTH CODE TABLES

Introduction

This annex contains the variable length code tables for macroblock addressing, macroblock type, macroblock pattern, motion vectors, and DCT coefficients.

B.1 Macroblock Addressing

Table B.1. Variable length codes for macroblock_address_increment.

macroblock_address_ increment VLC code	increment value	macroblock_address_ increment VLC code	increment value
1	1	0000 0101 10	17
011	2	0000 0101 01	18
010	3	0000 0101 00	19
0011	4	0000 0100 11	20
0010	5	0000 0100 10	21
0001 1	6	0000 0100 011	22
0001 0	7	0000 0100 010	23
0000 111	8	0000 0100 001	24
0000 110	9	0000 0100 000	25
0000 1011	10	0000 0011 111	26
0000 1010	11	0000 0011 110	27
0000 1001	12	0000 0011 101	28
0000 1000	13	0000 0011 100	29
0000 0111	14	0000 0011 011	30
0000 0110	15	0000 0011 010	31
0000 0101 11	16	0000 0011 001	32
		0000 0011 000	33
		0000 0001 111	macroblock_stuffing
		0000 0001 000	macroblock_escape

B.2 Macroblock Type

Table B.2a. Variable length codes for macroblock_type in intra-coded pictures (I-pictures).

VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock motion_ backward	macroblock_ pattern	macroblock_ intra
1	0	0	0	0	1
01	1	0	0	0	1

Table B.2b. Variable length codes for macroblock_type in predictive-coded pictures (P-pictures).

VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock motion_ backward	macroblock_ pattern	macroblock_ intra
1	0	1	0	1	0
01	0	0	0	1	0
001	0	1	0	0	0
00011	0	0	0	0	1
00010	1	1	0	1	0
00001	1	0	0	1	0
000001	1	0	0	0	1

Table B.2c. Variable length codes for macroblock_type in bidirectionally predictive-coded pictures (B-pictures).

VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra
10	0	1	1	0	0
11	0	1	1	1	0
010	0	0	1	0	0
011	0	0	1	1	0
0010	0	1	0	0	0
0011	0	1	0	1	0
00011	0	0	0	0	1
00010	1	1	1	1	0
000011	1	1	0	1	0
000010	1	0	1	1	0
000001	1	0	0	0	1

Table B.2d. Variable length codes for macroblock_type in DC intra-coded pictures (D-pictures).

VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra
1	0	0	0	0	1

B.3 Macroblock Pattern

Table B.3. Variable length codes for coded_block_pattern.

coded_block_pattern VLC code	cbp	coded_block_pattern VLC code	cbp
111	60	0001 1100	35
1101	4	0001 1011	13
1100	8	0001 1010	49
1011	16	0001 1001	21
1010	32	0001 1000	41
1001 1	12	0001 0111	14
1001 0	48	0001 0110	50
1000 1	20	0001 0101	22
1000 0	40	0001 0100	42
0111 1	28	0001 0011	15
0111 0	44	0001 0010	51
0110 1	52	0001 0001	23
0110 0	56	0001 0000	43
0101 1	1	0000 1111	25
0101 0	61	0000 1110	37
0100 1	2	0000 1101	26
0100 0	62	0000 1100	38
0011 11	24	0000 1011	29
0011 10	36	0000 1010	45
0011 01	3	0000 1001	53
0011 00	63	0000 1000	57
0010 111	5	0000 0111	30
0010 110	9	0000 0110	46
0010 101	17	0000 0101	54
0010 100	33	0000 0100	58
0010 011	6	0000 0011 1	31
0010 010	10	0000 0011 0	47
0010 001	18	0000 0010 1	55
0010 000	34	0000 0010 0	59
0001 1111	7	0000 0001 1	27
0001 1110	11	0000 0001 0	39
0001 1101	19		

B.4 Motion Vectors

Table B.4. Variable length codes for motion_horizontal_forward_code, motion_vertical_forward_code, motion_horizontal_backward_code, and motion_vertical_backward_code.

motion VLC code	code
0000 0011 001	-16
0000 0011 011	-15
0000 0011 101	-14
0000 0011 111	-13
0000 0100 001	-12
0000 0100 011	-11
0000 0100 11	-10
0000 0101 01	-9
0000 0101 11	-8
0000 0111	-7
0000 1001	-6
0000 1011	-5
0000 111	-4
0001 1	-3
0011	-2
011	-1
1	0
010	1
0010	2
0001 0	3
0000 110	4
0000 1010	5
0000 1000	6
0000 0110	7
0000 0101 10	8
0000 0101 00	9
0000 0100 10	10
0000 0100 010	11
0000 0100 000	12
0000 0011 110	13
0000 0011 100	14
0000 0011 010	15
0000 0011 000	16

B.5 DCT Coefficients

Table B.5a Variable length codes for dct_dc_size_luminance.

VLC code	dct_dc_size_luminance
100	0
00	1
01	2
101	3
110	4
1110	5
11110	6
111110	7
1111110	8

Table B.5b. Variable length codes for dct_dc_size_chrominance.

VLC code	dct_dc_size_chrominance
00	0
01	1
10	2
110	3
1110	4
11110	5
111110	6
1111110	7
11111110	8

Table B.5c. Variable length codes for dct_coeff_first and dct_coeff_next.

dct_coeff_first and dct_coeff_ne variable length code (NOTE1	run	level
10	end_of_block	
1 s (NOTE2)	0	1
11 s (NOTE3)	0	1
011 s	1	1
0100 s	0	2
0101 s	2	1
0010 1 s	0	3
0011 1 s	3	1
0011 0 s	4	1
0001 10 s	1	2
0001 11 s	5	1
0001 01 s	6	1
0001 00 s	7	1
0000 110 s	0	4
0000 100 s	2	2
0000 111 s	8	1
0000 101 s	9	1
0000 01	escape	
0010 0110 s	0	5
0010 0001 s	0	6
0010 0101 s	1	3
0010 0100 s	3	2
0010 0111 s	10	1
0010 0011 s	11	1
0010 0010 s	12	1
0010 0000 s	13	1
0000 0010 10 s	0	7
0000 0011 00 s	1	4
0000 0010 11 s	2	3
0000 0011 11 s	4	2
0000 0010 01 s	5	2
0000 0011 10 s	14	1
0000 0011 01 s	15	1
0000 0010 00 s	16	1

NOTE1 - The last bit 's' denotes the sign of the level, '0' for posi
'1' for negative.
NOTE2 - This code shall be used for dct_coeff_first.
NOTE3 - This code shall be used for dct_coeff_next.

Table B.5d. Variable length codes for dct_coeff_first and dct_coeff_next (continued).

dct_coeff_first and dct_coeff_ne variable length code (NOTE	run	level
0000 0001 1101 s	0	8
0000 0001 1000 s	0	9
0000 0001 0011 s	0	10
0000 0001 0000 s	0	11
0000 0001 1011 s	1	5
0000 0001 0100 s	2	4
0000 0001 1100 s	3	3
0000 0001 0010 s	4	3
0000 0001 1110 s	6	2
0000 0001 0101 s	7	2
0000 0001 0001 s	8	2
0000 0001 1111 s	17	1
0000 0001 1010 s	18	1
0000 0001 1001 s	19	1
0000 0001 0111 s	20	1
0000 0001 0110 s	21	1
0000 0000 1101 0 s	0	12
0000 0000 1100 1 s	0	13
0000 0000 1100 0 s	0	14
0000 0000 1011 1 s	0	15
0000 0000 1011 0 s	1	6
0000 0000 1010 1 s	1	7
0000 0000 1010 0 s	2	5
0000 0000 1001 1 s	3	4
0000 0000 1001 0 s	5	3
0000 0000 1000 1 s	9	2
0000 0000 1000 0 s	10	2
0000 0000 1111 1 s	22	1
0000 0000 1111 0 s	23	1
0000 0000 1110 1 s	24	1
0000 0000 1110 0 s	25	1
0000 0000 1101 1 s	26	1

NOTE - The last bit 's' denotes the sign of the level, '0' for positi '1' for negative.

Table B.5e. Variable length codes for dct_coeff_first and dct_coeff_next (continued).

dct_coeff_first and dct_coeff_ne variable length code (NOTE	run	level
0000 0000 0111 11 s	0	16
0000 0000 0111 10 s	0	17
0000 0000 0111 01 s	0	18
0000 0000 0111 00 s	0	19
0000 0000 0110 11 s	0	20
0000 0000 0110 10 s	0	21
0000 0000 0110 01 s	0	22
0000 0000 0110 00 s	0	23
0000 0000 0101 11 s	0	24
0000 0000 0101 10 s	0	25
0000 0000 0101 01 s	0	26
0000 0000 0101 00 s	0	27
0000 0000 0100 11 s	0	28
0000 0000 0100 10 s	0	29
0000 0000 0100 01 s	0	30
0000 0000 0100 00 s	0	31
0000 0000 0011 000 s	0	32
0000 0000 0010 111 s	0	33
0000 0000 0010 110 s	0	34
0000 0000 0010 101 s	0	35
0000 0000 0010 100 s	0	36
0000 0000 0010 011 s	0	37
0000 0000 0010 010 s	0	38
0000 0000 0010 001 s	0	39
0000 0000 0010 000 s	0	40
0000 0000 0011 111 s	1	8
0000 0000 0011 110 s	1	9
0000 0000 0011 101 s	1	10
0000 0000 0011 100 s	1	11
0000 0000 0011 011 s	1	12
0000 0000 0011 010 s	1	13
0000 0000 0011 001 s	1	14

NOTE - The last bit 's' denotes the sign of the level, '0' for positi '1' for negative.

Table B.5f. Variable length codes for dct_coeff_first and dct_coeff_next (continued).

dct_coeff_first and dct_coeff_ne variable length code (NOTE	run	level
0000 0000 0001 0011 s	1	15
0000 0000 0001 0010 s	1	16
0000 0000 0001 0001 s	1	17
0000 0000 0001 0000 s	1	18
0000 0000 0001 0100 s	6	3
0000 0000 0001 1010 s	11	2
0000 0000 0001 1001 s	12	2
0000 0000 0001 1000 s	13	2
0000 0000 0001 0111 s	14	2
0000 0000 0001 0110 s	15	2
0000 0000 0001 0101 s	16	2
0000 0000 0001 1111 s	27	1
0000 0000 0001 1110 s	28	1
0000 0000 0001 1101 s	29	1
0000 0000 0001 1100 s	30	1
0000 0000 0001 1011 s	31	1

NOTE - The last bit 's' denotes the sign of the level, '0' for positi
'1' for negative.

Table B.5g. Encoding of run and level following escape code as a 20-bit fixed length code (-127 <= level <= 127) or as a 28-bit fixed length code (-255 <= level <= -128, 128 <= level <= 255).

fixed length code	run
0000 00	0
0000 01	1
0000 10	2
...	...
...	...
...	...
...	...
...	...
...	...
1111 11	63

fixed length code	level
forbidden	-256
1000 0000 0000 000	-255
1000 0000 0000 001	-254
...	
1000 0000 0111 111	-129
1000 0000 1000 000	-128
1000 0001	-127
1000 0010	-126
...	...
1111 1110	-2
1111 1111	-1
forbidden	0
0000 0001	1
...	...
0111 1111	127
0000 0000 1000 000	128
0000 0000 1000 000	129
...	
0000 0000 1111 111	255

APPENDIX C : Video Buffering Verifier

Constant rate coded video bit streams shall meet constraints imposed through a Video Buffering Verifier (VBV) defined in Section C.1.

The VBV is a hypothetical decoder which is conceptually connected to the output of an encoder. Coded data is placed in the buffer at the constant bit rate that is being used. Coded data is removed from the buffer as defined in Section C.1.4, below. It is a requirement of the encoder (or editor) that the bit stream it produces will not cause the VBV to either overflow or underflow.

C.1 Video Buffering Verifier

1. The VBV and the video encoder have the same clock frequency as well as the same picture rate, and are operated synchronously.

2. The VBV has a receiving buffer of size B, where B is given in the vbv_buffer_size field in the sequence header.

3. The VBV is initially empty. It is filled from the bitstream for the time specified by the vbv_delay field in the video bitstream.

4. All of the data for the picture which has been in the buffer longest is instantaneously removed. Then after each subsequent picture interval all of the data for the picture which (at that time) has been in the buffer longest is instantaneously removed. Sequence header and group of picture layer data elements which immediately precede a picture are removed at the same time as that picture. The VBV is examined immediately before removing any data (sequence header data, group of picture layer or picture) and immediately after each picture is removed. Each time the VBV is examined its occupancy shall lie between zero bits and B bits where B is the size of the VBV buffer indicated by vbv_buffer_size in the sequence header.

This is a requirement on the video bit stream including coded picture data, user data and all stuffing.

To meet these requirements the number of bits for the (n+1)'th coded picture d_{n+1} (including any preceding sequence header and group of picture layer data elements) must satisfy:

$$d_{n+1} > B_n + 2R/P - B$$

$$d_{n+1} <= B_n + R/P$$

where:

$n >= 0$
B_n is the buffer occupancy just after time t_n
R = bitrate
P = number of pictures per second
t_n is the time when the n'th coded picture is removed from the VBV buffer

Figure C.1 VBV Buffer Occupancy

APPENDIX D : Extension for scalability experiments

The necessary syntax extensions for scalable bitstreams have been described in Chapter 9. These extensions implement a hierarchical pyramid in the frequency (DCT) domain. Although the syntax is flexible, in the Test Model we restrict its application to two layers of spatial resolution; 1/4 and 1/16 of CCIR-601.

In the non-scalable Test Model a MB is subdivided into six 8x8 blocks of luminance and chrominance information, each block being coded using the 8x8 Discrete Cosine Transform (DCT). The scalability extensions allow coding of multiple video resolutions by implementing a hierarchy of layers corresponding to subsets of the 8x8 DCT coefficients. In decoding to a target resolution, we must use an inverse DCT of a size that matches the size of the subset of coefficients of the target. Finally, to increase coding efficiency, DCT data for high levels of resolution are coded differentially from DCT data from lower levels of resolution. This prediction scheme also allows for some degree of bandwidth control for each of the resolution layers.

D.1.1 MULTIPLEXING OF MULTIRESOLUTION DATA

Data for various resolution scales is multiplexed at the level of the Slice layer. The Slice layer, in a scaled bitstream, contains data for the lowest resolution scale; it is followed by a variable number of Slave_slices that contain data for the other resolution scales. The syntax for the scaled Slice layer is compatible with the syntax for the non-scaled Slice layer.

The Macroblock syntax for a scalable bitstream is also compatible with the corresponding syntax for a non-scalable bitstream. To preserve the MB structure, all MB attributes are coded with the lowest resolution scale, using the standard Macroblock syntax (except for the coding of "scaled_blocks" instead of "blocks"). Thus MB addresses, types, motion vectors, and coded block patterns, are coded together with the information for the lowest resolution "scaled_blocks". The low resolution "scaled_blocks" are coded, however, using a modification of the usual "block" syntax, as shown in Chapter 9.

The "Slave_slices" contain Slave_macroblock data which, in turn, contain additional DCT coefficient data. These data increase the resolution of the scaled_blocks in the MBs. Because the MB attributes of the Slice layer are inherited by the Slave_macroblocks, we need only code additional DCT data for blocks marked by the corresponding Coded Block Pattern.

Two slave_slices are used in the Test Model. The first contains data for 1/4 CCIR resolution, i.e. 4x4 DCT coefficients; the second brings the resolution up to full CCIR resolution, i.e. 8x8 DCT coefficients.

D.1.2 HIERARCHICAL PREDICTION

DCT coefficients in a low resolution layer are used to predict the corresponding coefficients in the next (higher) resolution layer. Thus, the 2x2 DCT coefficients of the Test Model's base Slice layer are used to predict the upper-left 2x2 coefficients of the corresponding coefficients in the 4x4 slave layer. Similarly, the 4x4 coefficients of the first slave layer are used to predict the upper-left 4x4 coefficients of the corresponding coefficients in the 8x8 slave layer. After computing the prediction differences, these differences together with the new (not predicted) coefficient data are coded using a zig-zag scan pattern as shown in Figure 4.8

D.1.3 DCT AND IDCT DEFINITIONS FOR LOW RESOLUTION SCALES

DCT transforms with non-standard normalization factors should be used to facilitate the process of quantization and inverse quantization. One can use the same quantization matrix in all resolution scales, if the following definitions are used:

SCALE	FDCT	IDCT
8x8	DCT(8x8)	IDCT(8x8)
4x4	2*DCT(4x4)	IDCT(4x4)/2
2x2	4*DCT(2x2)	IDCT(2x2)/4

where DCT(NxN) and IDCT(NxN) are the standard 2-dimensional definitions for the transforms of size N.

An alternative implementation, not supported in this Test Model, would use the standard DCT definitions and instead renormalize the quantization matrix.

D.1.4 QUANTIZATION

One quantization matrix is used for all resolution scales. For the 2x2 hierarchical layer, the upper left 2x2 elements of the 8x8 quantizer matrix are used. Similarly, for the 4x4 layer, the upper left 4x4 elements of the 8x8 matrix are used. In the Test Model, the quantized DCT coefficients in the 2x2 and 4x4 layers are obtained by simply extracting the appropriate coefficients from the corresponding set of quantized 8x8 coefficients.

In general, to rebuild the DCT coefficients for a target scale, the following steps are needed:

1. partial inverse quantization of the DCT coefficients of the target scale and all lower scales by the appropriate quantizer scale factor "mquant_x" (mquant_2 for scale 2x2, mquant_4 for scale 4x4, and mquant_8 for scale 8x8),

2. where appropriate, summation of the results of the previous step (hierarchical prediction),

3. final inverse quantization of the DCT coefficients of the target scale using the appropriate quantization matrix (intra or non-intra).

Note that mquant_2=mquant_4=mquant_8 in this Test Model. The partial and the final inverse quantizations are defined such that their combined formula corresponds exactly to the formula for inverse quantization in the Test Model (section 7.3). Thus, except for DC coefficients in intra blocks, partial inverse quantization is defined by:

```
if QAC(i,j) = 0 then
  partial_inv(i,j) = 0
else
  partial_inv(i,j) = (2*QAC(i,j)+k) * mquant_x
```

where partial_inv is the partial dequantization, k=0 for intra, and k=1 for non-intra macroblocks. Final inverse quantization is defined by:

```
rec(i,j) = (partial_inv(i,j) * w(i,j)) / 16
```

where w=wI for intra, and w=wN for non-intra macroblocks. This reconstruction is then followed by normal clipping and adjustment of even reconstruction values as described in section 7.3. For the DC value of intra blocks, the reconstruction is obtained by calculating 8*QDC, for all scaling layers.

D.1.5 PROVISIONS FOR BANDWIDTH CONTROL OF RESOLUTION LAYERS

The Slave_slice layer specification includes the quantizer_delta parameter. This delta is always specified with reference to the corresponding quantizer scale factor used in the Slice layer, as explained in section 9.3.6. This parameter is used to derive the "mquant" values used in the slave layers. For the Test Model the "quantizer_delta" parameter is always zero.

D.1.6 CODING OF MOTION VECTOR INFORMATION

Motion vector data are part of the MB attributes which are coded in the Slice layer. Whereas the pixel data in the Slice of the scalable Test Model has a 1/16 resolution, the motion vector data are that of a full resolution video. Motion vector data are the same as that which would be derived for a non-scalable bitstream; no additional computations are required. To reconstruct resolution scales other than full resolution, these data must be scaled appropriately. For example, the x- and y-motion-vector components at 1/4 resolution are 1/2 the corresponding full resolution components. Similarly the 1/16 vector components are 1/4 the corresponding full resolution components. After scaling, the motion vector components should be rounded away from zero to achieve a resolution of 1/2 pixel.

D.1.7 VLC CODING

Coding of 2x2 and 4x4 DCT coefficients is through a set of new VLC tables, described below. For the final 8x8 layer, coefficients are coded (intra macroblocks included) by using the standard MPEG-1 tables 2-B.5c-g, without the special-case treatment of the first run/amplitude event (note that, in slave_slices, there may be blocks where all prediction differences and all unpredicted coefficients are zero; in those cases an EOB is the first and only event coded).

The variable-length coding scheme used for coding of DCT coefficients in the lower scales, is a modified JPEG-style. In the JPEG scheme, VLC codes are assigned to RUNS and SIZE combinations, followed by a fixed code of length SIZE. The combined VLC and fixed length codes are used to specify the exact amplitude. The entire (RUN, SIZE) codeword tables can be constructed from knowledge of the number of codewords of each length, and an ordered list of (RUN, SIZE) pairs. We provide a pseudo-C program segment to generate the codewords and illustrate its operation below. Since the maximum RUN is 15, we represent a (RUN, SIZE) pair by

```
VALUE = 16*RUN + SIZE.
```

Note that VALUE=0 is reserved for the End of Block (EOB) symbol. The numbers of codewords of each length and ordered lists of VALUEs are given below for tables used in Scales 2, 4, and 8.

```
SCALE 2 NUMBER OF CODES of EACH LENGTH:
ncodes_2[16] = {
   1,  1,  0,  2,  2,  2,  3,  0,
   3,  1,  0,  2,  2,  0,  0, 15 };
```

Thus, there is one codeword of length 1, one of length 2, none of length 3, etc.

```
SCALE 2 VALUES:
values_2[34] = {
  0x00, 0x01, 0x02, 0x11, 0x03, 0x21, 0x12, 0x31,
  0x04, 0x13, 0x22, 0x05, 0x14, 0x32, 0x23, 0x06,
  0x15, 0x16, 0x24, 0x33, 0x25, 0x07, 0x08, 0x10,
  0x17, 0x18, 0x26, 0x27, 0x28, 0x34, 0x35, 0x36,
  0x37, 0x38 };
```

Thus, the codeword of length one has VALUE=0 (i.e. EOB), the codeword of length two has VALUE=1 (i.e. RUN=0, SIZE=1), etc.

```
SCALE 4 NUMBER of CODES of EACH LENGHT:
ncodes_4[16] = {
   0,  1,  2,  3,  6,  4,  4,  4,
   6,  1,  2,  2,  0,  0,  0, 95 };

SCALE 4 VALUES:
values_4[130] = {
  0x00, 0x01, 0x31, 0x11, 0x21, 0x51, 0x02, 0x32,
  0x41, 0x61, 0x81, 0x91, 0x12, 0x52, 0x71, 0xA1,
  0x03, 0x22, 0x33, 0xB1, 0x13, 0x42, 0xC1, 0xE1,
  0x34, 0x53, 0x62, 0x92, 0xD1, 0xF1, 0x82, 0x04,
  0x23, 0x72, 0x14, 0x43, 0xA2, 0x35, 0x54, 0x93,
  0x63, 0xB2, 0x24, 0x83, 0xC2, 0x05, 0x73, 0xE2,
  0x15, 0x94, 0xD2, 0x36, 0x55, 0x06, 0xF2, 0x07,
  0x08, 0x10, 0x16, 0x17, 0x18, 0x25, 0x26, 0x27,
  0x28, 0x37, 0x38, 0x44, 0x45, 0x46, 0x47, 0x48,
  0x56, 0x57, 0x58, 0x64, 0x65, 0x66, 0x67, 0x68,
  0x74, 0x75, 0x76, 0x77, 0x78, 0x84, 0x85, 0x86,
  0x87, 0x88, 0x95, 0x96, 0x97, 0x98, 0xA3, 0xA4,
  0xA5, 0xA6, 0xA7, 0xA8, 0xB3, 0xB4, 0xB5, 0xB6,
  0xB7, 0xB8, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8,
  0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xE3, 0xE4,
  0xE5, 0xE6, 0xE7, 0xE8, 0xF3, 0xF4, 0xF5, 0xF6,
  0xF7, 0xF8 };
```

```
-----------------------------------------------------------------------
PSEUDO-C CODE TO GENERATE CODEWORD VALUES:

  valpt = values_2;  /* Point to first value in list of values */
  ncodes = ncodes_2;
  code=0;            /* Initialize Huffman code value */
  for (length=1; length<=16; ++length) {
    code = (code<<1) + 1;
    for (i=1; i<=ncodes[length-1]; ++i) {
      ++valpt;
      --code;
    }
  }
```

Note: The list of "code"s generated in this fashion are converted to binary codewords by 1) expanding "code" as an unsigned binary number of length equal to the "length" of code, and 2) bit-wise complementing the resulting binary codeword.

Example from Scale 2 data above:

SCALE 2 CODE WORD TABLE

VALUE	RUN	SIZE	CODE_LEN	BINARY CODE_WORD
0x 0	0	0	1	0
0x 1	0	1	2	10
0x 2	0	2	4	1100
0x11	1	1	4	1101
0x 3	0	3	5	11100
0x21	2	1	5	11101
0x12	1	2	6	111100
0x31	3	1	6	111101
0x 4	0	4	7	1111100
0x13	1	3	7	1111101
0x22	2	2	7	1111110
0x 5	0	5	9	111111100
0x14	1	4	9	111111101
0x32	3	2	9	111111110
0x23	2	3	10	1111111110
0x 6	0	6	12	111111111100
0x15	1	5	12	111111111101
0x16	1	6	13	1111111111100
0x24	2	4	13	1111111111101
0x33	3	3	16	1111111111110000
0x25	2	5	16	1111111111110001
0x 7	0	7	16	1111111111110010
0x 8	0	8	16	1111111111110011
0x10	1	0	16	1111111111110100
0x17	1	7	16	1111111111110101
0x18	1	8	16	1111111111110110
0x26	2	6	16	1111111111110111
0x27	2	7	16	1111111111111000
0x28	2	8	16	1111111111111001
0x34	3	4	16	1111111111111010
0x35	3	5	16	1111111111111011
0x36	3	6	16	1111111111111100
0x37	3	7	16	1111111111111101
0x38	3	8	16	1111111111111110

D.2. IMPLEMENTATIONS

D.2.1 DECODER

A block diagram of the scalable Test Model decoder is shown in Figure D.1. The 3-layer decoder in the scalable of the Test Model supports 2x2 (low), 4x4 (medium), and 8x8 (high) resolution scales. After demultiplexing and entropy decoding, there will be for every 8x8 block, corresponding 2x2 and 4x4 blocks, all of which are necessary to build the final 8x8 matrix of DCT coefficients. The low resolution 2x2 blocks are used as a prediction to the four lowest order coefficients of their corresponding 4x4 blocks. Similarly, the 4x4 blocks are used as prediction to the 16 lowest order coefficients of their corresponding 8x8 blocks.

Because the same quantization matrix, Q8, is used for all hierarchical layers, the low resolution DCT coefficient data are only partially dequantized as we build the target resolution set of coefficients. Dequantization by Q8 is only needed once we reach the target resolution (an additional simplification can be achieved in the Test Model. Since the quantizer_delta parameter is always set to zero, dequantization by the quantizer_scale parameter is needed only at the target resolution. In this case, no additional multiplies are required over those required in the non-scalable syntax).

Once all 8x8 blocks in a macroblock have been dequantized, they can be used to reconstruct their corresponding pixels by the same techniques used in the non-scalable Test Model. In this regard, 16x16 motion compensation prediction and interpolation can be used.

For a decoder operating at a lower than 8x8 resolution, only the required DCT coefficients are rebuilt. Except for the use of IDCTs other than the 8x8 IDCT, and motion compensation for scaled macroblocks of 4x4 or 8x8, the reconstruction method is the same as with full resolution video. Of special note is that motion vectors need to be scaled as explained before.

D.2.2 ENCODER

In summary, the Test Model encoder is a simple one. In this encoder the 2x2, and 4x4 DCT data are simply extracted from the corresponding quantized coefficients in the 8x8 data. The prediction differences of the frequency pyramid are, therefore, always zero.

Because there is no feedback loop in the lower resolution scales, this encoder will result in accumulation of quantization and motion compensation errors at these resolution scales. The error, however will be naturally reset back to zero whenever a new Group of Frames starts. This limitation can be overcome by more complex encoders.

D.2.2.1 RATE CONTROL AND MQUANT SELECTION

These features are implemented in the same manner as with the non-scalable Test Model, i.e. rate control and mquant are implemented counting the bits generated by each macroblock together with its corresponding slave_macroblocks; the quantizer_scale of the macroblock layer is then set using the mquant value of the full resolution macroblock.

D.2.2.2 FRAME/FIELD CODING, COMPATIBLE, AND OTHER MACROBLOCK TYPES

Only frame prediction and coding are permitted for experiments with scalability. Other macroblock-type decisions are made using the full resolution video, just as with the non-scalable Test Model (except for the above restrictions). Motion vector estimation is performed only for the full resolution video. The MB type, address, and coded block pattern are defined with the full resolution data, exactly as in the non-scalable Test Model; these attributes are coded, however, together with the lowest resolution layer and used throughout all resolution scales.

Appendix E : Extensions for Purely Field Based Coding

TM0(Field)
*SM3 is simply applied to Rec 601(each fields), 1GOP=24fields(60),21fields(50)
BBIBBPBBPBBPBBPBBPBBP(BBP)

*P-pictures are predicted by previous 2P(I)-pictures

```
                forwerd
   o(e)    P ----> P
                        /different(field parity)
   e(o)        P-/
```

VLC(syntax) is basically same as B-pictures, but only names of prediction mode are changed.
Backward -> Dfferent
Bi-directionally -> Average=(For+Diff)/2

*ME :Full search or Hierachical Method
Range:+/-31(not depend on the field distance)
Accuracy:half-pel

SM3Fi++(optional)
*B-pictures are predicted by 3P(I)-picture

```
                For.  Back.
   o(e)    P-->B<------P
                    \Diff.
   e(o)             P
```

*MB Type (noMC's are rejected)

Pred.Type	For	Diff	Back	VLC non-coded	VLC coded
F+B	1	0	1	10	11
F+D	1	1	0	0100	0101
D+B	0	1	1	0110	0111
F	1	0	0	0010	00010
B	0	1	0	0011	00011
D	0	0	1	00001	000001
Intra	0	0	0	-	0000001

Optimizing for field base coding (example)
Format 4:2:0 —> 4:2:2
MBsize 16*16 —> 16*8
Scanning for Y

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

Appendix F: Cell loss experiments

F.1 Cell loss

Cell loss can occur unpredictably in ATM networks. This document proposes a method of simulating cell loss. A specification for a packetized bitstream has been defined. A model of bursty cell loss is defined and analysed in order to allow the simulation of bursty cell loss. The proposed specification and model are simplified; no attempt is made to model actual ATM networks; the main objective of the model is to allow consistent simulation of the effects of cell loss on video coding.

F.1.1 Bitstream specification

The coded bit stream is packetized into 48 byte cells consisting of a four bit sequence number (SN), a four bit sequence number protection field (SNP) and 47 bytes of coded data. In the stored file each cell is preceded by a Cell Identification byte (CI). The syntax is as follows:

< CI ><SN><SNP>< 47 bytes of data >

The CI byte consists of the bit string '1011010' followed by the priority bit. The priority bit is set to '1' for low priority cells and '0' for high priority cells. The cell loss ratio for low priority cells may be different to that for high priority cells. SN is incremented by one after every cell. The sequence number protection is set to zero.

For a lost cell the cell is discarded.

F.1.2 Calculation of cell loss probabilities

This section outlines a method for determining whether any cell in a bitstream should be marked as lost. Cell loss is assumed to be random, with the probability of cell loss depending only on whether the previous cell of the same priority was lost.

Firstly the mean cell loss rate and the mean burst of consecutive cells lost is calculated from the probabilities of cell loss. These equations are then rearranged in order to express the cell loss probabilities in terms of the mean cell loss rate and the mean burst of consecutive cells lost.

The following notation is used. The probability that any cell is lost is given by P, the probability that a cell is lost given that the previous one was not lost is given by P_n and the probability that a cell is lost given that the previous one was lost is given by P_l. These probabilities are illustrated in the tree diagram below.

```
                                            |------ Lost Cell
                                         Pl |
                 |------- Lost Cell --------|
                 |                     1-Pl |
               P |                          |---- Not Lost Cell
Unknown          |
Previous   ------|
State            |
             1-P |                          |------ Lost Cell
                 |                       Pn |
                 |------ Not Lost Cell -----|
                                       1-Pn |
                                            |---- Not Lost Cell
```

F.1.3 Calculation of mean cell loss rate

The mean cell loss probability is given by P. In this section a relationship between P, P_n and P_l is derived, as follows, by finding two equivalent expressions for the probability of a given cell being lost. A lost cell can occur in two ways: immediately after a cell has been lost and after a cell has been received. The probability that a cell is lost, P, is the sum of the probability that the cell is lost given the previous cell was lost multiplied by the probability that the previous cell was lost, $P * P_l$, and the probability that the cell is lost given the previous cell was not lost multiplied by the probability that the previous cell was not lost, $(1 - P) * P_n$. So,

$$P = P * P_l + (1 - P) * P_n$$

So

$$P = P_n / (1 - P_l + P_n)$$

(1)

F.1.4 Calculation of mean burst of consecutive cells lost

A burst of lost cells is defined as a sequence of consecutive cells all of which are marked as lost. It is preceded by and followed by one or more cells that are marked as not lost. The length of the burst of lost cells is defined as the number of cells in a burst that are marked as lost. The mean burst of consecutive cells lost is defined as the mean burst length. This number must always be greater than or equal to one.

A burst starts when a cell is lost after one or more cells have not been lost. The probability that this is a burst of length one is equal to the probability that the next cell is not lost, that is, $1 - P_l$. The probability that this is a burst of length two is equal to the probability that the next cell is lost and the one after that is not lost, that is, $P_l * (1 - P_l)$. The probability of a burst of length n is $P_l^{(n-1)} * (1 - P_l)$. The mean burst length, B, is therefore given by:

$$B = (1 - P_l) + 2 * P_l * (1 - P_l) + 3 * P_l^2 * (1 - P_l) + \dots$$

Summing this series leads to the result:

$$B = 1 / (1 - P_l)$$

(2)

F.1.5 Calculation of cell loss probabilities

Rearranging equation (2) gives:

$$P_l = 1 - 1 / B \quad (3)$$

Rearranging equation (1) gives:

$$P_n = P * (1 - P_l) / (1 - P)$$

Using equation (3) gives:

$$P_n = P / (B * (1 - P)) \quad (4)$$

F.1.6 Simulation of cell loss

Equations (3) and (4) allow the probabilities of cell loss to be calculated from the average cell loss rate and the mean length of bursts of lost cells. Cell loss can easily be simulated using these probabilities: assume that the first cell is received, then the probability that the next will be lost is given by P_n. The probability that a cell is lost is always P_n, unless the previous cell was also lost in which case the relevant probability is P_l.

A simulation of cell loss only needs a random number generator, the values of P_n and P_l and the knowledge of whether the previous cell of the same priority was lost or not. Pseudo-Pascal code to perform cell loss is given below. Random is a function that returns a random number between zero and one; its implementation is given below.

```
PreviousCellLost := FALSE;
Write('Enter mean cell loss rate and burst length');
Readln(P,B);
PL := 1 - 1/B
PN := P / (B * (1-P) )

For CellCount := 1 To NumberOfCells DO
  BEGIN
    CASE PreviousCellLost OF
      TRUE  :  IF Random < PL THEN CellLost := TRUE
                              ELSE CellLost := FALSE;
      FALSE :  IF Random < PN THEN CellLost := TRUE
                              ELSE CellLost := FALSE;
      END;
    Write(CellLost);
    PreviousCellLost := CellLost;
    END;
END.
```

If the priority bit is used then the cell loss generator must be implemented separately for each of the priorities.

F.1.7 Random number generation

To ensure the consistent simulation of cell loss, it is necessary to ensure that the same sequence of random numbers is generated by all simulations regardless of the machine or programming language used. This section describes a method for the generation of such random numbers.

Random numbers are generated by use of a 31 bit shift register which cycles pseudo-randomly through (2^31 - 1) states (the value of zero is never achieved). The shift operation is defined by the pseudo-Pascal code below.

```
DO 31 times
Begin
  Bit30  := (ShiftRegister & 2^30) DIV 2^30
  Bit25  := (ShiftRegister & 2^25) DIV 2^25
  ShiftRegister := (2*ShiftRegister MOD 2^31) + (Bit30 XOR Bit25);
End
```

To generate a random number, the shift register is first shifted as above and then divided by (2^31 - 1). It may be easier to use it as it is, and multiply the probabilities in the program above by (2^31 - 1).

A separate random number generator is used for low and high priority cell loss. For each, the shift register is initialised to a value of 1 and is then shifted 100 times. If this is not done, the first few random numbers will be small, leading to the loss of the first cells in the bitstream.

F.2 Parameters

This section suggests specific values of the parameters to allow consistent simulation of the effects of cell loss on video coding.

The cell loss experiment will use a mean cell loss rate of 1 in 1000 and a mean burst length of 2. Only low priority cells are lost. The following formula gives the value of P to use for low priority cells.

$$P = 10^{-3} \times \frac{\text{Total Bit rate}}{\text{Total bitrate - Bit rate for high priority cells}}$$

For example:

Total bit rate 4Mbits/s

High priority bit rate 2Mbits/s (50% of Total)

then the mean cell loss rate figure for the cell loss simulation program is 2×10^{-3}.

Other cell loss experiments at different cell loss rates can also be shown.

For all experiments the following table should be completed.

		High priority bit rate	Low priority bit rate
1-layer			
2-layer	base		
	enhance		

CONTENTS